A simple and fast method for computing the Poisson Binomial distribution function

William Biscarri[a,*], Dave Zhao[a], Robert Brunner[a,b,c,d]

[a] *Department of Statistics, University of Illinois at Urbana-Champaign, IL, United States*
[b] *Department of Accountancy, University of Illinois at Urbana-Champaign, IL, United States*
[c] *Department of Information Science, University of Illinois at Urbana-Champaign, IL, United States*
[d] *National Center for Supercomputing Applications, Urbana, IL, United States*

Abstract

We demonstrate that the Poisson Binomial distribution function can be efficiently and exactly calculated by simply and directly using the convolution definition of a sum of random variables. Direct calculation in this way outperforms current state of the art and widely used methods for calculating the Poisson Binomial distribution function. We further present another method of computation which improves upon the efficiency of the direct approach. These methods can also be used to efficiently calculate the distribution function of a generalized form of the Poisson Binomial.

Keywords: Poisson-Binomial, Convolution, Fourier Transform, Sum of independent Bernoullis

1. Introduction

Suppose that $\left\{X_i\right\}_{i=1}^{n}$ is a collection of independent, but not neccessarily identically distributed $Bernoulli(p_i)$ random variables, and we are interested in calculating the distribution function of their sum, $Y = \sum X_i$. Known as the Poisson Binomial distribution function, this quantity is of particular interest to a variety of fields and applications, and has seen wide use in recent years. For

*Corresponding author
Email addresses: wbiscar2@illinois.edu (William Biscarri), sdzhao@illinois.edu (Dave Zhao), bigdog@illinois.edu (Robert Brunner)

example, it has been used in genetics [22], device failure time analysis [19], reliability analysis in meteorological forecasting [10], insurance [24], item response theory [16], ecology [4], survey sampling [6], and even analysis of golf [13].

The distribution itself has received significant attention in past statistical literature (for example, [17]), where the focus has primarily been on calculating the cumulative distribution function

$$P(Y \leq y) = \sum_{s=0}^{y} \sum_{A \in F_s} \prod_{i \in A} p_i \prod_{i \in A^c} (1 - p_i) \tag{1}$$

Where F_s is defined as the set of all subsets of size s that can be chosen from the set $\{1, 2, ..., n\}$. It is clear that attempting to compute this quantity directly is intractable for even small sample sizes as each F_s contains $\binom{n}{s}$ elements, and this issue has motivated many interesting works.

A bound on the total variation distance between the Poisson Binomial density and the Poisson density was provided in Le Cam et al. [21], and is aptly named Le Cam's theorem. Further results dedicated to the use of Poisson distributions as approximations can be found in Chen [5], Deheuvels et al. [11], Steele [28], and Wang [31]. Other proposed approximation methods utilize Binomial distributions, for example Ehm [12], Soon [27], and Roos [26]. These have been shown to generally be more accurate than Poisson approximations [8].

Another well-known approximation, given by the central limit theorem, allows the Poisson Binomial distribution function to be approximated using a standard normal distribution. For example, see Mikhailov [23], Berry [3]. This approximation was improved by Volkova [29] which corrects for skewness in the distribution of Y, and is known as the Refined Normal Approximation. It is generally accurate in estimating the true distribution for large enough n, and is very computationally efficient.

Recursive algorithms for exactly calculating the distribution function have also been proposed. Such schemes can be found in, for example, Wadycki et al. [30], Barlow & Heidtmann [1], Radke & Evanoff [25], and Chen et al. [7]. A related approach based on generating functions is provided in Belfore [2], which

utilizes the method of Barlow & Heidtmann [1]. While these recursive algorithms are capable to computing the exact distribution function, they can have expensive memory costs, and some can experience issues of numerical stability. A more in depth treatment of some recursive algorithms is provided in Hong [18].

Recently, two other exact methods have been proposed by Fernández & Williams [14] and Hong [18]. In Fernández & Williams [14] a closed form expression for both the Poisson Binomial pmf and cdf are derived using polynomial interpretation and fourier transforms. These expressions are then used to, among other things, calculate moments of the Poisson Binomial distribution. Hong [18] provides an alternative derivation of the cdf in Fernández & Williams [14] using characteristic functions, and also develops an efficient algorithm to evaluate it, which is called DFT-CF. A further study and efficiency comparison of past methods, including approximations, is also provided. The conclusion of which is that the derived DFT-CF algorithm is recommended for general computation of the Poisson Binomial distribution, and that the Refined Normal Approximation should be used for large n.

Having an efficient method to exactly compute the Poisson Binomial distribution function can be especially important for cases in which the distribution function must be computed many times. For example, Hong et al. [20] describes a situation in which bootstrapping in used to calculate prediction intervals for the cumulative number of power transformer failures by a given date. In this case the distribution function was evaluated $10,000$ times. A similar situation is described in Hong & Meeker [19]. In these cases, and for generally any exact computation of the Poisson Binomial distribution function, the DFT-CF algorithm is the current state of the art approach.

Curiously, we have observed that directly calculating the Poisson Binomial distribution function following the convolution definition of the sum of random variables is more efficient than the DFT-CF algorithm. In this paper we demonstrate that direct calculation of the distribution can be efficiently accomplished using only the definition of the sum of random variables from basic probability.

We then provide a faster algorithm which improves upon the method of direct calculation by utilizing Fast Fourier Transforms to carry out the convolutions. Both strategies are shown to provide large improvements over the DFT-CF method.

2. Calculating the Poisson Binomial distribution function

2.1. Via Direct Convolution Calculation

Suppose that we have two random variables, X_1 and X_2, and denote their sum by Y. The density function of Y is calculated by the convolution of the individual densities of X_1 and X_2.

$$P(X_1 + X_2 = k) = (f_1 * f_2)(k) = \sum_{i=0}^{k} f_1(i) \cdot f_2(k - i) \tag{2}$$

Note that this can easily be extended to any collection of random variables, $\{X_i\}_{i=1}^{n}$, so that the density function of their sum is $f_1 * ... * f_n$. For two $Bernoulli(p_i)$ random variables, calculating (2) is equivalent to calculating the linear convolution of two sequences, P_1 and P_2 where each $P_i = [1 - p_i, p_i]$. It is easy to see that this can similarly be extended to any number of $Bernoulli(p_i)$ random variables so that the density function of their sum is given by $P_1 * ... * P_n$. Thus, the density function of Y can be found by first calculating $P_1 * P_2$, and then sequentially convolving the resulting sequence with each remaining P_i until P_n has been reached. Algorithm 1 describes the process in more detail.

To gain some intuition as to why this method of computation could be efficient, consider first calculating the density function when $n = 2$, which would require computing:

$$P_1 * P_2 = [(1 - p_1)(1 - p_2), (1 - p_1)p_2 + (1 - p_2)p_1, \ p_1 p_2] \tag{3}$$

For larger n, we would then convolve P_3 with (3) and continue until all P_i have been incorporated. The jth step of the process involves convolving a length 2 sequence with a sequence of length $j + 1$. This, coupled with the fact that

$1 - p_i$ must be calculated for each i, makes it clear that $n^2 + 2n - 1$ arithmetic operations will be required and the procedure will have a time complexity of $O(n^2)$. The key, however, is that all of the required operations will be very fast as only addition and multiplication are needed. Finally, we note that due to its simplicity, directly calculating the convolution is easily implemented in any programming language.

Algorithm 1 Calculation via direct convolution

procedure DIRECTCONVOLUTION($P = (p_1, ...p_n)$)
    Set $PB = [1 - p_1, p_1]$
    for $i = 1$ to n **do**
        Set $P_i = [1 - p_i, p_i]$
        for $j = 0$ to length of PB **do**
            $new\_PB[j] = p_i \cdot PB[j] + (1 - p_i) \cdot PB[j + 1]$
            Set $PB = new\_PB$
    return PB

2.2. Via divide and conquer FFT approach

One obvious way to try to improve on the direct convolution approach is to use the well known convolution theorem which states that

$$\mathcal{F}(f_1 * f_2) = \mathcal{F}(f_1) \cdot \mathcal{F}(f_2) \qquad \text{and thus} \qquad f_1 * f_2 = \mathcal{F}^{-1}(\mathcal{F}(f_1) \cdot \mathcal{F}(f_2)) \quad (4)$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote the fourier and inverse fourier transforms respectively, and f_1 and f_2 are the densities of two random variables. Thus, the convolution of two sequences can be calculated by first transforming them into the frequency domain, multiplying the transformed sequences piecewise, and then transforming the result back. For long sequences this process can be especially useful in achieving efficiency gains since there exist many fast fourier transform procedures, perhaps the most famous of which is [9].

However, for convolutions of shorter sequences, or convolutions involving one very short sequence with another that is much longer, use of the fourier transform method may not be efficient relative to the method of direct calculation. In our case, use of the fourier transform method will incur even more disparities in efficiency relative to the method of direct calculation since a very short sequence (of length 2) will be convolved with a relatively much larger sequence many times.

Since the convolution operator is both associative and commutative we can calculate the convolutions in any order. This suggests the following scheme to calculate the distribution of Y, which can leverage both the strengths of the fourier transform method of convolution and the method of direct computation. First, split the data into M subsets, and apply direct convolution to each of them. Then, convolve each pair of sequences together using (4), resulting in $\frac{M}{2}$ sequences. Continue convolving each pair of sequences using (4) until only one final sequence remains, which will be the desired result.

In practice, of course, some care must be taken in implementing the process, since for general M there is no guarantee that the resulting subsets will be of equal size, and (4) can be inefficient when one sequence is larger than another. Furthermore, M may not divide n, and even if it does, if M is not a power of two then we will eventually encounter an odd number of sequences (perhaps multiple times). This problem is fixable by taking the remaining sequence and convolving it with one of the pairs before moving to the next step, however, this similarly runs into the same inefficiency of forcing the convolution of a potentially much larger sequence with a much smaller one.

To remedy these issues, we choose $M = 2^k$ for some $k \in \mathbb{Z}$, so that M must be a power of 2. This avoids the problem of encountering an odd number of subsets, as when the pairing occurs, we will always end up with an even number of pairs. Of course, we still have the problem that when M is chosen, we may end up with sequences with very different sizes. To fix this issue, we will assign each p_i to subsets in such a way that we maintain M subsets, but that size differences between subsets are minimized. As an example, $n = 100$

and $M = 8$ would begin with 4 subsets of size 9, and 4 subsets of size 8, since $100 \pmod 8 = 4$. The process is described in Algorithm 2.

Algorithm 2 divide and conquer Convolution via FFT and direct calculation

procedure DIVIDEANDCONQUER($P = (p_1, ...p_n)$,M)
  Set $k = log_2(M)$
  Set *subsets* = Divide $(p_1, ...p_n)$ into M subsets minimizing size disparity
  for *subset* in *subsets* **do**
    Set *convolved_subsets* = Convolve using Algorithm 1
  while $k > 1$ **do**
    Set *Poisson_Binomial* = Convolve all subset pairs in *convolved_subsets* using (4)
    Update $k = \frac{k}{2}$
  return *Poisson_Binomial*

3. Method Comparisons

In order to evaluate the efficacy of the proposed approaches, we recreate the experiment conducted in Hong [18], which first evalutes accuracy, and then the speed of several algorithms for calculating the Poisson Binomial distribution function. Since the conclusion is that the DFT-CF method is generally preferred to others, we will only compare the proposed method to DFT-CF.

The DFT-CF method is written in C and then ported to R, so we similarly coded our approaches in C and then ported the code to R to carry out the experiment. For our fourier transform implementation we choose the to use the Fastest Fourier Transform in the West ([15]), which is a library in C.

Aside from being widely used and a standard for FFT computation, it also allows the utilization of specialized algorithms which can speed up the FFT computation - for example, exploiting the fact that our input sequences consist of only real numbers. The output sequence after applying FFT will thus be Hermitian symmetric, which means only half of the FFT result will need to be

calculated.

All comparisons throughout the paper were performed with 64-bit R on a machine with a 2.50GHz processor and 8GB of RAM.

3.1. Accuracy Comparisons

To give a ground truth distribution function for comparison, we first generate 3 independent bernoulli random variables, X_1, X_2, and X_3, with distributions $Binomial(n_1, p_1), Binomial(n_2, p_2)$, and $Binomial(n_3, p_3)$, respectively, as is done in [18]. The density of their sum will then be given by:

$$P(X_1 + X_2 + X_3 = k) = \sum_{j=0}^{k} \sum_{i=0}^{j} b_{i,n_1} b_{j-i,n_2} b_{k-j,n_3} \tag{5}$$

where b_{t,n_i} is the $Binomial(p_i)$ pmf evaluated at t. Since each of the Binomial pmfs are able to be accurately calculated with already known procedures, the Poisson Binomial distribution function resulting from their sum can also be accurately calculated. Thus, Equation 5 can be used to evaluate the accuracy of the DFT-CF and proposed methods by calculating the total absolute error (TAE) between the true cdf and the calculated cdf of each method. The TAE is defined as $\sum_{k=0} |F(k) - \hat{F}(k)|$, where $F(k)$ is taken as the true cdf and $\hat{F}(k)$ is the cdf calculated by either the DFT-CF, Direct Convolution approach, or divide and conquer approach.

The results of the comparison are provided in Table 1. We report the original TAE values for the DFT-CF algorithm from [18] and denote them as DFT-CF(1), along with the TAE values computed on our own machine, which are denoted as DFT-CF(2). All methods are accurate in their ability to calculate the correct distribution, however, we note that the direct calculation approach is almost uniformly more accurate than the divide and conquer approach and DFT-CF . For the divide and conquer approach we set $M = 4$ for the cases when $n = 30$ and $M = 32$ for all others.

							TAE			
n	n_1	n_2	n_3	p_1	p_2	p_3	Direct	DC	DFT-CF(1)	DFT-CF(2)
30	10	10	10	0.500	0.500	0.500	$\mathbf{7.0 \times 10^{-15}}$	8.2×10^{-15}	1.6×10^{-14}	1.5×10^{-14}
30	10	5	15	0.500	0.500	0.500	$\mathbf{1.4 \times 10^{-15}}$	2.6×10^{-15}	1.3×10^{-14}	9.1×10^{-15}
30	10	5	15	0.010	0.500	0.990	$\mathbf{2.7 \times 10^{-15}}$	4.4×10^{-15}	1.4×10^{-14}	1.8×10^{-14}
300	100	50	150	0.010	0.500	0.990	$\mathbf{5.9 \times 10^{-14}}$	3.1×10^{-13}	1.9×10^{-12}	1.7×10^{-11}
3000	1000	500	1500	0.010	0.500	0.990	$\mathbf{8.4 \times 10^{-12}}$	9.7×10^{-12}	3.6×10^{-10}	1.2×10^{-08}
3000	1000	500	1500	0.001	0.010	0.020	9.2×10^{-11}	9.7×10^{-11}	$\mathbf{3.1 \times 10^{-11}}$	8.9×10^{-10}
3000	1000	500	1500	0.999	0.990	0.998	$\mathbf{7.6 \times 10^{-14}}$	1.3×10^{-12}	1.4×10^{-09}	1.3×10^{-07}
3000	1000	500	1500	0.001	0.500	0.999	$\mathbf{2.8 \times 10^{-13}}$	4.4×10^{-12}	3.4×10^{-10}	1.3×10^{-08}
3000	1000	500	1500	0.300	0.500	0.700	5.1×10^{-11}	$\mathbf{4.9 \times 10^{-11}}$	3.8×10^{-10}	4.1×10^{-09}

Table 1: TAE Accuracy Comparison of Direct, divide and conquer (DC), and DFT-CF Methods

3.2. Efficiency Comparisons

In order to compare the efficiency of the three approaches we first generated various sample sizes of p_i from 6 different distributions 1000 times each, as in Hong [18]. We then computed the average speed of each approach over each set of 1000 trials for various values of n. Due to the large number of combinations of n, M, and distributions for generating each p_i we only report the M with that resulted in the fastest performance for the divide and conquer approach. Further discussion on choosing M is provided in Section 3.3 As with comparing the accuracy, we report the original speed of the DFT-CF method as DFT-CF(1) and the speed of the DFT-CF on our machine as DFT-CF(2). The results of the speed comparisons are shown in Table 2.

We see that for $n \leq 200$ there is generally no preference between any of the three approaches. For $500 \leq n \leq 1000$ all methods are fast, but there is a preference for the Direct and divide and conquer approaches, which can be especially important in cases which require repeated calculation of the distribution function. When n is greater than 1000 both the Direct and divide and conquer approaches are faster than DFT-CF, with the divide and conquer approach also being faster than Direct Convolution.

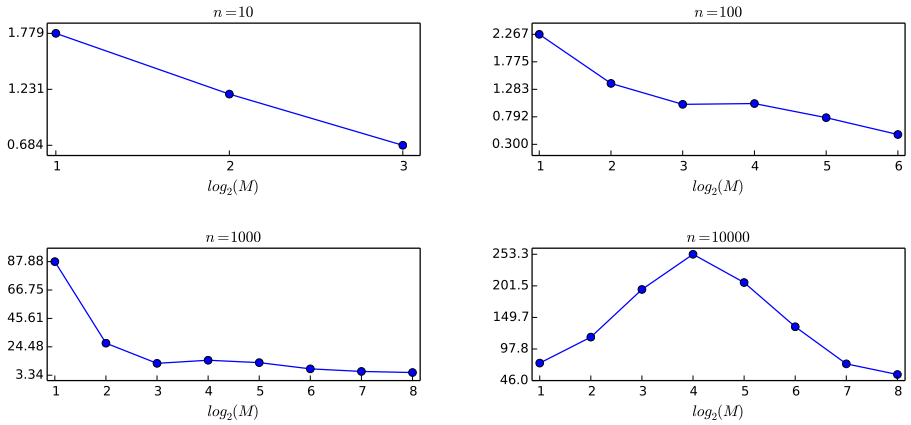

Figure 1: Ratio of average run time (in seconds) of DFT-CF(2) versus the divide and conquer (DC) approach for n different p_i generated from a Beta(3,3) distribution.

3.3. Choosing M

The divide and conquer approach requires the selection of the number of subsets, M, in which to split P, the vector containing each p_i. In order to study how M affects the efficiency of the approach relative to DFT-CF, we plot the ratio of the average speed of the DFT-CF(2) versus the divide and conquer approach from the Beta(3,3) simulation in Section 3.2 for $n = 10, 100, 1000$, and 10000 as a function of M. The results are shown in Figure 1.

We see that incorrect selection of M has the potential to increase the runtime of the divide and conquer approach, relative to DFT-CF(2), by a substantial amount. An incorrect choice of M can also cause the divide and conquer approach to be slower than DFT-CF. For example, when $n = 100$, the divide and conquer approach is approximately twice as fast as DFT-CF for $M = 2$, but becomes about 1.3 times slower than DFT-CF for $M = 64$. A similar effect is observed for $n = 10$. We note, however, that at these samples sizes both the DFT-CF and divide and conquer run almost instantly, so any speed disparities between them are inconsequential.

For $n = 1000$ and $n = 10000$ the effect of M is even more pronounced. This is especially apparent when $n = 1000$ as the divide and conquer approach is

about 88 times as fast as the DFT-CF when $M = 2$, but drops to being about 5 times as fast when $M = 256$. We note that, despite the changes in speed for different M, the divide and conquer method is uniformly faster than the DFT-CF in these two cases. That these increases in speed occur, however, is not surprising since for relatively small sample sizes increasing M will force the calculation of many fourier transforms which are likely to not be cost effective when applied to small sequences versus direct calculation.

The plots suggest that for $n \leq 1000$, the optimal choice of M is 2. For $n = 10000$ the optimal choice is $M = 16$. This information, coupled with the results of Table 2, suggest that M should be chosen such that $M = 2$ when $n \leq 1000$, and when $n > 1000$, M should be chosen in such a way that the number of p_i in each split is somewhere between 500 and 1000.

4. Discussion

4.1. Generalized Poisson Binomial distribution function

Recently, a generalization of the Poisson Binomial was proposed in Zhang et al. [32], which considers a two-valued random variable which can take values other than 0 or 1. Specifically, suppose that $X_i \sim Bernoulli(p_i)$ and let a_i and b_i be constants such that $a_i < b_i$. Then define

$$Z_i = a_i(1 - X_i) + b_i X_i \tag{6}$$

Thus, Z_i is a random variable which takes value a_i with probability $1 - p_i$ and value b_i with probability p_i. As before, the interest is in calculating the distribution function of their sum, $Y = \sum_{i=1}^{n} Z_i$, which is said to follow the aptly named Generalized Poisson Binomial distribution.

An efficient algorithm for computing the distribution function following the same approach as the DFT-CF algorithm is provided in Zhang et al. [32] and is capable of calculating the distribution for general a_i and b_i. The methods that we have presented, although not directly capable of handling general a_i and b_i, are directly applicable to calculating the distribution function of Y when

$a_i = a$ and $b_i = b$ for $i = 1, ..., n$. Of course, a direct sequence based convolution approach is possible for general a_i and b_i, but doing so would require input sequences different than we have used, which could incur a large computational burden.

We further note that when all a_i are equal and all b_i are equal, the approaches presented in this paper do not suffer from increased computational cost as $\sum a_i$ and $\sum b_i$ grow large. This is because these approaches are support agnostic, in the sense that they only rely on the probabilities of each X_i for computation, and incur no increased complexities if the support changes, provided that each X_i has the same support.

4.2. Pracitcal Considerations

Since the divide and conquer approach is capable of very fast computation for even large n, and is exact, we recommend that it be used for general computation of the Poisson Binomial distribution function, and for the Generalized Poisson Binomial distribution function when applicable. When $n \leq 1000$ either the direct approach, divide and conquer approach, or DFT-CF algorithm can be used. However, we note that if the distribution function must be evaluated many times as is the case when bootstrap procedures must be performed, or if an applicable case of the Generalized Poisson Binomial is needed, the direct convolution approach or divide and conquer approach should be used.

When $n \leq 2000$ the divide and conquer approach and direct convolution approach are generally equivalent in speed to the exceptionally fast Refined Normal Approximation of Volkova [29]. For $n > 2000$ the divide and conquer method is still recommended for single calculations, as it is still very fast and an exact method of calculation. If the distribution function needs to be calculated many times, the divide and conquer method can still be effective, but if the number of repeated calculations becomes very large and if $n > 2000$, the Refined Normal Approximation should be used.

5. Acknowledgements

The authors would like to thank Vincent Reverdy for exceptional and valuable comments and discussion about the C programming language and Matias Carrasco-Kind for suggesting the Fastest Fourier Transform in the West. Robert Brunner and William Biscarri acknowledge support from the National Science Foundation Grant No. AST-1313415. Robert Brunner and William Biscarri were also supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA0003525. Dave Zhao was supported in part by NSF grant DMS-1613005.

References

[1] Barlow, R., & Heidtmann, K. (1984). Computing k-out-of-n system reliability. *IEEE Transactions on Reliability*, *4*, 322–323.

[2] Belfore, L. A. (1995). An o (n/spl middot/(log/sub 2/(n))/sup 2/) algorithm for computing the reliability of k-out-of-n: G and k-to-l-out-of-n: G systems. *IEEE Transactions on Reliability*, *44*, 132–136.

[3] Berry, A. C. (1941). The accuracy of the gaussian approximation to the sum of independent variates. *Transactions of the american mathematical society*, *49*, 122–136.

[4] Calabrese, J. M., Certain, G., Kraan, C., & Dormann, C. F. (2014). Stacking species distribution models and adjusting bias by linking them to macroecological models. *Global Ecology and Biogeography*, *23*, 99–112.

[5] Chen, L. H. (1974). On the convergence of poisson binomial to poisson distributions. *The Annals of Probability*, (pp. 178–180).

[6] Chen, S. X., & Liu, J. S. (1997). Statis-tical applic-ations of the poisson-binomial and condi-tional bern-oulli distrib-utions. *Statistica Sinica*, (pp. 875–892).

[7] Chen, X.-H., Dempster, A. P., & Liu, J. S. (1994). Weig-hted fin-ite population sam-pling to max-imize ent-ropy. *Biometrika*, (pp. 457–469).

[8] Choi, K., Xia, A. et al. (2002). Approximating the number of successes in independent trials: Binomial versus poisson. *The Annals of Applied Probability*, *12*, 1139–1148.

[9] Cooley, J. W., & Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, *19*, 297–301.

[10] DeChant, C. M., & Moradkhani, H. (2015). On the assessment of reliability in probabilistic hydrometeorological event forecasting. *Water Resources Research*, *51*, 3867–3883.

[11] Deheuvels, P., Pfeifer, D. et al. (1986). A semigroup approach to poisson approximation. *The Annals of Probability*, *14*, 663–676.

[12] Ehm, W. (1991). Bino-mial appro-xima-tion to the pois-son bin-omial distribution. *Sta-tist-ics & Prob-ability Le-tte-rs*, *11*, 7–16.

[13] Elmore, R., & Urbaczewski, A. (2016). 'hot hand'on the pga tour: Does it exist?, .

[14] Fernández, M., & Williams, S. (2010). Closed-form expression for the poisson-binomial probability density function. *IEEE Transactions on Aerospace and Electronic Systems*, *46*, 803–817.

[15] Frigo, M., & Johnson, S. G. (2005). The design and implementation of fftw3. *Proceedings of the IEEE*, *93*, 216–231.

[16] González, J., Wiberg, M., & von Davier, A. A. (2016). A note on the poissons binomial distribution in item response theory. *Applied Psychological Measurement*, *40*, 302–310.

[17] Hoeffding, W. (1956). On the distribution of the number of successes in independent trials. *The Annals of Mathematical Statistics*, (pp. 713–721).

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

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

[20] Hong, Y., Meeker, W. Q., & McCalley, J. D. (2009). Prediction of remaining life of power transformers based on left truncated and right censored lifetime data. *The Annals of Applied Statistics*, (pp. 857–879).

[21] Le Cam, L. et al. (1960). An approximation theorem for the poisson binomial distribution. *Pacific Journal of Mathematics*, *10*, 1181–1197.

[22] Melton, C., Reuter, J. A., Spacek, D. V., & Snyder, M. (2015). Recurrent somatic mutations in regulatory regions of human cancer genomes. *Nature genetics*, *47*, 710–716.

[23] Mikhailov, V. G. (1994). On a refinement of the central limit theorem for sums of independent random indicators. *Theory of Probability & Its Applications*, *38*, 479–489.

[24] Pitacco, E. (2007). Mortality and longevity: a risk management perspective.

[25] Radke, G., & Evanoff, J. (1994). A fast recursive algorithm to compute the probability of m-out-of-n events. In *Reliability and Maintainability Symposium, 1994. Proceedings., Annual* (pp. 114–117). IEEE.

[26] Roos, B. (2001). Binomial approximation to the poisson binomial distribution: The krawtchouk expansion. *Theory of Probability & Its Applications*, *45*, 258–272.

[27] Soon, S. Y. T. (1996). Binomial approximation for dependent indicators. *Statistica Sinica*, *6*, 703–714. URL: http://www.jstor.org/stable/24305618.

[28] Steele, J. M. (1994). Le cam's ineq-uality and po-isson approx-imations. *The American Mathematical Monthly*, *101*, 48–54.

[29] Volkova, A. Y. (1996). A refinement of the central limit theorem for sums of independent random indicators. *Theory of Probability & Its Applications*, *40*, 791–794.

[30] Wadycki, W. J., Shah, B. K., Ghangurde, P. D., Dudewicz, E. J., Mantel, N., Brown, C. C., Larson, H. J., Barr, D. R., Frane, J. W., Saperstein, B., Good, I. J., & Jones, H. L. (1973). Letters to the editor. *The American Statistician*, *27*, 123–127. URL: http://www.jstor.org/stable/2683639.

[31] Wang, Y. H. (1993). On the number of successes in independent trials. *Statistica Sinica*, *3*, 295–312. URL: http://www.jstor.org/stable/24304959.

[32] Zhang, M., Hong, Y., & Balakrishnan, N. (2017). An algorithm for computing the distribution function of the generalized poisson-binomial distribution. *arXiv preprint arXiv:1702.01326*, .

		Uniform(0,1)					Beta(0.1,3)			
Size	M	DC	Direct	DFT-CF(1)	DFT-CF(2)	M	DC	Direct	DFT-CF(1)	DFT-CF(2)
10	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
20	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
50	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
100	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
200	2	**0.000**	**0.000**	0.001	0.002	2	**0.000**	**0.000**	0.001	0.001
500	2	0.001	**0.000**	0.008	0.0108	2	0.001	**0.000**	0.006	0.008
1000	2	**0.000**	0.001	0.029	0.043	2	**0.001**	**0.001**	0.022	0.032
2000	4	**0.001**	0.005	0.111	0.165	4	**0.002**	0.003	0.084	0.128
5000	8	**0.007**	0.025	0.691	1.037	4	**0.009**	0.016	0.528	0.832
10000	16	**0.018**	0.089	2.735	4.128	8	**0.022**	0.060	2.100	3.177
15000	16	**0.031**	0.182	6.176	9.319	16	**0.032**	0.127	4.736	7.262
		Beta(3,.1)					.5Beta(3,0.1)+.5Beta(0.1,3)			
Size	M	DC	Direct	DFT-CF(1)	DFT-CF(2)	M	DC	Direct	DFT-CF(1)	DFT-CF(2)
10	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
20	4	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
50	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
100	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
200	2	**0.000**	**0.000**	0.001	0.002	2	**0.000**	**0.000**	0.001	0.002
500	2	0.001	**0.000**	0.006	0.009	2	0.001	**0.000**	0.006	0.009
1000	2	**0.001**	**0.001**	0.026	0.035	2	**0.001**	**0.001**	0.024	0.035
2000	4	**0.002**	0.003	0.098	0.142	4	**0.002**	0.004	0.094	0.141
5000	4	**0.009**	0.016	0.617	0.906	8	**0.010**	0.020	0.581	0.922
10000	8	**0.022**	0.058	2.445	3.513	16	**0.021**	0.068	2.271	3.599
15000	16	**0.033**	0.127	5.517	8.031	16	**0.035**	0.148	5.141	7.960
		Beta(3,3)					.5Beta(3,10)+.5Beta(10,3)			
Size	M	DC	Direct	DFT-CF(1)	DFT-CF(2)	M	DC	Direct	DFT-CF(1)	DFT-CF(2)
10	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
20	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
50	2	**0.000**	**0.000**	**0.000**	**0.000**	2	**0.000**	**0.000**	**0.000**	**0.000**
100	2	**0.000**	**0.000**	**0.000**	0.001	2	**0.000**	**0.000**	**0.000**	0.001
200	2	**0.000**	**0.000**	0.001	0.002	2	**0.000**	**0.000**	0.001	0.002
500	2	0.001	**0.000**	0.007	0.011	2	0.001	**0.000**	0.007	0.010
1000	2	**0.000**	0.001	0.029	0.042	2	**0.000**	0.001	0.027	0.041
2000	4	**0.001**	0.004	0.109	0.165	4	**0.001**	0.005	0.104	0.168
5000	8	**0.007**	0.028	0.689	1.070	8	**0.007**	0.027	0.654	1.044
10000	16	**0.017**	0.097	2.743	4.332	16	**0.017**	0.094	2.593	4.181
15000	16	**0.024**	0.203	6.181	9.433	16	**0.028**	0.191	5.847	9.255

Table 2: Speed Comparison (in seconds) of divide and conquer (DC), Direct convolution, and DFT-CF Methods