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

October 7, 2021

Abstract

The Poisson multinomial distribution (PMD) can be considered as a generalization of multinomial distribution in a way that the probability vectors for multinomial trials are of same length but differ in values. The PMD is useful in many areas such as, voting theory, machine learning and uncertainty quantification. The distribution functions of PMD, however, is usually difficult to compute. In this paper, we develop algorithms to compute the probability mass function (pmf) for PMD using methods of Fourier transformation, normal approximation and simulation. We implement the algorithms into an R package so that users can calculate the pmf of PMD 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 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

Suppose there are n independent multinomial trials, each trial has same categories but with different probabilities, then the sum of outcomes from the n trials follows a probability distribution and we call it Poisson multinomial distribution. For example, there are n independent balls, and one needs to throw them into m different bins. Each ball has its own probabilities of falling into a specific bin. Then the probability distribution of the ball counts in each bin is defined as Poisson multinomial distribution (PMD). PMD is a generalization of multinomial distribution. In addition, when $m = 2$ the PMD reduces to the Poisson binomial distribution.

PMD has applications in many fields including computer science, probability, statistics and political science. It has applications in game theory (Cheng, Diakonikolas, and Stewart 2017), digital imaging (Akter, Moon, and Kwon 2019), machine learning (Kamath 2014), etc. One of the intriguing areas 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 different voting behavior. Mostly, people pay attention to the result of an election while statisticians focus on the probability for each electoral outcome to happen. An 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 interest.

Additionally, in soft classification context, people introduce randomness in decision making process. The process can be characterized by PMD. Suppose a machine learning model computes probabilities of an observation falling into different categories, this is actually equivalent to making one draw from a multinomial distribution. Thus PMD is a perfect tool to characterize the situation of classifying multiple observations since it is as same as making multiple draws from different independent multinomial distributions.

In ecological inference, consider making statistical inference to aggregated data. An aggregated dataset is to separate data points into groups and consider the groups as new data points. Let's assume the data response variable is categorical. In the grouped dataset, we can count the number of each category in each group and take it as our new response variable and this variable will follow PMD accordingly. All the areas above that involve PMD also require computation of its probability mass function. Without the pmf, we are unable to draw any conclusion over the discussed examples.

Enumeration can be a way because if $n \times m$ is small we can list all outcomes by hand. However, when n and m increase, 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. Not only in the fields mentioned above, computing pmf is frequently required in many scenarios that PMD are involved. There is no existing algorithm for computing pmf for PMD. Hereby we are motivated to build one that can calculate pmf of PMD efficiently.

1.2 Related Literature and Contribution of This Work

Some former studies have uncovered PMD's structure and some of its properties. In 2015, Daskalakis, Kamath, and Tzamos (2015) proved that PMD is ϵ-cover, which means there exists a set of distributions small enough to cover the set of all PMDs. PMD can also be ϵ-close so that it can be approximated by other distributions. The paper also reveal that exists learning algorithms for PMDs. This result shows that we can find a sampling scheme to draw enough samples out of a PMD to depict its structure and complexity. Simply saying, we could study PMD's probability mass function via sampling. At almost the same time, Diakonikolas, Kane, and Stewart (2016) obtained a different understanding of the structures of PMDs' using Fourier transformation and disclosed the sparsity of PMD. The paper also provided the first computationally efficient for PMDs and showed central limit theorem (CLT) can be applied to PMD. The most important statistical characteristic of PMD, probability mass function has not been explored yet. Hong (2013b) consider the exact and approximate methods for computing the pmf of the Poisson binomial distribution. Zhang, Hong, and Balakrishnan (2018) introduce the general Poisson binomial distribution and develop an algorithm to compute its distribution functions.

Current studies have shown that PMDs can be applied to many fields. In game theory, based on the structure and properties of PMD, Diakonikolas, Kane, and Stewart (2016) constructed a efficient scheme to approximate Nash equilibria in anonymous games and Cheng, Diakonikolas, and Stewart (2017) proved that any n-player anonymous games can have approximate Nash equilibria in polynomial time. Akter, Moon, and Kwon (2019) introduced PMD in image processing field for the first time. They used PMD to count the encrypted image data and computed PMD's probability mass functions to obtain optimal results. However, the probability mass function of PMD is computed by using simplified multinomial distributions which is actually an approximation method.

In this paper, the contributions are following. The main contribution of this paper is constructing an exact method that uses fast Fourier transformation algorithm (FFT) to compute pmf of PMD. We call this method DFT-CF. Besides, we also construct two approximation methods to compute pmf of PMD, the one is a simulation scheme and the other one is approximation based on CLT. Along the journey of constructing methods, several properties and theorems are built.

Specifically, the DFT-CF method applies multidimensional discrete Fourier transformation to the characteristic function (cf) of PMD to obtain its pmf. The method NA uses CLT to approximate pmf of PMD. We prove an analytic error bound for this method based on its CLT converge rate. The SIM method is the other approximation method. SIM uses multiple multinomial distributions to simulate PMD and obtains pmf via simulation results. An expected absolute error bound of this method is provided and proved.

We also studying accuracy and time efficiency of each methods under different scenarios. Since there is no way to compute true pmf for PMD, the logic of accuracy study is as following. We explore the situations under which we are able to compute the pmf of PMD and construct accuracy test for DFT-CF. Then we use DFT-CF as our true pmf to conduct accuracy tests for other methods. For time efficiency, we only test the DFT-CF method since it is the only method that automatically compute the whole pmf while others only compute partial pmf (eg. a single probability mass point). We give $m = 2, 3, 4, 5$ and let n grow sufficiently large while we record the time DFT-CF takes to finish computing.

Accuracy test for DFT-CF is conducted by using binomial, Poisson binomial and small dimensional PMDs. It suffices to show DFT-CF is pretty accurate and the error comes from the fast Fourier transformation algorithm. Due to computational limitation, accuracy test for SIM is done by setting $m = 3$ and n to be moderately large. Result shows that the method is accurate and the error bound can be controlled via improving simulation repeating times. Through setting $m = 3, 5, 7$ and letting n be sufficiently large, our accuracy test support NA's accuracy when n is moderate and large.

Then we separate all PMDs into different subsets according to their dimensions and give out recommendations with respect to which method to be chosen. Based on our developed methods, we make applications in the fields of ecological inference, political science and classification.

In ecological inference, people separate the data into multiple groups and consider each group as a new data point, this is also called aggregated data. We use the Predictive Maintenance Dataset Data Set (Dua and Graff 2017) from UCI machine learning repository to conduct the application of PMD in ecological inference, the dataset is separated into several groups to form aggregated data and a logistic type model is build.

We also provide an example of applying PMD in voting scenario. The probability of a voting result can be computed via PMD. Thus PMD provide a new perspective of predicting electoral results.

We illustrate an application of PMD in soft classification context using Electroluminescence (EL) image data. We show that PMD is a good tool to describe the distribution of the counts in confusion matrix.

We build an R package to contain all methods we develop and demonstrate the usage of the package.

1.3 Overview

The rest of the paper is organized as follows. In section 2, we describe the mathematical definition of Poisson Multinomial distribution and list some of its statistical properties. In section 3, we demonstrate the details of the three methods that we develop to compute PMD's pmf. The section 4 studies the curacy and time efficiency characteristics of the three methods, we also give out recommendations of preferred situations for each method. In the section 5, we illustrate application of our methods via three examples respectively in the fields of voting theory, ecological inference and soft classification. In the section 6, we introduce our R package that implemented with the methods and the section 7 concludes the paper and lists prospective research areas.

2 Poisson Multinomial Distribution

2.1 Definition of the Distribution

Suppose there are n mutually independent multinomial experiments with the following properties. Each of the multinomial experiments only has one trial and the outcome will be one of m categories. Let $\boldsymbol{I}_i = (I_{i1}, \ldots, I_{im})', i = 1, \ldots, n$ be random indicator vectors to denote the outcomes of the multinomial experiments described above. Also, let the associated probability vectors be $\boldsymbol{p}_i = (p_{i1}, \ldots, p_{im})'$. Correspondingly, for a fixed i we have $\sum_{j=1}^{m} p_{ij} = 1$ and $\sum_{j=1}^{m} I_{ij} = 1$.

The we say the sum $\boldsymbol{X} = (X_1, \ldots, X_m)' = \sum_{i=1}^{n} \boldsymbol{I}_i$ follows a PMD. We denote the distribution as

$$\boldsymbol{X} \sim \text{PMD}(\mathbf{P}).$$

where $X_j = \sum_{i=1}^{n} I_{ij}, j = 1, \ldots, m$. The matrix $\mathbf{P}$ is called success probability matrix (SPM).

$$\mathbf{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)'.$$

Note that the random variables, $X_1, \ldots, X_m$ satisfy the following constraint

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

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

Let $\boldsymbol{x} = (x_1, \ldots, x_m)'$ be a realization of a PMD random variable $\mathbf{X}$. The probability mass function (pmf) of PMD is defined as,

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

Here we give a simple example in election to illustrate the calculation of the pmf by hand.

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

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

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 example, the probability of a result that the first candidate gets 4 votes and others gets 0 vote, that is,* $\boldsymbol{x} = (4, 0, 0)'$.

$$P\left(\mathbf{X} = \boldsymbol{x}\right) = 0.1 \times 0.5 \times 0.4 \times 0.8 = 0.016.$$

Also, if we take $\boldsymbol{x} = (1, 3, 0)'$, *the probability of* $\mathbf{X} = \boldsymbol{x}$ *is*

$$\begin{aligned} P\left(\mathbf{X} = \boldsymbol{x}\right) =& 0.1 \times 0.2 \times 0.5 \times 0.1 + 0.5 \times 0.2 \times 0.5 \times 0.1 \\ &+ 0.4 \times 0.2 \times 0.2 \times 0.1 + 0.8 \times 0.2 \times 0.2 \times 0.5 = 0.0236. \end{aligned}$$

When the 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 $x_1 + \cdots + x_m = n$.

For some special cases, when the SPM is identical across all rows, that is, I_i, $i = 1, \ldots n$ are identically distributed, the distribution of X can be simplified as multinomial distribution. Hence, the PMD is a generalization of the multinomial distribution. When $m = 2$, the PMD is reduced to the Poisson binomial distribution.

In related literature, Hong (2013b) consider the exact and approximate methods for computing the pmf of the Poisson binomial distribution. Zhang, Hong, and Balakrishnan (2018) introduce the general Poisson binomial distribution and develop an algorithm to compute its distribution functions.

2.2 Properties of the Distribution

Now let us study some basic properties of Poisson multinomial distribution.

Proposition 1 *Suppose* $\mathbf{X} \sim$ *PMD*$(\mathbf{P})$*, the mean of* $\mathbf{X}$ *is*

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

where $p_{\cdot k} = \sum_{i=1}^{n} p_{ik}$.
The variance-covariance matrix of $\mathbf{X}$ *denoted by* $\mathbf{\Sigma}$ *is an* $m \times m$ *matrix that has entries* σ_{ij} *defined as*

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

For $\boldsymbol{x} = (x_1, \ldots, x_m)'$*, denote the probability* $\Pr(\mathbf{X} = \boldsymbol{x})$ *as* $p(x_1, \ldots, x_m)$*. The characteristic function (cf) for the PMD is*

$$\phi_{\boldsymbol{x}}(t_1, \ldots, t_m) = \sum_{x_1=0}^{n} \cdots \sum_{x_m=0}^{n} p(x_1, \ldots, x_m) \exp\left(\boldsymbol{i} \sum_{j=1}^{m} t_j x_j \right).$$

where $\boldsymbol{i} = \sqrt{-1}$.

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

Notice that $\sum_{j=1}^{m} X_j = n$, without loss of generality, let $\boldsymbol{X}^* = (X_1, \ldots, X_{m-1})'$ with corresponding $\boldsymbol{x}^* = (x_1, \ldots, x_{m-1})'$ and $\mathbf{P}^*$ equals to the first $m-1$ columns of $\mathbf{P}$. That is to define

$$\mathbf{P}^* = (\boldsymbol{p}_1^*, \ldots, \boldsymbol{p}_n^*)'.$$

where $\boldsymbol{p}_i^* = (p_{i1}, \ldots, p_{i,m-1})', i = 1, \ldots, n$.

Then we denote the non-singular $(m-1) \times (m-1)$ covariance matrix of $\boldsymbol{X}^*$ as $\mathbf{\Sigma}^*$. We have

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

where $\boldsymbol{p}_i^*$ is the ith row of $\mathbf{P}^*$. Also, we denote the mean of $\boldsymbol{X}^*$ as μ^*, then

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

In addition, the cf of $\boldsymbol{X}^*$ is

$$\phi_{\boldsymbol{x}^*}(t_1, \ldots, t_{m-1}) = \sum_{x_1=0}^{n} \cdots \sum_{x_{m-1}=0}^{n} p(x_1, \ldots, x_{m-1}) \exp\left(\boldsymbol{i} \sum_{j=1}^{m-1} t_j x_j \right).$$

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

$$\Pr(\boldsymbol{X} = \boldsymbol{x}) = \Pr(\boldsymbol{X}_1 = \boldsymbol{x}_1, \ldots, \boldsymbol{X}_S = \boldsymbol{x}_S) = \prod_{k=1}^{S} \Pr(\boldsymbol{X}_k = \boldsymbol{x}_k).$$

To show this, we assume $\mathbf{P}$ is $n \times m$ and $\mathbf{P}_k$ is $n_k \times m_k$. $\sum_{k=1}^{S} n_k = n$ and $\sum_{k=1}^{S} m_k = m$. 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, $k = 1, \ldots, S$, 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 multidimensional 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 (Frigo and Johnson 2005).

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

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

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

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

By definition,

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

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

$$\mathbb{E}\left[\exp\left(\boldsymbol{i}\sum_{i=1}^{n}\sum_{j=1}^{m-1} t_j I_{ij}\right)\right] = \mathbb{E}\left[\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j I_{1j} + \cdots + \boldsymbol{i}\sum_{j=1}^{m-1} t_j I_{nj}\right)\right]. \qquad (4)$$

$$= \prod_{i=1}^{n} \mathbb{E}\left[\exp\left(\boldsymbol{i}\sum_{j=1}^{m-1} t_j I_{ij}\right)\right] = \prod_{i=1}^{n}\left[(1 - \sum_{j=1}^{m-1} p_{ij}) + \sum_{j=1}^{m-1} p_{ij}\exp(\boldsymbol{i}t_j)\right].$$

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

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

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

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

where

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

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

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

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

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 $\boldsymbol{X} \sim$ PMD is singular, we use the reduced PMD $\boldsymbol{X}^*$ to establish the normal approximation. From Daskalakis, Kamath, and Tzamos (2015) 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 $\mathbf{X} = (X_1, \ldots, X_m)'$ that has a reduced mean vector $\boldsymbol{\mu}^* = (p_{\cdot 1}, \ldots, p_{\cdot, m-1})'$ and non-singular reduced covariance matrix $\boldsymbol{\Sigma}^*$. For each possible outcome $\boldsymbol{x}_r$, and its neighbourhood interval $N_{\boldsymbol{x}_r} = [\boldsymbol{x}_r - 0.5, \boldsymbol{x}_r + 0.5]$. 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*

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

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

By central limit theorem (CLT),

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

where $\boldsymbol{p} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{p}_i$. To show Theorem 1, notice

$$\boldsymbol{X}^* = (X_1, \ldots, X_{m-1})' = \sum_{i=1}^{n} Ivec_i^* = \sum_{i=1}^{n} (I_{i1}, \ldots, I_{i,m-1})'$$

Intuitionally, $\boldsymbol{I}_i^* - \mathbb{E}(\boldsymbol{I}_i^*)$ has mean 0. Hereby $\boldsymbol{X}^* - \mathbb{E}\boldsymbol{X}^* = \sum_{i=1}^{n}(\boldsymbol{I}_i^* - \mathbb{E}\boldsymbol{I}_i^*)$ also has mean 0. The covariance of $\boldsymbol{X}^*$ is $\boldsymbol{\Sigma}^* = CC'$.
By extended Berry-Esseen theory (Raič 2019), there existing a constant c such that

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

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

3.3 Simulation-Based Method

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

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

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

Step 3 count the frequency of $\boldsymbol{x}$ showing up in $\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(B)}$ 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 $\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(B)}$.

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

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

The following theorem studies the expected error bound of the simulation method by given $\mathbf{P}$ and 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 $\boldsymbol{x}_r, r = 1, \ldots, N$. Let the probability for a specific result $\boldsymbol{x}_r$ be $p_{\boldsymbol{x}_r}$, $i = 1, \ldots, N$. The estimate of $p_{\boldsymbol{x}_r}$ using simulation method is $\widehat{p}_{\boldsymbol{x}_r}$. We have the following expected error given by Central Limit Theory,*

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

The expected total absolute error,

$$\sum_{r=1}^N \mathbb{E}|p_{\boldsymbol{x}_r} - \widehat{p}_{\boldsymbol{x}_r}| \le \sqrt{\frac{2(N-1)}{\pi B}}.$$

For any result $\boldsymbol{x}_r, i = 1, \ldots, N$. Consider a Bernoulli rv Y_r with probability $p_{\boldsymbol{x}_r}$ to be 1, and $1 - p_{\boldsymbol{x}_r}$ to be 0. By repeating the trail for B times, we obtain iid random variables $Y_r^{(1)}, \ldots, Y_r^{(B)}$. By WLLN

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

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

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

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

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

The equality will be achieved if $p_{\boldsymbol{x}_1} = \cdots = p_{\boldsymbol{x}_N}$. 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 $\chi = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$. MAE is defined as following,

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

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

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

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 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 particular 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 Hong (2013b), 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 2×2 to 6×4 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.

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 Hong (2013b) 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

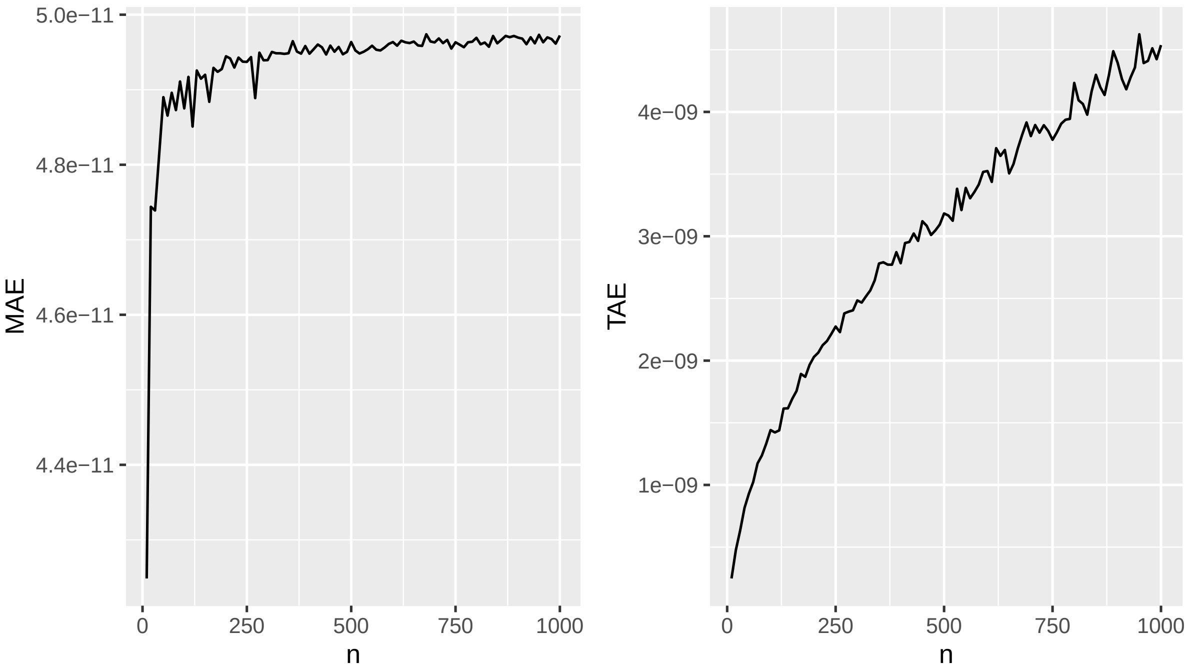

Figure 1: As n increasing to 1000, the MAE is around 10^{-11} and the TAE is around 10^{-9}.

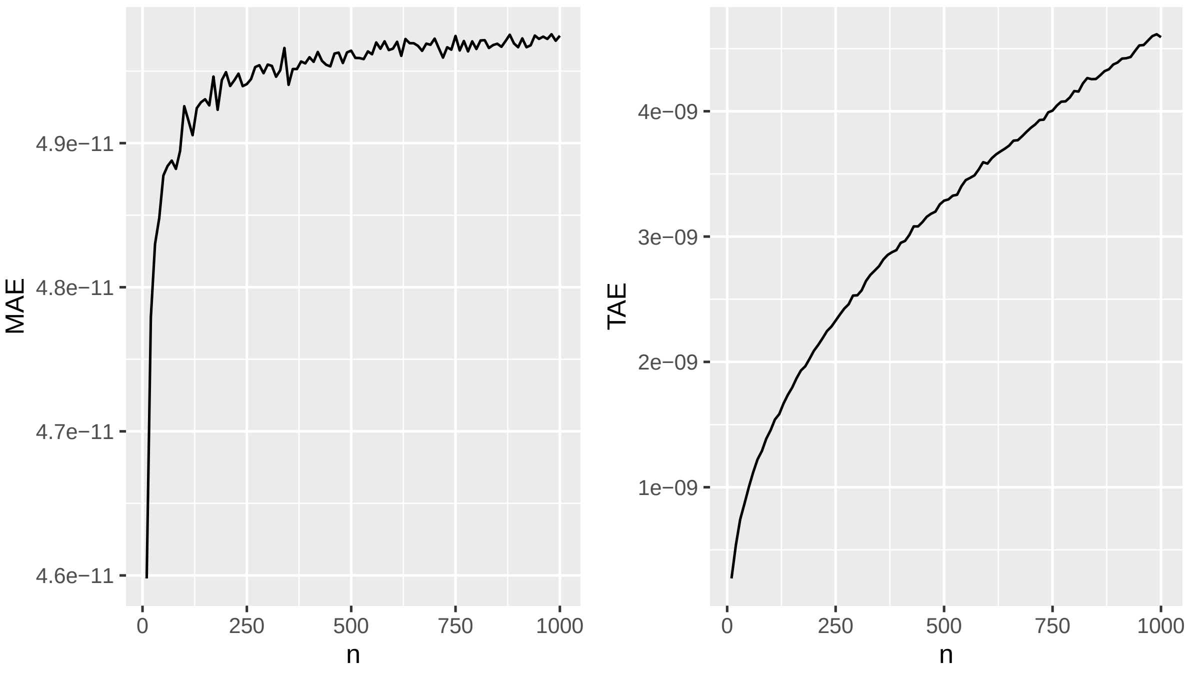

Figure 2: Accuracy result of DFT-CF under Poisson binomial scenario.

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_{true}. MAE and TAE for normal approximation are given by

$$\text{MAE} = \max_{\boldsymbol{x} \in \mathcal{X}} |p_{\text{NA}}(\boldsymbol{x}) - p_{\text{true}}(\boldsymbol{x})|,$$

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 Figures 3 and 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 the Original to express the sparsity of PMD as n and m increase. The Original is just to estimate all probability mass points by 0's. 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.

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 $x_j, j = 1, \ldots, m-1$ 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). 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

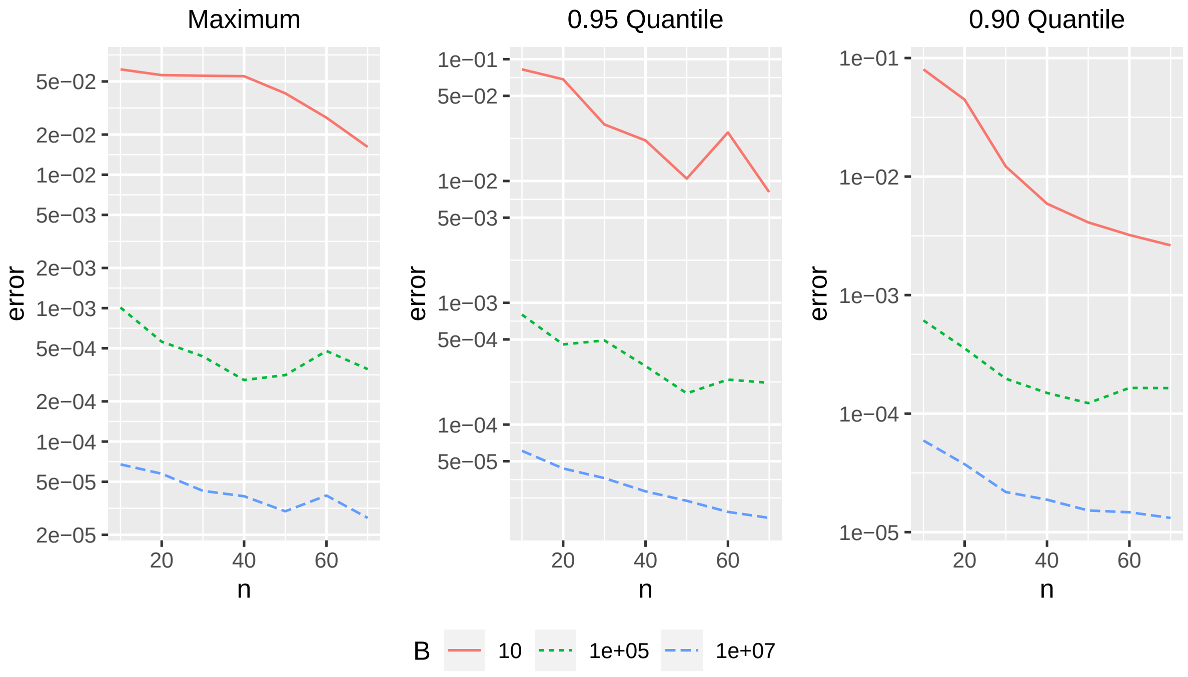

Figure 3: Accuracy of SIM method based on quantile of PMD when $m = 3$.

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.

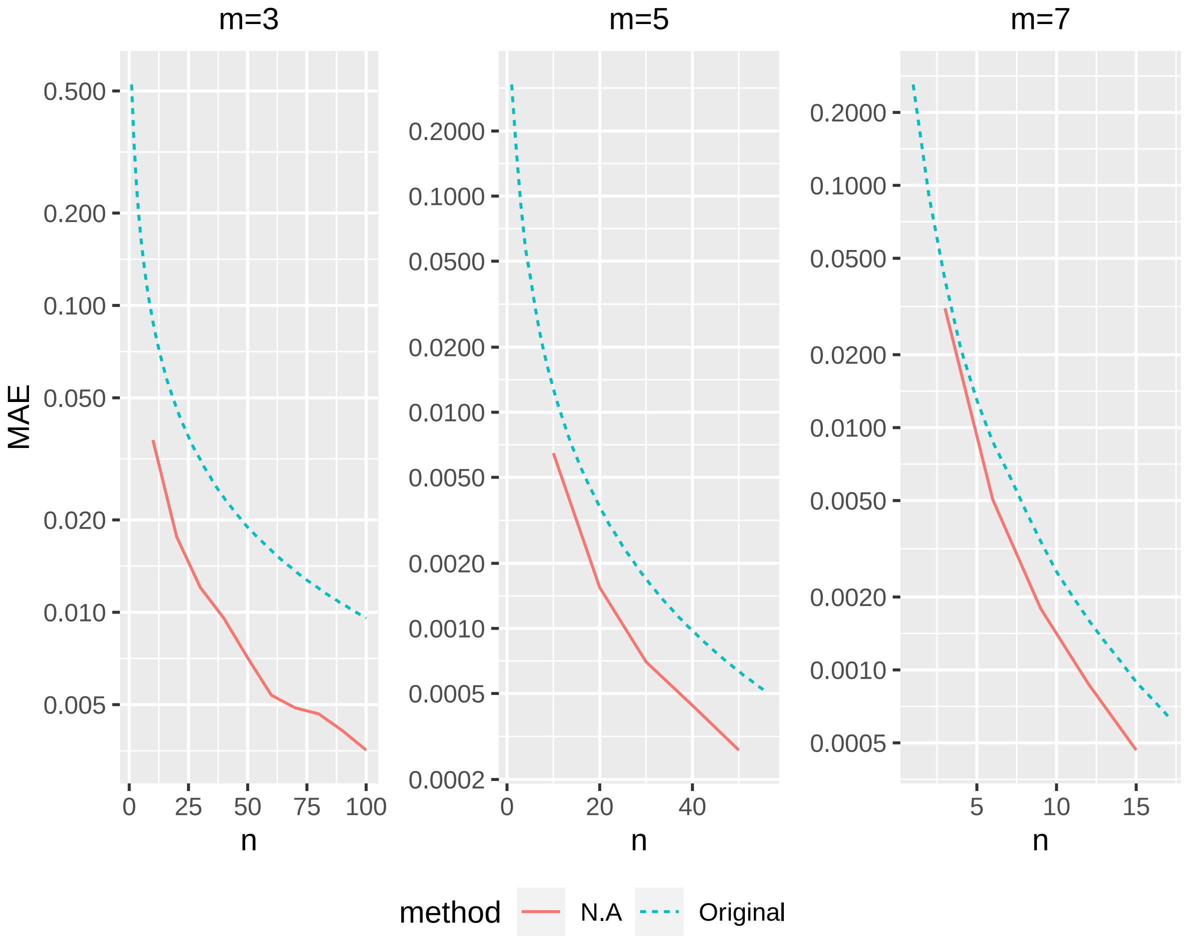

Figure 4: MAE of NA compared with baseline (Original).

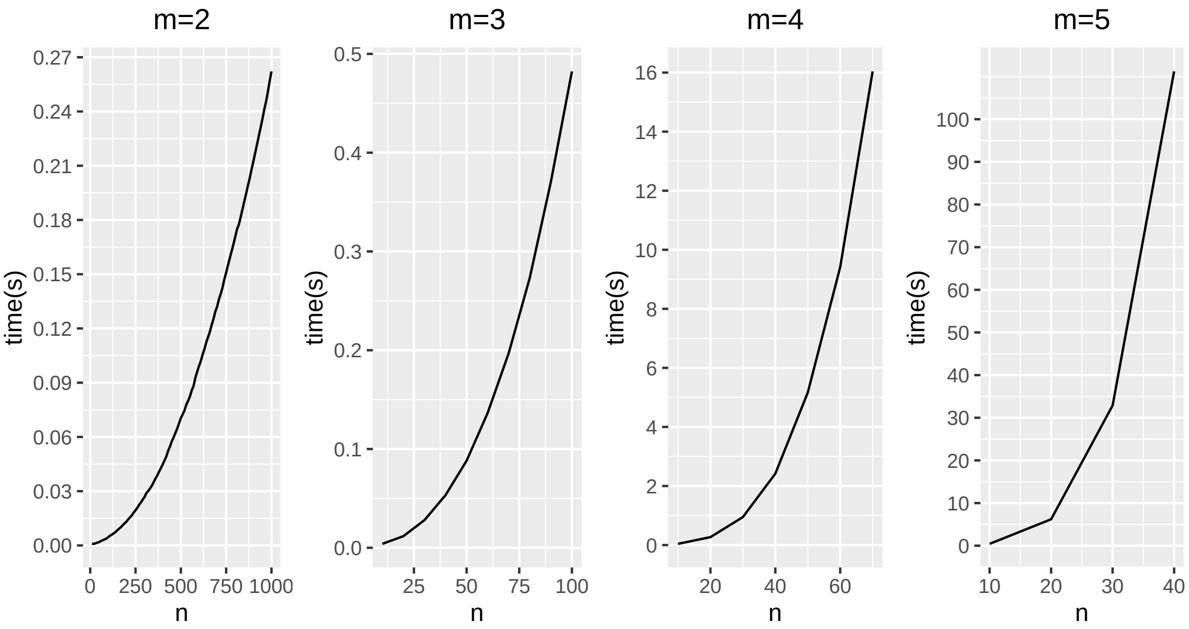

Figure 5: Time efficiency study of DFT-CF.

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 $\boldsymbol{x}_r, r = 1, \ldots, N$, respectively. Each $\boldsymbol{x}_r$ 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

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

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 $\boldsymbol{x} = (4, 3, 3)$, 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 $\boldsymbol{x}^{(i)} = (x_1^{(i)}, \dots, x_m^{(i)})'$, m is the category number. Denote $\mathbf{H}^{(i)}$ to be the covariate matrix of G_i, then $\mathbf{H}^{(i)} = (\mathbf{1}, \boldsymbol{h}_1^{(i)}, \dots, \boldsymbol{h}_{s_i}^{(i)})'$ 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 $\boldsymbol{x}^{(i)}$ for group G_i be $p(\boldsymbol{x}^{(i)})$. The total log-likelihood for all groups can be computed as

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

Where $p(\boldsymbol{x}^{(i)})$ 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

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

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

In the rest of this part, we apply the model to the dataset "ai4i" (Dua and Graff 2017). 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 $G_1, \ldots, G_{100}$, 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 $\boldsymbol{\beta}$ will be 3×2 if we set category 3 as baseline.

The estimates of the parameters are

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

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

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

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

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. Using the soft classifiers, 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

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

(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 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 (Deitsch et al. 2019).

The work of Deitsch et al. (2019), Buerhop-Lutz et al. (2018), and Deitsch et al. (2021) 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 3×3 and stride 1×1. 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.

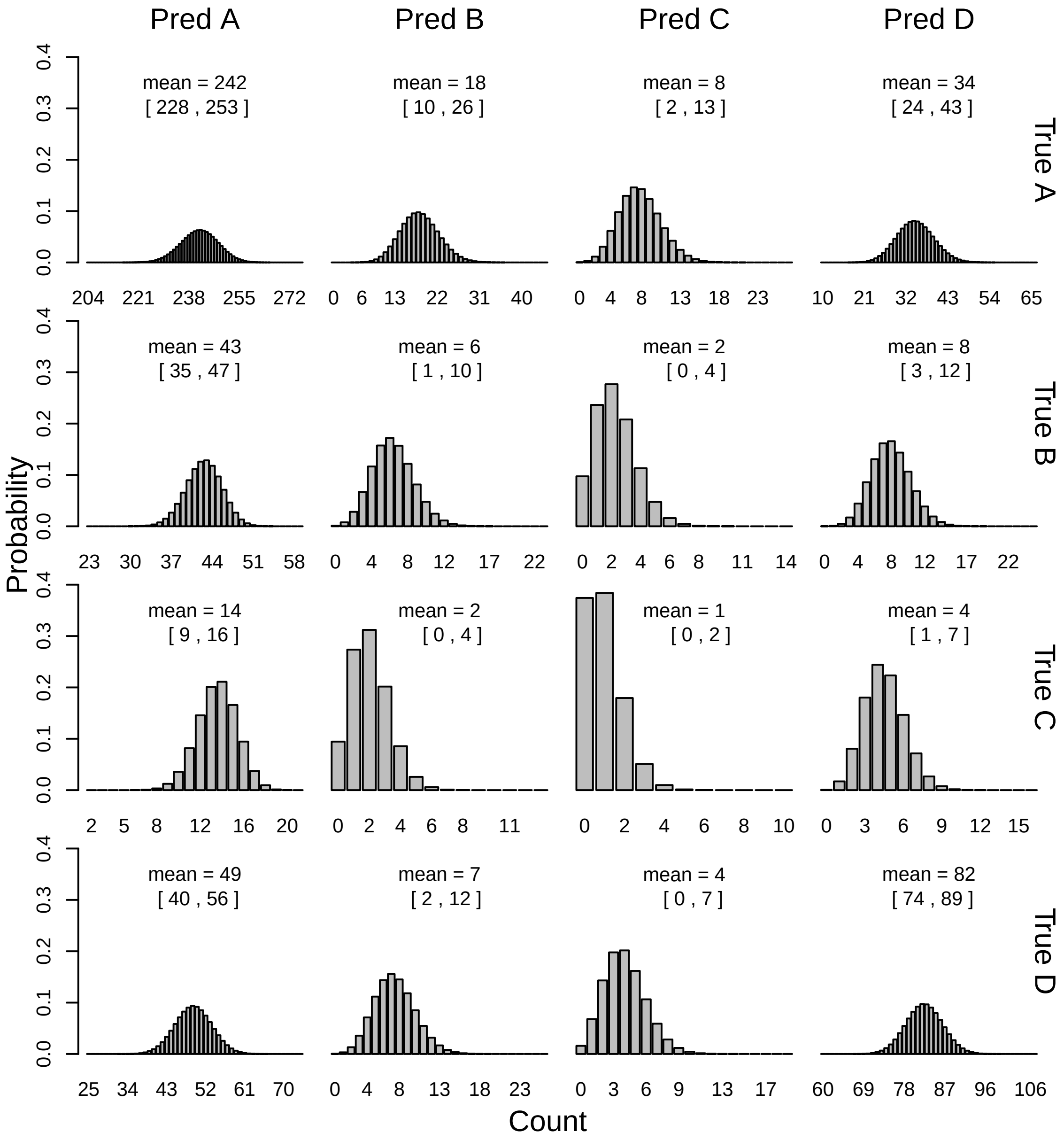

Figure 6: The barplot shows the confusion matrix prediction. For each panel cell, the plot shows the corresponding prediction counts fall in this cell as well as their probability. We also present the mean and 95% naive prediction interval.

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

We develop an 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.

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 $\boldsymbol{x}$, `dpmd`() will automatically calculate the whole pmf and the output will be a multi-dimensional array. If user inputs $\boldsymbol{x}$, the output of `dpmd`() will be partial pmf chosen by $\boldsymbol{x}$. 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 $\Pr(\boldsymbol{X} \leq \boldsymbol{x})$ and the function `rpmd` generates random samples from PMD. Examples of using `dpmd` is following,

First, `pp` is an input matrix that specifies a PMD. It looks as

```
pp=matrix(c(0.1, 0.1, 0.1, 0.7,
            0.1, 0.3, 0.3, 0.3,
            0.5, 0.2, 0.1, 0.2),
            byrow=T, ncol=4, nrow=3)
```

and $\boldsymbol{x} = (0, 0, 1, 2)'$. Then the codes of using `dpmd` to calculate pmf, is given as following

```
dpmd(pmat = pp)
dpmd(pmat = pp, xmat = x)
dpmd(pmat = pp, xmat = x, method = "NA" )
dpmd(pmat = pp, method = "SIM", B = 1e3)
dpmd(pmat = pp, xmat = x, method = "SIM", B = 1e3)
```

The first line computes the whole pmf, the second one computes partial pmf of given $\boldsymbol{x}$, the rest of the codes just do the similar job using NA and SIM. We shall demonstrate the one to one map between the output and the pmf of an $n \times m$ PMD. First we have a location indicator vector $v = (v_1, \ldots, v_{m-1})'$ and an $\boldsymbol{x} = (x_1, \ldots, x_m)'$, let $v_j - 1 = x_j, j = 1, \ldots, m-1$ then the array element with location v has the value of $\Pr(X = \boldsymbol{x})$. We further illustrate this map via the $\mathbf{P}$ given above.

The $\mathbf{P}$ is 3×4 so the output of `dpmd` is a $4 \times 4 \times 4$ dimensional array listed in figure (a), (b), (c), (d). Since the input `pp` is dimension 3×4, the output is an $4 \times 4 \times 4$ array. First consider the element of row 3 column 2 in (a). Then the first two elements of v is 3 and 2. the third element depends on the comma number on the top of each figure. On the top of (a) there are two commas followed by 1, thus the third element of v is 1. Now we have constructed $v = (3, 2, 1)$, then we minus 1 on each element we get $(2, 1, 0) = \boldsymbol{x}'$ which is the corresponding probability mass point. So far we can get the element that has location indicated by $v = (3, 2, 1)$ is 0.022 and we have $\Pr(X = (2, 1, 0)') = 0.022$. In the other direction, if we want to know the probability of $X = \boldsymbol{x} = (0, 3, 0)'$ we need to construct its corresponding v, which can be computed by plus 1 to each element in $\boldsymbol{x}$ so that we get $v = (1, 4, 1)$. The v we just constructed indicates a location of (1,4,1) of our array. That location is the first row, 4th column in the , , 1 figure which has value 0.006.

```
res=dpmd(pmat=pp)
res
, , 1                              , , 2
      [,1]  [,2]  [,3]  [,4]             [,1]  [,2]  [,3] [,4]
[1,] 0.042 0.090 0.054 0.006       [1,] 0.069 0.084 0.015    0
[2,] 0.125 0.148 0.023 0.000       [2,] 0.138 0.042 0.000    0
[3,] 0.052 0.022 0.000 0.000       [3,] 0.021 0.000 0.000    0
[4,] 0.005 0.000 0.000 0.000       [4,] 0.000 0.000 0.000    0
, , 3                              , , 4
      [,1]  [,2] [,3] [,4]               [,1] [,2] [,3] [,4]
[1,] 0.030 0.012    0    0         [1,] 0.003    0    0    0
[2,] 0.019 0.000    0    0         [2,] 0.000    0    0    0
[3,] 0.000 0.000    0    0         [3,] 0.000    0    0    0
[4,] 0.000 0.000    0    0         [4,] 0.000    0    0    0
```

7 Conclusions and Areas for Future Research

We develop three methods that can be useful for computing the pmf of the PMD, which is challenging to compute but useful in many application scenarios. We include the three methods we designed in an R package. 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 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 due to the computational limit of machines, this area remains to be disclosed.

References

Akter, L. A., I. Moon, and G.-R. Kwon (2019). Double random phase encoding with a poisson-multinomial distribution for efficient colorful image authentication. *Multimedia Tools and Applications 78*(11), 14613–14632.

Buerhop-Lutz, C., S. Deitsch, A. Maier, F. Gallwitz, S. Berger, B. Doll, J. Hauch, C. Camus, and C. J. Brabec (2018). A benchmark for visual identification of defective solar cells in electroluminescence imagery. In *European PV Solar Energy Conference and Exhibition (EU PVSEC)*.

Cheng, Y., I. Diakonikolas, and A. Stewart (2017). Playing anonymous games using simple strategies. In *SODA*.

Daskalakis, C., G. Kamath, and C. Tzamos (2015). On the structure, covering, and learning of poisson multinomial distributions. *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, 1203–1217.

Deitsch, S., C. Buerhop-Lutz, E. Sovetkin, A. Steland, A. Maier, F. Gallwitz, and C. Riess (2021). Segmentation of photovoltaic module cells in uncalibrated electroluminescence images. *32*(4).

Deitsch, S., V. Christlein, S. Berger, C. Buerhop-Lutz, A. Maier, F. Gallwitz, and C. Riess

(2019, June). Automatic classification of defective photovoltaic module cells in electroluminescence images. *Solar Energy 185*, 455–468.

Diakonikolas, I., D. M. Kane, and A. Stewart (2016). The fourier transform of poisson multinomial distributions and its algorithmic applications. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 1060–1073.

Dua, D. and C. Graff (2017). UCI machine learning repository.

Frigo, M. and S. Johnson (2005). The design and implementation of fftw3. *Proceedings of the IEEE 93*(2), 216–231.

Hong, Y. (2013a). On computing the distribution function for the poisson binomial distribution. *Computational Statistics & Data Analysis 59*, 41–51.

Hong, Y. (2013b). On computing the distribution function for the Poisson bionomial distribution. *Computational Statistics and Data Analysis 59*, 41–51.

Kamath, G. G. C. (2014). *On learning and covering structured distributions*. Ph. D. thesis, Massachusetts Institute of Technology.

Raič, M. (2019). A multivariate berry–esseen theorem with explicit constants. *Bernoulli 25*(4A), 2824–2853.

Zhang, M., Y. Hong, and N. Balakrishnan (2018). The generalized poisson-binomial distribution and the computation of its distribution function. *Journal of Statistical Computation and Simulation 88*(8), 1515–1527.