The Generalized Poisson-Binomial Distribution and the Computation of Its Distribution Function

February 2, 2018

Abstract

The Poisson-binomial distribution is useful in many applied problems in engineering, actuarial science, and data mining. The Poisson-binomial distribution models the distribution of the sum of independent but non-identically distributed random indicators whose success probabilities vary. In this paper, we extend the Poisson-binomial distribution to a generalized Poisson-binomial (GPB) distribution. The GPB distribution corresponds to the case where the random indicators are replaced by two-point random variables, which can take two arbitrary values instead of 0 and 1 as in the case of random indicators. The GPB distribution has found applications in many areas such as voting theory, actuarial science, warranty prediction, and probability theory. As the GPB distribution has not been studied in detail so far, we introduce this distribution first and then derive its theoretical properties. We develop an efficient algorithm for the computation of its distribution function, using the fast Fourier transform. We test the accuracy of the developed algorithm by comparing it with enumeration-based exact method and the results from the binomial distribution. We also study the computational time of the algorithm under various parameter settings. Finally, we discuss the factors affecting the computational efficiency of the proposed algorithm, and illustrate the use of the software package.

Key Words: Actuarial Science; Discrete Distribution; Fast Fourier Transform; Rademacher Distribution; Voting Theory; Warranty Cost Prediction.

1 Introduction

Consider a sequence of n independent and identically distributed (iid) random indicators, $I_1, \ldots, I_n \sim$ Bernoulli(p), where p is the success probability. The distribution of the sum, $\sum_{k=1}^{n} I_k$, is well known as the binomial distribution. The cumulative distribution function (cdf) and probability mass function (pmf) of the binomial distribution can be easily computed and closed-form expressions are available. When I_k's are non-identically distributed (i.e., $I_k \sim$ Bernoulli(p_k), $k = 1, \ldots, n$, with different probabilities p_k), the sum, $\sum_{k=1}^{n} I_k$, is distributed as the Poisson-binomial distribution. The Poisson-binomial distribution can be considered as a generalization of the binomial distribution (e.g., Hong 2013).

Even though the Poisson-binomial distribution is applicable in a wide range of areas including engineering, actuarial science, and data mining, a more general distribution is often of interest in practice. To motivate the need for this new distribution, we provide the following examples:

• In voting theory (e.g., Alger 2006), each voter may vote in favor of a candidate with different probabilities. In some settings, some voters may be in a more powerful position than others, for example, a voter may have two votes. The interest here is in the total number of votes that is in favor of the candidate. The general question that is of interest is about the distribution of the total number of votes in favor of the candidate;

• In warranty cost prediction (e.g., Hong and Meeker 2013), suppose there is a batch of units in the field and their failure probabilities within one year are different from unit to unit. If one unit fails, the company needs to cover a certain amount of expenses and these expenses may be different from unit to unit. The total expense is of interest in this case, and we are naturally interested here in the distribution of the total expenses;

• In actuarial science (e.g., Pitacco 2007), the amount of insurance payout is related to the payout to each customer and the payout probabilities. The distribution of the total amount of payout is generally of interest;

• In probability theory, the Rademacher distribution is defined as the distribution of a random variable which has 50% chance of being either $+1$ or -1 (Montgomery-Smith 1990). The Rademacher series is defined as a weighted sum of a series of independent Rademacher random variables (Cheng and Duncan 2014). The distribution of the Rademacher series is of interest (e.g., Dilworth and Montgomery-Smith 1993).

To solve the problems mentioned in the above examples, we consider here a Generalized Poisson-Binomial (GPB) distribution. Define n two-point random variables as follows:

$$E_k = \begin{cases} a_k, & \text{with probablity } 1 - p_k; \\ b_k, & \text{with probability } p_k, \end{cases} \tag{1}$$

where $a_k < b_k$ are constants and $k = 1, \cdots, n$. E_k is called a two-point random variable as its support is on two points $\{a_k, b_k\}$. Note that E_k can take two arbitrary values instead of 0 and 1 as in the case of a random indicator. Using I_k, E_k can be written as

$$E_k = a_k(1 - I_k) + b_k I_k.$$

By properly specifying the values of a_k and b_k, the problems mentioned in the above examples can be solved by finding the distribution of the sum $X = \sum_{k=1}^{n} E_k$. In regard to the voting example, each voter may vote with different probabilities favoring the candidate and some of their votes may weigh more than others. Then, X corresponds to the weighted sum of votes that are in favor of the candidate. We now formally define the GPB distribution.

Definition 1 *Consider n two-point random variables $E_1, \ldots, E_n$ as defined in* (1). *The random variable of interest is the sum of the n two-point random variables given by*

$$X = \sum_{k=1}^{n} E_k. \tag{2}$$

The generalized Poisson-binomial (GPB) distribution is defined as the distribution of X in (2).

Clearly, when all $a_k = 0$ and all $b_k = 1$, X reduces to $X = \sum_{k=1}^{n} I_k$, which follows the Poisson-binomial distribution.

The contribution of this paper is the introduction of the new GPB distribution, and the investigation of its theoretical properties. We further develop an efficient algorithm for the computation of its distribution function, implement the algorithm in an R package, and provide an illustration for practitioners. The computation of the cdf, however, is non-trivial. There is no existing algorithm that can efficiently compute the cdf, despite the usefulness of the distribution in practice. Although a similar algorithm as in Hong (2013) is used for computing the cdf of the Poisson-Binomial distribution, our paper focuses on a broader picture by introducing this new distribution, studying its theoretical properties, and then carrying out a comparison of different algorithms for computing the distribution function.

The rest of this paper is organized as follows. Section 2 describes the theoretical properties of the GPB distribution. Section 3 discusses the development and implementation of the algorithm for the computation of the distribution function. Section 4 covers the validation of the proposed algorithm, while Section 5 provides an illustration of the algorithm. Finally, Section 6 presents some concluding remarks.

2 Theoretical Properties of the Distribution

In this section, we establish some theoretical properties of the GPB distribution. The mean and variance of the distribution is presented in the following lemma.

Lemma 1 *Let X be a random variable that follows the GPB distribution. Then, the mean and variance of X are as follows:*

$$\mu = \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} p_k(b_k - a_k) \quad \text{and} \quad \sigma^2 = \sum_{k=1}^{n} p_k(1-p_k)(b_k - a_k)^2.$$

The proof of this lemma is trivial and so we omit it.

We now consider the cdf of X, which is defined as

$$F(x) = \Pr(X \leq x).$$

Let $\boldsymbol{I} = (I_1, \ldots, I_n)$. Note that I_k takes values in $\{0, 1\}$. Thus, the support of the random vector $\boldsymbol{I}$ is

$$\mathcal{B} = \underbrace{\{0,1\} \times \cdots \times \{0,1\} \times \cdots \times \{0,1\}}_{n \text{ times}}.$$

The number of elements in $\mathcal{B}$ is clearly 2^n. Let $\boldsymbol{r} = (r_1, \cdots, r_k, \cdots, r_n)$ be an element in $\mathcal{B}$. The support of X is

$$\mathcal{X} = \left\{ x : \text{ there exists at least one } \boldsymbol{r} \in \mathcal{B} \text{ such that } x = \sum_{k=1}^{n} a_k(1 - r_k) + b_k r_k \right\}.$$

To illustrate the set $\mathcal{X}$, we provide the following example.

Example 1 *Let $I_1 \sim$ Bernoulli(p_1), $I_2 \sim$ Bernoulli(p_2) and $I_3 \sim$ Bernoulli(p_3) be three independent random indicators with success probabilities $p_1 = 0.1$, $p_2 = 0.2$ and $p_3 = 0.3$, respectively. The values of a_k's and b_k's are specified as $a_1 = 1, a_2 = 2, a_3 = 3$ and $b_1 = 2, b_2 = 3, b_3 = 4$. The support of random vector (I_1, I_2, I_3) is*

$$\mathcal{B} = \{(0,0,0), (1,0,0), (1,0,1), (1,1,0), (1,1,1), (0,1,0), (0,1,1), (0,0,1)\}.$$

The support of $X = \sum_{k=1}^{3} a_k(1 - I_k) + b_k I_k$ is $\mathcal{X} = \{6, 7, 8, 9\}$. For example, $\boldsymbol{r} = (0,0,0)$ yields $x = 6$, and similarly $\boldsymbol{r} = (0,0,1)$ yields $x = 7$.

We have the following theorem for the calculation of the pmf and the cdf of the GPB distribution.

Theorem 1 *Let $\boldsymbol{r} \in \mathcal{B}$ and $\mathcal{S}_x$ be a set of $\boldsymbol{r}$ such that $x = \sum_{k=1}^{n} a_k(1 - r_k) + b_k r_k$. Let $a = \sum_{k=1}^{n} a_k$ and $b = \sum_{k=1}^{n} b_k$. The pmf ξ_x can be calculated as*

$$\xi_x = \Pr(X = x) = \sum_{\boldsymbol{r} \in \mathcal{S}_x} \prod_{k=1}^{n} p_k^{r_k}(1 - p_k)^{1-r_k}, \quad a \leq x \leq b.$$

The cdf can then be calculated as $F(x) = \sum_{u \leq x} \xi_u$.

Note that $\prod_{k=1}^{n} p_k^{r_k}(1-p_k)^{1-r_k}$ gives the probability $\Pr(\boldsymbol{I} = \boldsymbol{r})$. Thus, the result in **Theorem 1** is essentially based on enumeration. To illustrate **Theorem 1**, we provide the following example.

Example 2 *Under the setting of **Example 1**, the $\mathcal{S}_x$ set corresponding to each value x is obtained as*

$$\begin{aligned} \mathcal{S}_6 =& \{(0,0,0)\}, \quad \mathcal{S}_7 = \{(1,0,0),(0,1,0),(0,0,1)\} \\ \mathcal{S}_8 =& \{(1,1,0),(1,0,1),(0,1,1)\}, \quad \mathcal{S}_9 = \{(1,1,1)\}. \end{aligned}$$

The pmf is then calculated as

$$\begin{aligned} \xi_6 &= \Pr(X=6) = 0.9 \times 0.8 \times 0.7 = 0.504, \\ \xi_7 &= \Pr(X=7) = 0.1 \times 0.8 \times 0.7 + 0.9 \times 0.2 \times 0.7 + 0.9 \times 0.8 \times 0.3 = 0.398, \\ \xi_8 &= \Pr(X=8) = 0.1 \times 0.2 \times 0.7 + 0.1 \times 0.8 \times 0.3 + 0.9 \times 0.2 \times 0.3 = 0.092, \\ \xi_9 &= \Pr(X=9) = 0.1 \times 0.2 \times 0.3 = 0.006. \end{aligned}$$

Note that the enumeration based method can be computationally infeasible for large n (e.g., when $n > 30$). When $n = 30$, 2^{30} is around one billion. Thus, a computationally efficient method is very much needed, which will be presented in the next section.

3 Computation of Distribution Function

For the development of the algorithm, we restrict a_k and b_k to be integers. We will discuss non-integer cases later in Section 5. Let

$$\xi_j = \Pr(X = j + a), j = 0, \cdots, m,$$

be the pmf of X, where $a = \sum_{k=1}^{n} a_k$, $b = \sum_{k=1}^{n} b_k$, and $m = b - a$. The objective is to compute the pmf $\xi_j, j = 0, \cdots, m$. A computationally efficient algorithm is based on the characteristic function (CF) of X. The following theorem shows two ways of obtaining the CF of X.

Theorem 2 *Let X be a random variable that follows the GPB distribution. Let $a = \sum_{k=1}^{n} a_k$ and $b = \sum_{k=1}^{n} b_k$. Let $\xi_j = \Pr(X = j + a)$ be the pmf of X, where $j = 0, \cdots, m$ and $m = b - a$. By the definition of CF, it can be obtained from the pmf as*

$$\varphi(t) = \mathbf{E}[\exp(\boldsymbol{i}tX)] = \sum_{j=0}^{m} \xi_j \exp[\boldsymbol{i}t(j + a)], \tag{3}$$

where $\boldsymbol{i} = \sqrt{-1}$. Alternatively, the CF can be calculated by using the fact that X is defined as an independent sum (i.e., $X = \sum_{k=1}^{n} a_k(1 - I_k) + b_k I_k$). In particular,

$$\varphi(t) = \mathbf{E}\left\{\exp\left[\boldsymbol{i}t\sum_{k=1}^{n} a_k(1 - I_k) + b_k I_k\right]\right\} = \prod_{k=1}^{n}\left[(1 - p_k)\exp(\boldsymbol{i}ta_k) + p_k\exp(\boldsymbol{i}tb_k)\right]. \quad (4)$$

The formula in (3) involves ξ_j which is to be computed, while the formula in (4) involves the product of n complex numbers which can be directly computed from a_k's, b_k's, and p_k's. Thus, the formula in (4) possesses some computational advantages. To develop the computational algorithm, we link (3) and (4), and obtain

$$\sum_{j=0}^{m} \xi_j \exp(\boldsymbol{i}tj) = \exp[-\boldsymbol{i}ta]\prod_{k=1}^{n}\left[(1 - p_k)\exp(\boldsymbol{i}ta_k) + p_k\exp(\boldsymbol{i}tb_k)\right]. \quad (5)$$

Let $x_l = \exp[-\boldsymbol{i}\omega la]\prod_{k=1}^{n}\left[(1 - p_k)\exp(\boldsymbol{i}\omega la_k) + p_k\exp(\boldsymbol{i}\omega lb_k)\right]$, where $\omega = 2\pi/(m+1)$. Note that x_l is a complex number that can be represented as $x_l = u_l + v_l\boldsymbol{i}$. We substitute $t = \omega l$ into (5) and obtain a set of $m + 1$ equations

$$\frac{1}{m+1}\sum_{j=0}^{m} \xi_j \exp(\boldsymbol{i}\omega lj) = \frac{1}{m+1}x_l, \quad l = 0, 1, \cdots, m. \quad (6)$$

According to Bracewell (2000), the left hand side of (6) is the inverse discrete Fourier transform of the sequence $(\xi_0, \xi_1, \cdots, \xi_m)$, which can be recovered by applying the discrete Fourier transform (DFT) to both sides of (6). Thus, we can compute the pmf ξ_j as

$$\xi_j = \frac{1}{m+1}\sum_{l=0}^{m} \exp(-\boldsymbol{i}\omega lk)x_l, \quad j = 0, 1, \cdots, m. \quad (7)$$

Because there are efficient algorithms to do the DFT, the right hand side of (7) can be computed efficiently. We call this method as the DFT-CF algorithm. Note that Hong (2013) used a similar algorithm to compute the cdf of the Poisson-binomial distribution. The algorithm in this paper deals with the GPB, which is a different distribution and involves more complicated calculations.

The technical details of computing the x_l's are shown in the appendix. We provide the following example to illustrate the calculation of x_l and ξ_j for a small n.

Example 3 *We consider four random indicators $I_1 \sim$ Bernoulli(p_1), $I_2 \sim$ Bernoulli(p_2), $I_3 \sim$ Bernoulli(p_3) and $I_4 \sim$ Bernoulli(p_4), where $p_1 = 0.1$, $p_2 = 0.2$, $p_3 = 0.3$ and $p_4 = 0.4$. We set the a_k's to be $a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 1$, and the b_k's to be $b_1 = 1, b_2 = 2, b_3 = 3, b_4 = 4$. Under this setting, the support of X is $\mathcal{X} = \{4, 5, 6, 7, 8, 9, 10\}$. Also, we have $a = 4$, $b = 10$ and $m = 6$. The values of x_l and ξ_j are listed in Table 1.*

Formally, we describe the DFT-CF algorithm as follows, which is used to compute the pmf ξ_j, for $j = 0, 1, \cdots, m$.

Table 1: A numerical example illustrating the DFT-CF algorithm.

X	l	$x_l = u_l + v_l\boldsymbol{i}$	j	ξ_j
4	0	$1.0000 + 0.0000\boldsymbol{i}$	0	0.336
5	1	$0.0652 + 0.1822\boldsymbol{i}$	1	0.084
6	2	$0.2928 - 0.1218\boldsymbol{i}$	2	0.144
7	3	$0.3180 + 0.1874\boldsymbol{i}$	3	0.260
8	4	$0.3180 - 0.1874\boldsymbol{i}$	4	0.056
9	5	$0.2928 + 0.1218\boldsymbol{i}$	5	0.096
10	6	$0.0652 - 0.1822\boldsymbol{i}$	6	0.024

The DFT-CF Algorithm:

1. We first assign $x_0 = 1$. Then, we compute the real and imaginary parts of x_l by using the formulae in (8) in the appendix, $l = 1, \ldots [m/2]$, and $[\cdot]$ is the ceiling function;
2. We compute the real and imaginary parts of x_l by using the formula $u_l = u_{m+1-l}$ and $v_l = -v_{m+1-l}$, $l = [m/2] + 1, \ldots, m$;
3. We then apply the fast Fourier transform (FFT) algorithm to the sequence $(x_0/(m+1), x_1/(m+1), \ldots, x_n/(m+1))$ to obtain $(\xi_0, \xi_1, \ldots, \xi_m)$.

The DFT-CF algorithm has been implemented in C and it can be called from R. We also wrap the major functions into an R package `GPB` (Hong and Zhang 2016). The use of the R package will be illustrated in Section 5.

4 Algorithm Validation

This section focuses on the validation of the developed algorithm.

4.1 Accuracy Comparison with an Exact Method

We develop an enumeration-based algorithm to compute the exact cdf of the GPB distribution based on the theoretical formula in Section 2. Then, we use the maximum absolute error (MAE) and the total absolute error (TAE) as accuracy metrics by comparing the cdf calculated with the DFT-CF algorithm with that computed from the enumeration-based method for different values of n, a_k, b_k and p_k. The maximum absolute error (MAE) is defined as

$$\text{MAE} = \max_x |F(x) - F_{\text{enum}}(x)|,$$

Table 2: Accuracy of the DFT-CF algorithm compared with the enumeration method.

n	a	b	$\min(p_k)$	$\max(p_k)$	MAE	TAE
10	10	20	0.01	0.50	6.7×10^{-16}	4.4×10^{-14}
10	10	20	0.50	0.99	7.3×10^{-16}	2.8×10^{-15}
10	10	20	0.01	0.99	9.4×10^{-16}	3.8×10^{-15}
10	10	50	0.01	0.50	6.7×10^{-16}	4.4×10^{-14}
10	10	50	0.50	0.99	7.3×10^{-16}	2.8×10^{-15}
10	10	50	0.01	0.99	9.4×10^{-16}	3.8×10^{-15}
10	50	100	0.01	0.50	6.7×10^{-16}	4.4×10^{-14}
10	50	100	0.50	0.99	7.3×10^{-16}	4.6×10^{-15}
10	50	100	0.01	0.99	9.4×10^{-16}	4.5×10^{-15}
20	20	40	0.01	0.50	4.4×10^{-16}	4.1×10^{-15}
20	20	40	0.50	0.99	6.7×10^{-16}	4.1×10^{-14}
20	20	40	0.01	0.99	1.3×10^{-15}	1.1×10^{-14}
20	40	100	0.01	0.50	4.4×10^{-16}	4.1×10^{-15}
20	40	100	0.50	0.99	6.7×10^{-16}	4.1×10^{-14}
20	40	100	0.01	0.99	1.3×10^{-15}	1.1×10^{-14}
20	100	200	0.01	0.50	4.4×10^{-16}	4.1×10^{-15}
20	100	200	0.50	0.99	6.7×10^{-16}	4.1×10^{-14}
20	100	200	0.01	0.99	1.3×10^{-15}	1.1×10^{-14}

while the total absolute error (TAE) is defined as

$$\text{TAE} = \sum_{x=a}^{b} |F(x) - F_{\text{enum}}(x)|,$$

where $F(x)$ is the cdf computed by using the DFT-CF algorithm and $F_{\text{enum}}(x)$ is the cdf computed by using the enumeration formula in Section 2.

The accuracy test results are shown in Table 2 for different parameter settings for n, a_k, b_k and p. All computations were done on Linux 64-bit server with Intel Xeon CPU (E5-2680, 2.50GHz) and 263 GB RAM. Due to the complex enumeration calculation, the exact method can only handle at most 30 random indicators (i.e., $n = 30$) under the capacity of the computer server. Table 4 shows the accuracy of the cdf calculated with the DFT-CF algorithm for various values of n, a_k, b_k and p. The MAE are generally less than 5×10^{-15} and the TAE are less than 5×10^{-14} for the DFT-CF algorithm, when n is less than 20. Overall, the results show that the DFT-CF algorithm can accurately compute the cdf of the GPB distribution.

4.2 Accuracy Comparison with the Binomial Distribution

To test the accuracy of the DFT-CF algorithm for large values of m and n, we compare the cdf computed by the DFT-CF algorithm with that of the binomial distribution. As mentioned earlier, the binomial distribution is a special case of the GPB distribution when all p_k's are the same, and $a_k = 0, b_k = 1$ for all n random indicators. Thus, in this comparison setting, we let $p_k = p$ to be the same, and $a_k = 0, b_k = 1$; that is,

$$X = \sum_{k=1}^{n} a_k(1 - I_k) + b_k I_k = \sum_{k=1}^{n} I_k.$$

The exact pmf of X can be calculated from the binomial distribution as

$$\Pr(X = x) = \binom{n}{x} p^x (1 - p)^{n-x}.$$

Here again, the MAE and TAE are used as accuracy metrics, which are given by

$$\text{MAE} = \max_x |F(k) - F_{\text{bin}}(k)|, \quad \text{and} \quad \text{TAE} = \sum_{x=0}^{n} |F(x) - F_{\text{bin}}(x)|,$$

where $F(x)$ is the cdf computed by the DFT-CF algorithm and $F_{\text{bin}}(x)$ is the cdf computed using the binomial distribution function implemented in R (2016). The accuracy test results are shown in Table 3 with different parameter settings for n, a_k, b_k and p. Basically, the TAE and MAE accumulates as n, p and a_k, b_k increases. When n is less than 10,000, the MAE is within 1×10^{-12} and TAE is under 1×10^{-8}. The results in Table 3 show that the DFT-CF algorithm can accurately compute the cdf for large n. With no efficient algorithm available for computing the GPB distribution, the proposed method provides an important and efficient way to model the GPB distribution in real-life applications.

4.3 Computational Efficiency

The computational time of the DFT-CF algorithm is mostly determined by n and m. Note that $m = b - a = \sum_{k=1}^{n} b_k - \sum_{k=1}^{n} a_k$. We first consider the computational time when n is large. We choose 10 different success probability p's from [0.01, 0.99], and for different values of n and m, we set $p_k = p, a_k = 0, b_k = m/n$ for all n indicators. The time for calculating the entire cdf using the DFT-CF algorithm is averaged across 10 different values for p. The averaged computing time is shown in Table 4. The unit of the computational time is in seconds.

Figure 1 visualizes the results in Table 4. Both the x-axis and y-axis are on log-scales. Each line indicates the average computational time for a specific n, where n = 10, 100, 1,000 and 10,000. Figure 1 shows that as n increases, the computational time increases exponentially. The computational time is negligible (less than 10 milliseconds) when $n \leq 100$. When n is fixed at 10, 100, 1,000 and 10,000, the general average computational time increases as m increases. However, there are minor drops in computational time even when m increases to large numbers. These drops may be caused by the computational complexity of the FFT algorithm, which

Table 3: Accuracy of the DFT-CF algorithm compared with the binomial distribution, with $a = 0$, and $b = n$.

n	p	MAE	TAE	n	p	MAE	TAE
10	0.01	8.9×10^{-16}	3.9×10^{-15}	2,000	0.01	2.9×10^{-14}	2.3×10^{-11}
10	0.50	4.4×10^{-16}	1.6×10^{-15}	2,000	0.50	1.4×10^{-13}	1.1×10^{-10}
10	0.90	6.7×10^{-16}	3.2×10^{-15}	2,000	0.90	4.2×10^{-13}	3.4×10^{-10}
20	0.01	4.4×10^{-16}	4.7×10^{-15}	5,000	0.01	1.3×10^{-13}	3.2×10^{-10}
20	0.50	7.1×10^{-16}	6.3×10^{-15}	5,000	0.50	4.3×10^{-13}	6.4×10^{-10}
20	0.90	1.9×10^{-15}	1.8×10^{-14}	5,000	0.90	7.1×10^{-13}	1.1×10^{-9}
50	0.01	2.0×10^{-15}	5.3×10^{-14}	10,000	0.01	4.2×10^{-13}	1.8×10^{-9}
50	0.50	2.9×10^{-15}	5.2×10^{-14}	10,000	0.50	1.1×10^{-12}	3.2×10^{-9}
50	0.90	3.5×10^{-15}	5.3×10^{-14}	10,000	0.90	1.6×10^{-12}	4.6×10^{-9}
100	0.01	1.4×10^{-14}	5.7×10^{-13}	20,000	0.01	6.9×10^{-13}	3.6×10^{-9}
100	0.50	1.9×10^{-15}	3.7×10^{-14}	20,000	0.50	2.7×10^{-12}	1.7×10^{-8}
100	0.90	7.4×10^{-15}	2.7×10^{-13}	20,000	0.90	5.4×10^{-12}	3.9×10^{-8}
200	0.01	7.8×10^{-15}	8.8×10^{-13}	50,000	0.01	3.1×10^{-12}	6.7×10^{-8}
200	0.50	5.3×10^{-15}	4.9×10^{-13}	50,000	0.50	9.8×10^{-12}	1.3×10^{-7}
200	0.90	3.0×10^{-14}	1.9×10^{-12}	50,000	0.90	1.6×10^{-11}	1.8×10^{-7}
500	0.01	2.4×10^{-14}	5.0×10^{-12}	100,000	0.01	5.7×10^{-12}	2.0×10^{-7}
500	0.50	2.4×10^{-14}	5.5×10^{-12}	100,000	0.50	1.6×10^{-11}	3.4×10^{-7}
500	0.90	6.7×10^{-14}	1.7×10^{-11}	100,000	0.90	4.3×10^{-11}	1.2×10^{-6}
1,000	0.01	5.2×10^{-14}	2.1×10^{-11}				
1,000	0.50	5.8×10^{-14}	2.0×10^{-11}				
1,000	0.90	1.8×10^{-13}	7.5×10^{-11}				

Table 4: Average computational time in seconds for the DFT-CF algorithm over p for different choices of n and m.

m \ n	10	100	1,000	10,000
10	0.000			
20	0.000			
50	0.000			
100	0.000	0.000		
200	0.000	0.001		
500	0.000	0.001		
1,000	0.001	0.003	0.023	
2,000	0.001	0.005	0.045	
5,000	0.006	0.017	0.118	
10,000	0.004	0.024	0.225	2.228
20,000	0.009	0.049	0.452	4.456
50,000	0.084	0.183	1.191	11.207
100,000	0.652	0.809	2.837	22.786
200,000	0.160	0.552	4.579	44.933
500,000	110.423	103.886	112.395	250.482
1,000,000	7.406	8.973	28.826	232.975
2,000,000	3765.094	1751.390	3753.482	2657.690
5,000,000	211.050	221.361	339.458	1358.968
10,000,000	26850.920	14401.410	26927.20	18215.330

depends on the factorization of $(m + 1)$. The DFT-CF algorithm can complete the required computation within five minutes when $n \leq 10{,}000$ and $m \leq 1{,}000{,}000$. As n exceeds 10,000 and m exceeds 1,000,000, the DFT-CF algorithm requires more than five minutes. Overall, the DFT-CF algorithm shows reasonable computational efficiency.

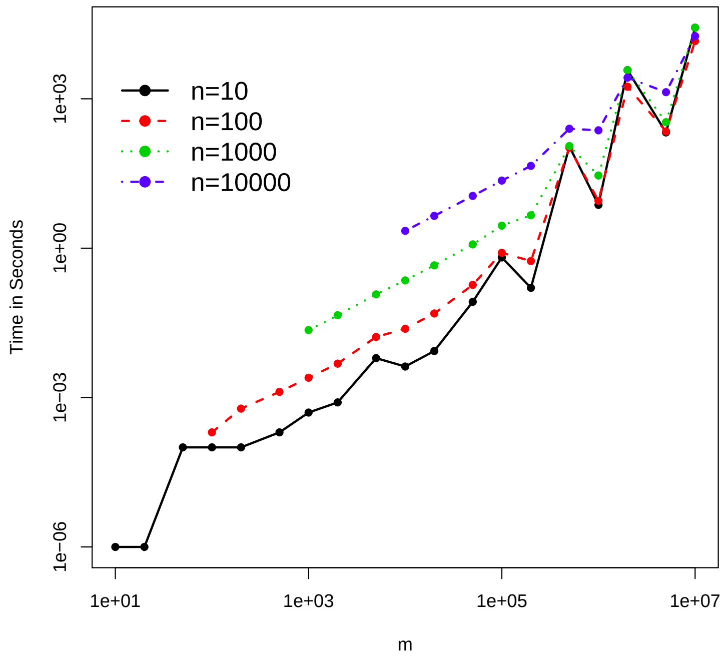

Figure 1: Average computational time of the DFT-CF algorithm for different choices of n and m. The x-axis and y-axis are both on the log-scale.

5 Illustrations

5.1 The Software Package

The DFT-CF algorithm has been implemented in the R package `GPB`, Hong and Zhang (2016), which can be downloaded from the Comprehensive R Archive Network (http://cran.r-project.org/). The R functions for computing the cdf, pmf, quantile function and random number generation are all available in the R package. For example, the function `pgpb()` takes p_k's, a_k's and b_k's as input parameters and computes the cdf of the distribution. See the following R code for reference:

```
library(GPB)
pgpb(kk=6:9,pp=c(0.1,0.2,0.3),aval=c(1,2,3),bval=c(2,3,4),
wts=c(1,1,1))
```

Here, `kk` is x (i.e., the values for which the cdf needs to be evaluated), `pp` is the vector of p_k's, `aval` is the vector of a_k's, `bval` is the vector of b_k's, and `wts` is the vector of weights for p_k's.

5.2 Computational Tips

There are a few tips that need to be given in the use of the algorithm. One can use the weights option to speed up the computing when the p_k's are all the same. For example, if there are 1,000 random indicators with the same p, instead of replicating the same random indicator 1,000 times, one can specify `wts`=1,000. The use of `wts` argument will speed up the computing process due to the way the implementation of the algorithm has been made.

The proposed algorithm can be slow in cases when n and m are extremely large. In some cases, the problem can be eased by taking out the greatest common divisor of a_k's and b_k's. For example, when $k = 1, 2, \ldots, 10$, $a_k = 10, 20, \ldots, 100$, and $b_k = 100, 200, \ldots, 1{,}000$, the cdf is equivalent to the cdf when $a_k = 1, 2, \ldots, 10$ and $b_k = 10, 20, \ldots, 100$, with a multiplier of 10 for the support values. Thus, by taking out the greatest common divisor, we can reduce the computational time especially when m is large and the common divisors between a_k's and b_k's are large. See the following R code for reference:

```
pgpb(kk=seq(10,100,by=10), pp=c(.1, .2, .3), aval=c(10,20,30),
bval=c(20,30,40), wts=c(1,1,1))
pgpb(kk=1:10, pp=c(.1, .2, .3), aval=c(1,2,3), bval=c(2,3,4),
wts=c(1,1,1))
```

Although the algorithm is developed based on cases when both a_k and b_k are integers, it can be applied to non-integer cases by multiplying powers of 10 to convert decimal digits to integers. For example, if $a_k = \{0.5, 1.5, \ldots, 9.5\}$, $b_k = \{1, 2, \ldots, 10\}$, and cdf needs to be computed at $x = 50.5$, then we can multiply the set of a_k, b_k and x by 10. The cdf value is the same as being computed at $x = 505$ with $a_k = \{5, 15, \ldots, 95\}$ and $b_k = \{10, 20, \ldots, 100\}$. Note that the multiplication by powers of 10 increases m and correspondingly increases the computational time. However, this process can extend the proposed algorithm to the cases of non-integer numbers. See the following R code for reference:

```
aval=seq(0.5,9.5,by=1)*10
bval=seq(1,10,by=1)*10
pgpb(kk=50.5*10,pp=seq(0.1,0.5,length.out=10),aval=aval,
bval=bval,wts=rep(1,10))
```

6 Concluding Remarks

In this paper, we introduce the GPB distribution and derive some of its theoretical properties. We develop an efficient DFT-CF algorithm to compute the cdf of the GPB distribution. We demonstrate that the proposed algorithm is accurate in terms of error as compared to an enumeration-based exact method and the binomial distribution in a special case. We further show the computational efficiency and the limitation of the DFT-CF algorithm in numerical analysis for different settings of n and m. The DFT-CF algorithm is generally accurate (with the TAE under 1×10^{-8}) and computationally efficient (less than five minutes for computing) when n is less than 10,000. The DFT-CF algorithm can be extended to non-integer numbers as well. The proposed method has been implemented in an R package named `GPB`.

From probability theory, the pmf of the sum of two discrete random variables can be found by using convolution. From the definition in (2), X is defined as an independent sum of E_k's. Because the pmf of E_k is simple, one possible way to find the pmf of X is to sequentially apply the convolution. For example, one can find the pmf of $E_1 + E_2$ by using the convolution of the pmf of E_1 and E_2. Then, one can find the pmf of $E_1 + E_2 + E_3$ by using the convolution of the pmf of $E_1 + E_2$ and E_3. When n and m are large, this process can also be very time consuming. However, it will be an interesting topic to investigate in the future research how such a method compares the method developed here in terms of accuracy as well as computational time.

In Section 1, we mention several possible areas of applications for the GPB distribution. However, the applications of the GPB distribution are much broader. For example, it can be useful in econometrics (e.g., Duffie, Saita, and Wang 2007), data mining (e.g., Tang and Peterson 2011), bioinformatics (e.g., Niida et al. 2012), renewable energy (Bossavy, Girard, and Kariniotakis 2013), and survey sampling (e.g., Chen and Liu 1997), in which cases there are costs associated with random indicators with different success probabilities. The implementation of the developed algorithm in R makes it convenient for the practitioners.

A Technical Details

In this appendix, we give the technical details of the calculation of $x_l, l = 0, 1, \ldots, m$, which are essential in the DFT-CF algorithm. We use the same process as in Hong (2013). However, the formulas are different due to different distribution settings. Note that x_l is a complex number, which is represented as $x_l = u_l + \boldsymbol{i} v_l$ with real part u_l and imaginary part v_l. From (6), we have

$$x_l = \sum_{j=0}^{m} \xi_j \exp(\boldsymbol{i}\omega l j)$$

that holds for $l = 0, 1, \cdots, m$. When $l = 0$,

$$x_0 = \sum_{j=0}^{m} \xi_j \exp(\boldsymbol{i}\omega 0 j) = \sum_{j=0}^{m} \xi_j \exp(0) = \sum_{k=0}^{n} \xi_k = 1,$$

because ξ_j is pmf summing up to one. In general, we denote $|z|$ as the modulus and $\mathrm{Arg}(z)$ as the principal value of the argument of a complex number z. Let

$$\begin{aligned} z_{0l} &= \cos(-\omega l a) + \boldsymbol{i}\sin(-\omega l a), \\ z_{kl} &= [(1-p_k)\cos(\omega l a_k) + p_k \cos(\omega l b_k)] + \boldsymbol{i}[(1-p_k)\sin(\omega l a_k) + p_k \sin(\omega l b_k)]. \end{aligned}$$

For $l \neq 0$, we obtain

$$\begin{aligned} x_l &= \exp\left[\sum_{k=0}^{n} \log(z_{kl})\right] = \exp\left(\sum_{k=0}^{n} \log\left\{|z_{kl}| \exp[\boldsymbol{i}\mathrm{Arg}(z_{kl})]\right\}\right) \\ &= \exp\left[\sum_{k=0}^{n} \log\left(|z_{kl}|\right)\right] \left\{\cos\left[\sum_{k=0}^{n} \mathrm{Arg}(z_{kl})\right] + \boldsymbol{i}\sin\left[\sum_{k=0}^{n} \mathrm{Arg}(z_{kl})\right]\right\}. \end{aligned}$$

In this case, $|z_{0l}| = 1$, $\mathrm{Arg}[z_{0l}] = \mathrm{atan2}\left\{\sin(-\omega l a), \cos(-\omega l a)\right\}$, and

$$\begin{aligned} |z_{kl}| &= \left\{[(1-p_k)\cos(\omega l a_k) + p_k\cos(\omega l b_k)]^2 + [(1-p_k)\sin(\omega l a_k) + p_k \sin(\omega l b_k)]^2\right\}^{\frac{1}{2}}, \\ \mathrm{Arg}[z_{kl}] &= \mathrm{atan2}\left\{[(1-p_k)\sin(\omega l a_k) + p_k\sin(\omega l b_k)], [(1-p_k)\cos(\omega l a_k) + p_k \cos(\omega l b_k)]\right\}. \end{aligned}$$

Here, the output of the $\mathrm{atan2}(y, x)$ function is $\arctan(y/x)$ if $x > 0$; $\pi + \arctan(y/x)$ if $y \geq 0$ and $x < 0$; $-\pi + \arctan(y/x)$ if $y < 0$ and $x < 0$; $\pi/2$ if $y > 0$ and $x = 0$; $-\pi/2$ if $y < 0$ and $x = 0$; and 0 if $y = 0$ and $x = 0$. Let $z_l = \exp\left[\sum_{k=0}^{n} \log\left(|z_{kl}|\right)\right]$. We obtain formulas for u_l and v_l as

$$u_l = z_l \cos\left[\sum_{k=0}^{n} \mathrm{Arg}(z_{kl})\right] \text{ and } v_l = z_l \sin\left[\sum_{k=0}^{n} \mathrm{Arg}(z_{kl})\right], l = 1, \ldots, m. \tag{8}$$

In addition, note that all ξ_j's are real numbers and $\exp[\boldsymbol{i}\omega(m+1)j] = 1$. Thus, the conjugate of x_l, $l = 1, \ldots, m$, is

$$\overline{x_l} = u_l - \boldsymbol{i}v_l = \sum_{j=0}^{m} \xi_j \exp(-\boldsymbol{i}\omega l j) = \sum_{j=0}^{m} \xi_j \exp[\boldsymbol{i}\omega(m+1-l)j] = x_{m+1-l} = u_{m+1-l} + \boldsymbol{i}v_{m+1-l}.$$

We obtain $u_l = u_{m+1-l}$ and $v_l = -v_{m+1-l}$ for $l = 1, \ldots, m$, which means we can save half of the computing time by only computing the first half of the sequence (i.e., $l = 1, \ldots [m/2]$).

References

Alger, D. (2006). Voting by proxy. *Public Choice 126*(1), 1–26.

Bossavy, A., R. Girard, and G. Kariniotakis (2013). Forecasting ramps of wind power production with numerical weather prediction ensembles. *Wind Energy 16*, 51–63.

Bracewell, R. (2000). *The Fourier Transform & Its Applications* (Third ed.). Singapore: McGraw-Hill.

Chen, S. X. and J. S. Liu (1997). Statistical applications of the Poisson-binomial and conditional Bernoulli distributions. *Statistica Sinica 7*, 875–892.

Cheng, M. C. N. and J. F. R. Duncan (2014). *Rademacher Sums and Rademacher Series*, pp. 143–182. Berlin, Heidelberg: Springer.

Dilworth, S. J. and S. J. Montgomery-Smith (1993). The distribution of vector-valued Radmacher series. *Annals of Probability 21*, 2046–2052.

Duffie, D., L. Saita, and K. Wang (2007). Multi-period corporate default prediction with stochastic covariates. *Journal of Financial Economics 83*, 635–665.

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

Hong, Y. and W. Q. Meeker (2013). Field-failure predictions based on failure-time data with dynamic covariate information. *Technometrics 55*, 135–149.

Hong, Y. and M. Zhang (2016). *GPB: Generalized Poisson Binomial Distribution.* R package version 1.0.

Montgomery-Smith, S. J. (1990). The distribution of Rademacher sums. *Proceedings of the American Mathematical Society 109*, 517.

Niida, A., S. Imoto, T. Shimamura, and S. Miyano (2012). Statistical model-based testing to evaluate the recurrence of genomic aberrations. *Bioinformatics 28*, i115–i120.

Pitacco, E. (2007). Mortality and longevity: A risk management perspective. In *IAA Life Colloquium*, Stockholm, available at `http://www.actuaries.org/LIFE/Events/Stockholm/Pitacco.pdf`.

R Development Core Team (2016). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing.

Tang, P. and E. A. Peterson (2011). Mining probabilistic frequent closed itemsets in uncertain databases. In *Proceedings of the 49th ACM Southeast Conference (ACMSE)*, Kennesaw, GA, pp. 86–91.