12-2004

Algorithms for Computing the Distributions of Sums of Discrete Random Variables

D. L. Evans

Lawrence Leemis
William & Mary, lmleem@wm.edu

Recommended Citation

Evans, D. L. and Leemis, Lawrence, Algorithms for Computing the Distributions of Sums of Discrete Random Variables (2004). *Mathematical and Computer Modeling*, 40(13), 1429-1452. https://doi.org/10.1016/j.mcm.2005.01.003

PERGAMON Mathematical and Computer Modelling 0 (2004) 1–0
www.elsevier.com/locate/mcm

Algorithms for Computing the Distributions of Sums of Discrete Random Variables

D. L. Evans*
Department of Mathematics, Rose-Hulman Institute of Technology
Terre Haute, IN 47803, U.S.A.
diane.evans@rose-hulman.edu

L. M. Leemis
Department of Mathematics, The College of William and Mary
Williamsburg, VA 23187, U.S.A.
leemis@math.wm.edu

(Received March 2003; revised and accepted August 2004)

Abstract—We present algorithms for computing the probablity density function of the sum of two independent discrete random variables, along with an implementation of the algorithm in a computer algebra system. Some examples illustrate the utility of this algorithm.

Keywords—Computer algebra systems, Convolution, Probability.

1. INTRODUCTION

An important operation in probability theory is to calculate the distribution of the sum (convolution) of two independent random variables X and Y. Applications of convolutions appear in many areas of mathematics, probability theory, physics, and engineering. Most texts devote the majority of their attention to convolutions of continuous random variables, rather than discrete random variables. We focus here on the case in which X and Y are discrete random variables with integer-valued supports.

There are several approaches for computing the probability density function (PDF) of $Z = X + Y$ in the discrete case. The event $\{X + Y = z\}$, $z \in \mathbb{Z}$, can be written as the union of the disjoint events $\{X = \zeta, Y = z - \zeta\}$, $\{X = \zeta + 1, Y = z - (\zeta + 1)\}$, $\ldots$, $\{X = z - \zeta, Y = \zeta\}$, where ζ is the minimum of the union of the support values of X and Y. The PDF $f_Z(z)$ of the convolution of the PDFs of the independent random variables X and Y (with integer supports) can be computed as

* Author gratefully acknowledges support from the Clare Boothe Luce Foundation.
Both authors thank Lt. Col. A. Glen for the original code for continuous convolution, Dr. D. Nicol for his idea of using heaps to compute the convolution, Dr. J. Drew for his help with the proof of the result that appears in Appendix 1, and two referees for their helpful comments.

PII:00

$$\begin{aligned}\Pr(Z=z) &= \Pr(X+Y=z)\\ &= \sum_{k=\zeta}^{z-\zeta} \Pr(X=k,\ Y=z-k)\\ &= \sum_{k=\zeta}^{z-\zeta} \Pr(X=k)\ \Pr(Y=z-k).\end{aligned}$$

The following example illustrates the use of this discrete convolution formula.

Example 1. If X and Y are independent Poisson random variables with respective parameters λ_1 and λ_2, compute the PDF of $Z = X + Y$ [1, p. 267].

Solution. The PDF of a Poisson random variable X with parameter λ is $f_X(x) = e^{-\lambda}\,\lambda^x/x!$, $x = 0, 1, 2, \ldots$. Since $\zeta = \min\{0, 1, 2, \ldots\} = 0$, then,

$$\begin{aligned}\Pr(Z=z) &= \Pr(X+Y=z)\\ &= \sum_{k=0}^{z} \Pr(X=k,\ Y=z-k)\\ &= \sum_{k=0}^{z} \Pr(X=k)\cdot\Pr(Y=z-k)\\ &= \sum_{k=0}^{z} \frac{e^{-\lambda_1}\lambda_1^k}{k!}\cdot\frac{e^{-\lambda_2}{\lambda_2}^{z-k}}{(z-k)!}\\ &= e^{-(\lambda_1+\lambda_2)}\sum_{k=0}^{z}\frac{{\lambda_1}^k{\lambda_2}^{z-k}}{k!\ (z-k)!}\\ &= \frac{e^{-(\lambda_1+\lambda_2)}}{z!}\sum_{k=0}^{z}\frac{z!}{k!\ (z-k)!}{\lambda_1}^k{\lambda_2}^{z-k} \qquad \text{(binomial series)}\\ &= \frac{e^{-(\lambda_1+\lambda_2)}}{z!}\cdot(\lambda_1+\lambda_2)^z, \qquad z=0,1,2,\ldots.\end{aligned}$$

Thus, $Z = X + Y$ has a Poisson distribution with parameter $\lambda_1 + \lambda_2$. ∎

If one wants to sum more than two iid random variables, then, the distribution function for Z can be determined by induction [2, p. 286]. Let $Z_n = X_1 + X_2 + \cdots + X_n$ be the sum of n independent random variables with common PDF $f_X(x)$ defined on the integers. Then, the PDF of Z_1 is $f_X(x)$. We can write

$$Z_i = Z_{i-1} + X_i,$$

for $i = 2, 3, \ldots, n$. Thus, since we know the distribution of X_i is $f_X(x)$, for $i = 1, 2, \ldots, n$, we can find the distribution function of Z_n by induction. When summing more than two iid random variables, our procedure `ConvolutionIID` (see Examples 7 and 10 in Section 5) uses this inductive process to compute the PDF of the convolution.

For example, let X_1, X_2, and X_3 have PDFs $f_X(x) = 1/6$ for $x = 1, 2, \ldots, 6$. If $Z_2 = X_1 + X_2$, then the PDF $f_{Z_2}(z)$ is

$$f_{Z_2}(z) = \begin{cases} \dfrac{z-1}{36}, & z = 2, 3, \ldots, 7,\\ \dfrac{13-z}{36}, & z = 8, 9, \ldots, 12.\end{cases}$$

The PDF of $Z_3 = X_1 + X_2 + X_3$ is the convolution of the PDFs of Z_2 and X_3. For example, $\Pr(Z_3 = 4) = \Pr(Z_2 = 3)\cdot\Pr(X_3 = 1) + \Pr(Z_2 = 2)\cdot\Pr(X_3 = 2) = (2/36)\cdot(1/6) + (1/36)\cdot(1/6) = (1/72)$.

Due to the mathematical intractability in implementing the discrete convolution formula, $\sum_{k=\zeta}^{z-\zeta} \Pr(X = k, Y = z - k)$, for certain nonidentically distributed random variables [e.g., $X \sim$ Poisson (λ), $Y \sim$ geometric (p)] and the inefficiency in making computations with this formula for random variables with arbitrary supports (e.g., X with support $\{-216,\ -57,\ 23,\ 81\}$ and Y with support $\{-1002,\ -15,\ 2,\ 62,\ 211\}$), only certain convolutions can or should be computed using this formula. For random variables with arbitrary supports, the discrete convolution formula *can* be used, but it is often inefficient because one or both of the random variables have support values ranging over a large domain of nonadjacent integer values. The following example displays the inefficiency that can be encountered by using this formula, even for random variables with only a small number of support values with nonzero probability.

EXAMPLE 2. Suppose X and Y are independent discrete random variables with PDFs defined as

$$f_X(x) = \begin{cases} 0.15, & x = -3, \\ 0.25, & x = -1, \\ 0.1, & x = 2, \\ 0.3, & x = 6, \\ 0.2, & x = 8, \end{cases} \qquad f_Y(y) = \begin{cases} 0.2, & y = -2, \\ 0.1, & y = 1, \\ 0.3, & y = 5, \\ 0.4, & y = 8. \end{cases}$$

Compute the PDF of Z.

SOLUTION. The support values for Z are $z = \{-5,\ -3,\ -2,\ 0,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 9,\ 10,\ 11,\ 13,\ 14,\ 16\}$. We'll use the formula $\sum_{k=\zeta}^{z-\zeta} \Pr(X = k,\ Y = z - k)$, where $\zeta = -3$, to compute $\Pr(Z = 4)$.

$$\begin{aligned} \Pr(Z = 4) &= \sum_{k=-3}^{7} \Pr(X = k, Y = 4 - k) \\ &= \Pr(X = -3) \cdot \Pr(Y = 7) \Pr(X = -2) \cdot \Pr(Y = 6) \\ &\quad + \Pr(X = -1) \cdot \Pr(Y = 5) \Pr(X = 0) \cdot \Pr(Y = 4) \\ &\quad + \Pr(X = 1) \cdot \Pr(Y = 3) \Pr(X = 2) \cdot \Pr(Y = 2) \\ &\quad + \Pr(X = 3) \cdot \Pr(Y = 1) \Pr(X = 4) \cdot \Pr(Y = 0) \\ &\quad + \Pr(X = 5) \cdot \Pr(Y = -1) \Pr(X = 6) \cdot \Pr(Y = -2) \\ &\quad + \Pr(X = 7) \cdot \Pr(Y = -3) \\ &= 0.15 \cdot 0 + 0 \cdot 0 + 0.25 \cdot 0.3 + 0 \cdot 0 + 0 \cdot 0 + 0.1 \cdot 0 \\ &\quad + 0 \cdot 0.1 + 0 \cdot 0 + 0 \cdot 0 + 0.3 \cdot 0.2 + 0 \cdot 0 \\ &= 0.135. \end{aligned}$$

The probabilities for the other support values are computed similarly. Because of the tedious calculations needed to compute the PDF of Z by the discrete convolution formula, we'll compute it fully in the next example using moment generating functions (MGFs). ∎

Unlike the discrete convolution formula, the algorithm to be presented in this paper avoids all of the zero term computations in the construction of the PDF of Z. Also, another way to compute the PDF of Z, while avoiding the numerous zero terms, is to use the moment generating function technique.

EXAMPLE 3. Suppose X and Y are the discrete random variables defined in Example 2 and $Z = X + Y$. Find the PDF of Z using the moment generating function technique [3].

SOLUTION. Since X and Y are independent, the MGF of Z is

$$M_Z(t) = E\left(e^{t(X+Y)}\right)$$

$$\begin{aligned} &= E\left(e^{tX}e^{tY}\right) \\ &= E\left(e^{tX}\right)E\left(e^{tY}\right) \\ &= M_X(t)\,M_Y(t). \end{aligned}$$

The MGFs of X and Y, respectively, are

$$M_X(t) = E\left(e^{tX}\right) = 0.15e^{-3t} + 0.25e^{-t} + 0.1e^{2t} + 0.3e^{6t} + 0.2e^{8t}$$

and

$$M_Y(t) = E\left(e^{tY}\right) = 0.2e^{-2t} + 0.1e^{t} + 0.3e^{5t} + 0.4e^{8t},$$

for $-\infty < t < \infty$. Thus, the MGF of Z is,

$$\begin{aligned} M_Z(t) = {} & 0.03e^{-5t} + 0.05e^{-3t} + 0.015e^{-2t} + 0.045 + 0.045e^{2t} + 0.01e^{3t} \\ & + 0.135e^{4t} + 0.06e^{5t} + 0.04e^{6t} + 0.16e^{7t} + 0.02e^{9t} + 0.04e^{10t} \\ & + 0.09e^{11t} + 0.06e^{13t} + 0.12e^{14t} + 0.08e^{16t}, \end{aligned}$$

for $-\infty < t < \infty$. Thus, the PDF of Z is:

$Z = z$	-5	-3	-2	0	2	3	4	5	6	7	9	10	11	13	14	16
$f_Z(z)$	.03	.05	.015	.045	.045	.01	.135	.06	.04	.16	.02	.04	.09	.06	.12	.08

In complicated examples, especially those involving continuous random variables, using the moment generating function technique to obtain convolution functions can be more efficient than direct summation or integration. Along the same lines as the moment generating function technique, the probability generating function technique can be used for determining the PDF of the convolution of discrete random variables with nonnegative integer-valued supports. Unfortunately, the implementation of these techniques in a computer algebra system (Maple) has drawbacks when the supports of the random variables X and/or Y are not integervalued. These implementation issues are discussed in Section 4.

Besides the integration/summation and generating function methods already described, the characteristic functions of independent random variables can be used to compute convolutions [4, p. 395]. In order to use this method, though, one must know the inversion formula of the PDF of a random variable in terms of its characteristic function. The complexity of the inversion makes this method unappealing for complicated or arbitrary distributions.

The purpose of this paper is to present an algorithm for determining the distributions of convolutions of discrete random variables, especially those with finite arbitrary supports. Like the algorithm presented in [5], our algorithm handles well-known distributions, such as the binomial and Poisson. But, our algorithm was primarily written for arbitrary distributions with moderate cardinality of their support. Computer algebra systems make it feasible to determine the distributions of convolutions for these types of distributions.

Section 2 describes the algorithm for determining the PDF of the convolution of discrete random variables. The algorithm that was constructed to compute this convolution appears in Section 3. Implementation issues that arose when the algorithm was coded in a computer algebra system are given in Section 4. Section 5 provides a collection of examples that can be solved with the convolution algorithm.

2. CONCEPTUAL DEVELOPMENT

One way to compute the PDF of the convolution of the PDFs of two independent discrete random variables is by what we call the "brute force method". Let X have support $x_1, x_2, \ldots, x_n$ and Y have support $y_1, y_2, \ldots, y_m$. This method does just what the name implies—it computes

all possible sums between the support of X and the support of Y by brute force, e.g., $x_1 + y_1, x_1 + y_2, \ldots, x_1 + y_m, x_2 + y_1, x_2 + y_2, \ldots, x_n + y_{m-1}, x_n + y_m$. The sums are placed in an one-dimensional array, called s, of length $n \cdot m$. The corresponding probabilities for each of these sums, $f_X(x_1) \cdot f_Y(y_1)$, $f_X(x_1) \cdot f_Y(y_2), \ldots, f_X(x_n) \cdot f_Y(y_m)$, are stored in an one-dimensional array called Probs, also of length $n \cdot m$. The probability in position Probs_i corresponds to the sum in position s_i, $i = 1, 2, \ldots, n \cdot m$.

For example, let X and Y be the random variables introduced in Example 2. The arrays s and Probs for the random variables X and Y are

$$\begin{aligned} s = &[-5, -2, 2, 5, -3, 0, 4, 7, 0, 3, 7, 10, 4, 7, 11, 14, 6, 9, 13, 16], \\ \text{Probs} = &[0.03, 0.015, 0.045, 0.06, 0.05, 0.025, 0.075, 0.1, 0.02, 0.01, \\ &0.03, 0.04, 0.06, 0.03, 0.09, 0.12, 0.04, 0.02, 0.06, 0.08]. \end{aligned}$$

We assume that s is unsorted and may contain repeated values, such as 0, 4, and 7 in this particular example. The array s is sorted and appropriate updates are made to the corresponding elements in the array Probs. After sorting, the arrays s and Probs are

$$\begin{aligned} s = &[-5, -3, -2, 0, 0, 2, 3, 4, 4, 5, 6, 7, 7, 7, 9, 10, 11, 13, 14, 16], \\ \text{Probs} = &[0.03, 0.05, 0.015, 0.025, 0.02, 0.045, 0.01, 0.075, 0.06, 0.06, \\ &0.04, 0.1, 0.03, 0.03, 0.02, 0.04, 0.09, 0.06, 0.12, 0.08]. \end{aligned}$$

Last, the redundancies in s are removed and the appropriate probabilities corresponding to those redundancies are combined in Probs. The final arrays are

$$\begin{aligned} s = &[-5, -3, -2, 0, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 16], \\ \text{Probs} = &[0.03, 0.05, 0.015, 0.045, 0.045, 0.01, 0.135, 0.06, 0.04, 0.16, \\ &0.02, 0.04, 0.09, 0.06, 0.12, 0.08]. \end{aligned}$$

One algorithm we employ to sort the array s is *insertion sort* [6, p. 254–255]. When $n \cdot m$ is small, the simplicity of insertion sort makes it an appropriate choice. The general strategy of insertion sort is to partition the array s into two regions: sorted and unsorted. Initially, the entire array s is considered unsorted, as already discussed. At each step, insertion sort takes the first value in the unsorted region and places it in its correct position in the sorted region. The entire array s will be sorted after the final element in the $n \cdot m$ array position is inserted.

Unfortunately, for random variables X and Y with larger support sizes n and m, such as, $n = m = 10$, insertion sort becomes inefficient. Since insertion sort is an $O(N^2)$ algorithm, where $N = n \cdot m$ in our setting, it is not an appropriate method for sorting lists containing more than a hundred or so elements. For this reason, another sorting algorithm, *heapsort* [6, p. 260–262], is used to sort larger arrays. Heapsort uses a heap, which is a binary tree with special properties, to sort the array s. Heapsort is an $O(N \cdot \log(N))$ algorithm [7, p. 430].

Heapsort builds the array s as a maximum heap data structure. It then swaps the maximum element (the root) of the heap with the element in the last array position $s_{n \cdot m}$. The heap is rebuilt with the remaining unsorted elements in array positions s_1 through $s_{n \cdot m-1}$. Then, the maximum element of the new heap is swapped with the element in the second to last position of the array s, which is position $s_{n \cdot m-1}$. Now, the last two positions in s are sorted in ascending order. The heap structure is again restored with the remaining unsorted elements in array positions s_1 through $s_{n \cdot m-2}$. This swap and rebuild process repeats itself until all elements are removed from the unsorted region of the heap and placed in ascending order from the front to the back of the heap. Heapsort proved more efficient than insertion sort for large values of N. The respective CPU times for a given example using insertion sort and heapsort are provided in Section 4 for comparison.

Shellsort, an improved insertion sort, is the algorithm employed by the mathematical software package Maple to sort polynomials (Maple 9's online help guide). Since shellsort's "performance is quite acceptable in practice, even for N [number of elements] in the tens of thousands" [6, p. 260], we take advantage of Maple's sorting algorithm for polynomials (when possible) by using the moment generating function technique to compute the convolution of discrete random variables. The MGFs for X and Y, which are $M_X(t)$ and $M_Y(t)$ respectively, are first computed. Next, the product of the MGFs, $M_Z(t)$, is computed. We manipulate the terms of the MGF with Maple's `expand` procedure so that they are written in a fashion that Maple interprets as polynomials terms. For example, if the MGF is $M_Z(t) = (1/3)e^{3t} + (1/6)e^{2t} + (1/2)e^{5t}$, then `expand` $(M_Z(t))$ returns $M_Z(t)$ as $1/3(e^t)^3 + 1/6(e^t)^2 + 1/2(e^t)^5$. The terms of the resulting expanded MGF are then sorted in descending order by the constant appearing in the exponent of each e^t term. Sorting the example expression $M_Z(t)$ returns $1/2(e^t)^5 + 1/3(e^t)^3 + 1/6(e^t)^2$. The probability and support values are extracted from the terms of the expression $M_Z(t)$, and the PDF of the convolution is formed. The PDF for the example expression $M_Z(t)$ is

$$f_Z(z) = \begin{cases} \frac{1}{6}, & z = 2, \\ \frac{1}{3}, & z = 3, \\ \frac{1}{2}, & z = 5. \end{cases}$$

Although in theory this is an ideal method, Maple recognizes that expressions, such as $(1/3)e^{3.1t} + (1/6)e^{2.5t} + (1/2)e^{5.4t}$, are not truly polynomials and will incorrectly sort expressions with non-integer valued constants in the exponents. Since the MGF $M_Z(t)$ may not always have integer constants for exponents, the moment generating function technique for computing convolutions is only reasonable to use for integer supports. Using probability generating functions to compute the PDF of a convolution of random variables results in the same complications. Further implementation issues faced by moment and probability generating functions are discussed in Section 4.

As suggested by [8], the sum array s can be constructed in such a way that the next largest sum element is placed in s as it is being built. Instead of constructing the array s first and then sorting it, our algorithm constructs s by sequentially appending the next ordered element. We refer to this method as the "moving heap method", and it involves building, deleting, and inserting sums into a minimum heap data structure. A minimum heap contains its smallest element in the root, rather than its largest as in a maximum heap.

The idea behind this sorting algorithm is the construction of a two-dimensional "conceptual" array A. The array A is not instantiated to save on memory, but is helpful in explaining the nature of the algorithm. The array A has $m+1$ rows and $n+1$ columns. The array A, illustrated in Example 4, is displayed in an unusual manner in order to resemble the axes in the Cartesian coordinate system. Without loss of generality, we assume that the supports of X and Y are arranged in increasing order; i.e., $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$. The array cell (i, j) contains the sum $A_{i,j} = y_i + x_j$, for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. The cells in row $m + 1$ of A hold a 0 or 1, for each column $j = 1, 2, \ldots, n$ to indicate whether the cell in column j is "active", which means its entry is in the minimum heap. Thus, $A_{m+1,j} = 0$ or 1, for $j = 1, 2, \ldots, n$. Likewise, the cells in column $n + 1$ of A also hold a 0 or 1, for each row $i = 1, 2, \ldots, m$ to indicate whether the cell in row i is "active"; i.e., $A_{i,n+1} = 0$ or 1, for $i = 1, 2, \ldots, m$. The $(m + 1, n + 1)$ cell is not used in the algorithm. Example 4 illustrates what is meant by an "active" cell.

Since $x_j < x_{j+1}$, for $j = 1, 2, \ldots, n - 1$ and $y_i < y_{i+1}$, for $i = 1, 2, \ldots, m - 1$, the entry in cell (i, j) is always guaranteed to be less than both the entries in cells $(i + 1, j)$ and $(i, j + 1)$; i.e., $A_{i,j} < A_{i+1,j}$ and $A_{i,j} < A_{i,j+1}$. This result, along with other properties of the array A proven

in Appendix 1, allow the algorithm to move the smallest number of candidate entries for the next largest sum from the array A to the minimum heap. Thus, this algorithm moves from the southwest cell to the northeast cell of the array A placing the next largest sum into s after first placing the competing sums into a minimum heap.

Since this process and its intricacies are best explained by an example, we'll reintroduce X and Y, the random variables from Example 2.

EXAMPLE 4. Let X and Y have PDFs:

$$f_X(x) = \begin{cases} 0.15, & x = -3, \\ 0.25, & x = -1, \\ 0.1, & x = 2, \\ 0.3, & x = 6, \\ 0.2, & x = 8, \end{cases} \qquad f_Y(y) = \begin{cases} 0.2, & y = -2, \\ 0.1, & y = 1, \\ 0.3, & y = 5, \\ 0.4, & y = 8. \end{cases}$$

Use the "moving heap method" to determine the PDF of $Z = X + Y$.

SOLUTION. Construct the 5×6 array A. Set $A_{i,n+1} = A_{i,6} = 0$, for $i = 1, 2, 3, 4$ and $A_{m+1,j} = A_{5,j} = 0$, for $j = 1, 2, 3, 4, 5$. The smallest value in A is positioned in cell $(1, 1)$ and is $A_{1,1} = y_1 + x_1 = -5$. The algorithm designates the cell $(1, 1)$ as an "active" cell in A by setting $A_{m+1,1} = A_{5,1} = 1$ and $A_{1,n+1} = A_{1,6} = 1$. The zeros in the other cells of row five and column six remain. Figure 1 displays this initial array. The entries of A increase in value as one moves up, to the right, or a combination of both (as in the Cartesian coordinate system).

row 5		1	0	0	0	0	
row 4	8						0
row 3	5						0
row 2	1						0
row 1	−2	−5					1
		−3	−1	2	6	8	
		col 1	col 2	col 3	col 4	col 5	col 6

Figure 1. Array A with active cell (1, 1), which contains the entry $A_{1,1} = -5$.

As in the brute force method, let the one-dimensional array s of length $n \cdot m$ hold the sums of the supports of the random variables X and Y. The corresponding probabilities for each of these sums will again be stored in the one-dimensional array called Probs, also of length $n \cdot m$. Clearly, the first (smallest) sum to be placed in the first position of array s is $A_{1,1}$. Accordingly, $f_X(x_1) \cdot f_Y(y_1) = 0.03$ is placed in the Probs array in its first position, Probs_1. After setting $s_1 = A_{1,1} = -5$ and $\text{Probs}_1 = \Pr(Z = A_{1,1}) = f_X(x_1) \cdot f_Y(y_1) = 0.03$, the cell $(1, 1)$ becomes inactive. In order to reflect the absence of an element in the first row and first column, reset $A_{m+1,1} = A_{5,1} = 0$ and $A_{1,n+1} = A_{1,6} = 0$. The next two cells to become "active" (i.e., these cells may contain the next largest sum) in the array A are $A_{1,2} = y_1 + x_2 = -3$ and $A_{2,1} = y_2 + x_1 = -2$. Since cell $(1, 2)$ in A is now active, reset $A_{1,6} = 1$ and set $A_{5,2} = 1$. Similarly, since cell $(2, 1)$ is active, set $A_{2,6} = 1$ and reset $A_{5,1} = 1$. The purpose of these ones and zeros along the boundary of the A array is to assure that there is no more than one active cell in each row and column. Figure 2 contains the current state of array A.

The values $A_{1,2}$ and $A_{2,1}$ are used to construct a minimum heap H. Informally, a *heap* is a complete binary tree with a special ordering property of its nodes. A complete binary tree is a

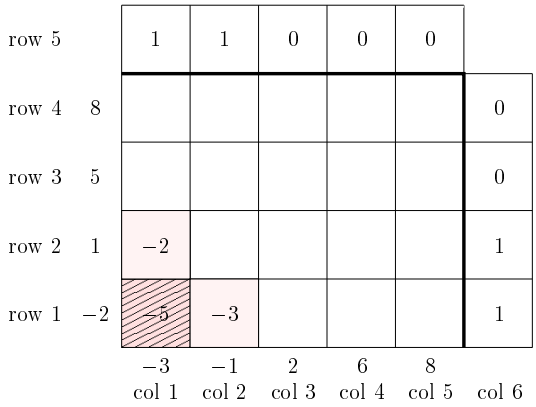

Figure 2. Array A after $A_{1,1}$ has been removed and added to the one-dimensional sum array s. The cells $(1, 2)$ and $(2, 1)$ are active, as indicated by the ones in cells $(1, 6)$, $(5, 2)$, $(2, 6)$, and $(5, 1)$.

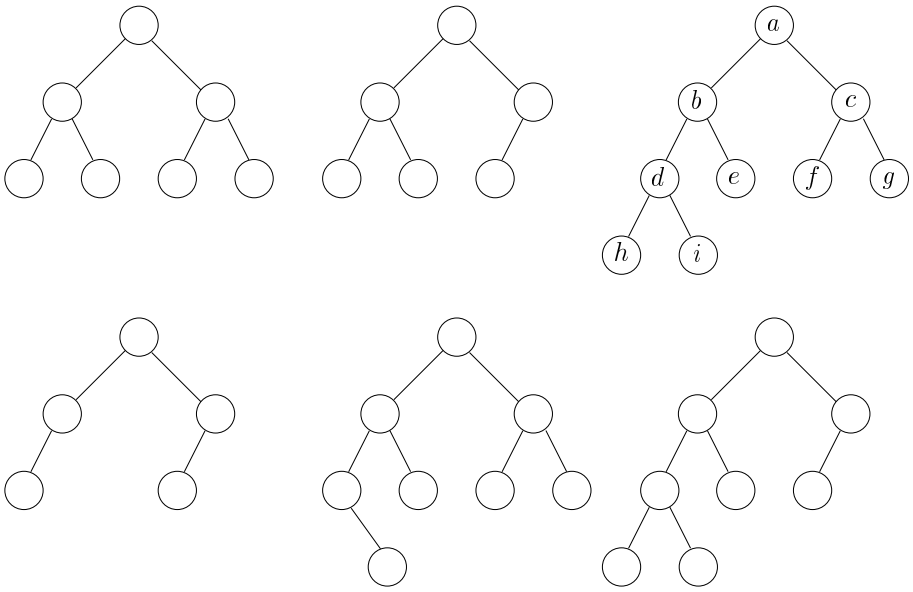

Figure 3. Six binary trees. The top three trees are complete binary trees and the bottom three are not.

tree that is completely filled with the possible exception of the bottom level, which is filled from left to right. Figure 3 contains illustrations of structures which are and are not complete binary trees. Each node of the tree has one parent, except the *root* of the tree, which has no parent. In a *minimum* heap, the smallest element of the heap is contained in its root. In the upper right tree in Figure 3, a is the root of the tree. Nodes b and c are a's *children*, where b is the *left child* and c is the *right child*. (According to the definition of a complete binary tree, when a node above the bottom level of the tree has only one child, it must be a left child.) Node b is the parent to nodes d and e. The *height* of a tree is the number of nodes from the root to a node at the bottom level of the tree. For example, the heights of the top trees in Figure 3 are three, three, and four, respectively. A complete binary tree of height h has between 2^h and $2^{h+1} - 1$ nodes [7, p. 496].

Thus, a minimum heap is a complete binary tree with the special ordering property that each parent node contains a value less than or equal to the values in its childrens' nodes. Because of this ordering property, the smallest value in a minimum heap will always be at the root.

The binary heap H formed with the values $A_{1,2}$ and $A_{2,1}$ is in Figure 4. The next sum to be entered into s in position s_2 is the root of the heap. Since $A_{1,2} = -3$ is the root, it is removed from the heap H and placed in s_2, while its corresponding probability is placed in Probs_2. Because

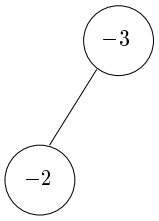

Figure 4. Heap H containing entries $A_{1,2} = -3$ and $A_{2,1} = -2$.

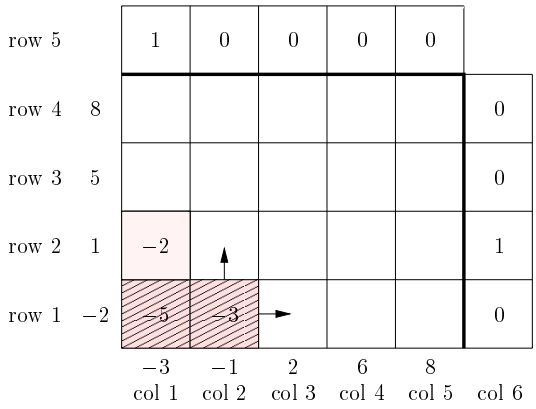

Figure 5. Array A after $A_{1,\,2} = -3$ is removed and appended to s. Cell (2, 1) is the only active cell. Candidates to become active are cells (1, 3) and (2, 2). Cell (2, 2) cannot become active since row two already contains an active cell.

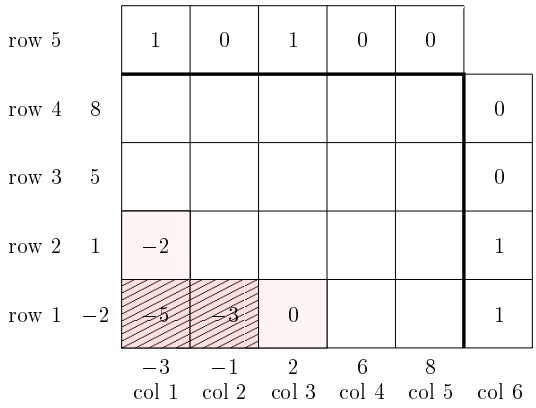

Figure 6. Array A with active cells $(1, 3)$ and $(2, 1)$.

the entry $A_{1,2}$ is removed from the array A, reset $A_{1,6} = 0$ and $A_{5,2} = 0$ to indicate that row one and column two no longer contain an active cell. After these changes, array A is displayed in Figure 5.

After setting cell $(1, 2)$ to inactive, the two cells that may enter into the array A (if the corresponding row and column do not already contain an active cell) are cells $(3, 1)$ and $(2, 2)$. Since row two contains an active cell, then, entry $A_{2,2}$ is not activated since its sum is greater than $A_{2,1}$. However, cell $(1, 3)$ does become active, and its entry is $A_{1,3} = y_1 + x_3 = 0$. Hence, $A_{1,6} = 1$ and $A_{5,3} = 1$. After these changes, array A is displayed in Figure 6.

The entry $A_{1,3}$ is inserted into the heap H, and the heap is rebuilt to fulfill its ordering property. After the addition of $A_{1,3}$, the heap H is displayed in Figure 7. The minimum element, $A_{2,1}$, is removed from the root of the heap and placed in the sum array s in position s_3. Its corresponding probability is placed in Probs_3.

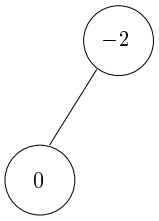

Figure 7. Heap H containing entries $A_{2,2} = -2$ and $A_{1,3} = 0$.

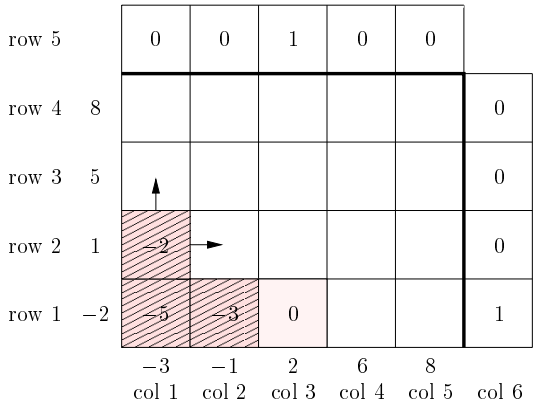

Figure 8. Array A after $A_{2,2} = -2$ is removed. Cell $(3, 1)$ is the only active cell. Candidates to become active are cells $(2, 2)$ and $(3, 1)$.

The two cells that may enter the array A after the removal of the $A_{2,1}$ entry are in cells $(2, 2)$ and $(3, 1)$, as indicated by the arrows in Figure 8. Both cells $(2, 2)$ and $(3, 1)$ become active, and their values are $A_{2,2} = y_2 + x_2 = 0$ and $A_{3,1} = y_3 + x_1 = 2$. Hence, $A_{2,6} = 1$, $A_{5,2} = 1$, $A_{3,6} = 1$, and $A_{5,1} = 1$, as displayed in Figure 9. Entries $A_{2,2}$ and $A_{3,1}$ are inserted into the heap H, and H is again rebuilt. Its structure is displayed in Figure 10.

Moving ahead to the seventeenth pass through the construction of the array A, its appearance is displayed in Figure 11. $A_{3,4} = 11$ is placed in s_{17}, and values $A_{3,5} = 13$ and $A_{4,4} = 14$ are activated in the array A and inserted into the heap H. Since $A_{3,5}$ is the root of the heap, it is deleted and placed in s_{18}. No new element is allowed to enter the heap, so the root element of the heap is now $A_{4,4} = 14$, and it is removed and placed in s_{19}. The last entry to be activated is $A_{4,5} = 16$, and it is placed in position s_{20} of the sum array s.

Thus, after twenty iterations of this process, s and Probs arrays are

$$\begin{aligned} s = &[-5, -3, -2, 0, 0, 2, 3, 4, 4, 5, 6, 7, 7, 7, 9, 10, 11, 13, 14, 16] , \\ \text{Probs} = &[0.03, 0.05, 0.015, 0.025, 0.02, 0.045, 0.01, 0.075, 0.06, 0.06, \\ &0.04, 0.1, 0.03, 0.03, 0.02, 0.04, 0.09, 0.06, 0.12, 0.08] . \end{aligned}$$

which are the same arrays encountered by using the moment generating function technique. The redundancies are removed from s and the appropriate probabilities are combined in Probs to complete the algorithm. (This could have been embedded into the iterative steps of the algorithm to save memory.) Thus, the PDF of $Z = X + Y$ is the same as determined in Example 3. ∎

3. ALGORITHM

The algorithm for the procedure `Convolution(X, Y)` returns the PDF of the convolution of the PDFs of the random variables X and Y. A brief description of the algorithm for discrete random variables follows.

If X and Y are discrete, their supports, finite or infinite, dictate which of the methods described in Section 2 is used to compute the convolution. The convolution of the PDFs of X and Y with

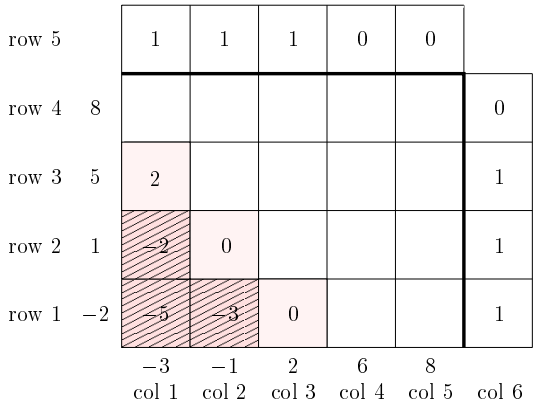

Figure 9. Array A with active cells $(1,3)$, $(2,2)$, and $(3,3)$.

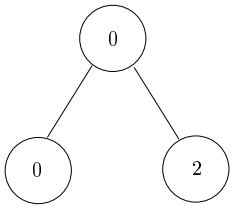

Figure 10. Heap H with entries $A_{1,3} = 0$, $A_{2,2} = 0$, and $A_{1,3} = 2$.

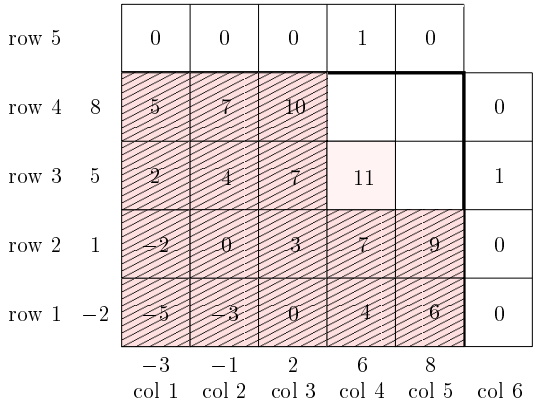

Figure 11. Array A with its seventeenth active cell $(3,4)$.

finite support is computed either using the `BruteForceMethod` or `MovingHeapMethod` procedures, whose algorithms appear in Appendices 2 and 3, respectively. The PDF of the convolution of Z is stored in a list-of-sublists format. The list of elements $f(z_1), f(z_2), \ldots, f(z_{n \cdot m})$ are the probability values of Z, while $z_1, z_2, \ldots, z_{n \cdot m}$ are its support values. The one-dimensional array s is created to contain the unique sums extracted from the array A. Similarly, the one-dimensional array Probs is created to hold the probability values corresponding to the sums in s.

The zeros (if any) at the end of the Probs array do not represent probability values; they correspond to the zeros in the Probs array that are not support values. These extra zeros indicate that there are redundant values in the support of Z. The nonzero probability values are removed from Probs and placed in the array FinalProbs. The support values that correspond to the removed probability values are removed and placed in the array FinalSupport.

If the supports of the random variables X and Y are infinite, either the discrete convolution formula is used to compute the convolution or the APPL ("A Probability Programming Language" that is discussed in the next section) procedure `MGF` is used to determine the MGF of the product of X and Y.

If either X or Y has infinite support, while the other has finite support, the product of their MGFs is returned. If both X and Y have infinite support and the discrete convolution formula formed with their PDFs results in an intractable sum, then the product of their MGFs is returned. Otherwise, the discrete convolution formula is used to determine the convolution of their PDFs.

Unless the MGF for Z or the PDF of the convolution for X and Y (with $n = 1$ and $m = 1$) has already been returned, the PDF $f_Z(z)$ is returned.

Procedure `Convolution`: Computes the PDF of the convolution of the PDFs of two independent random variables (indentation indicates nesting).
`Input`: The random variables X and Y. Assume the random variables X and Y are input in their PDF forms. The support of X is Ω_X and the support of Y is Ω_Y.
`Output`: The PDF of $Z = X + Y$.

If X and Y are continuous
    $f_Z(z) \leftarrow \int_{-\infty}^{\infty} f_X(z - y)\, f_Y(y)\, dy$
Else if X and Y are discrete
    If X and Y have finite support
        $n \leftarrow |\Omega_X|$
        $m \leftarrow |\Omega_Y|$
        If $n \cdot m \leq 100$
            $Z \leftarrow$ `BruteForceMethod`(X, Y)
        Else
            $Z \leftarrow$ `MovingHeapMethod`(X, Y)
        If ($n = 1$ and $m = 1$) [i.e. $f_Z(z) = 1$ for $z = c \in \mathbb{R}$]
            return$(f_Z(z))$
        Dimension $s[n \cdot m]$ [Create the sums array s]
        Dimension $Probs[n \cdot m]$ [Create the probability array $Probs$]
        For $i \leftarrow 1$ to $n \cdot m$
            $s_i \leftarrow 0$
            $Probs_i \leftarrow 0$
        $s_1 \leftarrow z_1$
        $s_2 \leftarrow z_2$
        $Probs_1 \leftarrow f(z_1)$
        $k \leftarrow 2$
        $j \leftarrow 2$
        While $(k < n \cdot m)$ do
            $Probs_j \leftarrow Probs_j + f(z_k)$
            If $z_k \neq z_{k+1}$ then [Eliminate redundant support values]
                $j \leftarrow j + 1$
                $s_j \leftarrow z_{k+1}$
            $k \leftarrow k + 1$
        $Probs_j \leftarrow Probs_j + f(z_k)$
        $NumZeros \leftarrow 0$
        For $i \leftarrow n \cdot m$ to 1 by -1 while $Probs_i = 0$
            $NumZeros \leftarrow NumZeros + 1$
        Dimension $FinalProbs[1, n \cdot m - NumZeros]$
        Dimension $FinalSupport[1, n \cdot m - NumZeros]$
        For $i \leftarrow 1$ to $(n \cdot m - NumZeros)$
            $FinalProbs_i \leftarrow Probs_i$
            $FinalSupport_i \leftarrow s_i$

$f_Z(z) \leftarrow [FinalSupport, FinalProbs]$
Else if (X or Y has infinite support or
X and Y have infinite support with intractable discrete convolution sum)
$mgfx \leftarrow \texttt{MGF}(X)$
$mgfy \leftarrow \texttt{MGF}(Y)$
$mgfprod \leftarrow mgfx \cdot mgfy$
return($mgfprod$)
Else [Discrete convolution formula]
$f_Z(z) \leftarrow \sum_{k=0}^{z}(f_X(z)) \cdot (f_Y(z-k))$
Else
print("ERROR: X and Y must both be continuous or discrete")
return
return($f_Z(z)$)

4. IMPLEMENTATION

The algorithm for the `Convolution` procedure has been implemented in Maple. This Maple procedure is one of over 30 procedures included in a probability package developed to automate the naming, processing, and applications of random variables. The software package is referred to as "A Probability Programming Language" (APPL) and is described in [9]. Several other APPL procedures, such as `Mean`, `Variance`, and `PDF`, will also be used in this paper.

The data structure used to represent the distribution of a random variable is a "list of three sublists." The first sublist contains the PDF, CDF, SF, HF, CHF, or IDF of the distribution. For explanation purposes, we will assume it contains the PDF. The function in the first sublist can either be written symbolically as a function, $\left[\mathtt{x} \rightarrow \binom{3}{\mathtt{x}}(2/5)^{\mathtt{x}}(3/5)^{3-\mathtt{x}}\right]$, or numerically as a list of probability values, $[27/125, 54/125, 36/125, 8/125]$. The second sublist contains the support values of the PDF. If the PDF of X is entered as a function in the first sublist, then, its support Ω is entered as a range in the second sublist as $[\min\{\Omega\} \ldots \max\{\Omega\}, k]$, where k is a positive increment with a default value of one. If $X \sim$ binomial$(3, 2/5)$, for example, then, the first and second sublists written in symbolic function notation are $\left[\mathtt{x} \rightarrow \binom{3}{\mathtt{x}}(2/5)^{\mathtt{x}}(3/5)^{3-\mathtt{x}}\right], [0 \ldots 3]$. If the PDF is entered as a list of probability values in the first sublist, then, each corresponding support value is listed in the second sublist. Again, if $X \sim$ binomial$(3, 2/5)$, then, the first and second sublists entered as numeric lists are $[(27/125), (54/125), (36/125), (8/125)], [0, 1, 2, 3]$. Last, the third sublist contains two strings. The first string, which is either `"Continuous"` or `"Discrete"`, indicates whether the random variable is continuous or discrete. The second string is `"PDF"`, `"CDF"`, `"SF"`, `"HF"`, `"CHF"`, or `"IDF"`, which indicates the type of function in the first sublist.

One piece of the algorithm for the `Convolution` procedure involves computing every possible sum between the support of the random variables X and Y. At the same time, the probabilities for those sums are calculated. This is called the "brute force method". The list of sums are ordered and redundancies combined. The corresponding probability values are repositioned to match their corresponding sums. One important reason for sorting the sums is that all other APPL procedures assume that discrete distributions, written in their list-of-sublists form, have supports listed in increasing order without repeated values. To be consistent with the APPL language and textbooks, the sums are sorted. Also, placing the values of the support into a list in sorted order means that tied $y_i + x_j$ values *can* be combined dynamically as the algorithm proceeds.

The first sorting method used to sort the list of sums was insertion sort. It was chosen because of its straightforward code and efficiency in computing convolutions when the support sizes of the random variables are small. Unfortunately, as the supports of random variables grow larger (e.g., random variables with 50 random numbers chosen on the interval $(-1, 1)$ as a support), the

time used to compute the PDF of the convolution becomes unreasonably large. A faster sorting method for larger list sizes is heapsort. Heapsort is employed in the `Convolution` procedure for sorting the list of sums created by the "brute force method".

Maple uses Shellsort to sort polynomials. In order to use the Shellsort procedure in Maple, the MGFs of X and Y need to be computed. The product of the MGFs of X and Y is an expression composed of exponential terms e^{kt}, where $k \in \mathbb{R}$, $t > 0$. Letting $u = e^t$, the MGF of the product can be rewritten as a polynomial-type expression. For example, if $u = e^t$, then, $M_Z(t) = (1/3)e^{3t} + (1/6)e^{2t} + (1/2)e^{5t}$ can be rewritten $M_Z(t) = (1/3)u^3 + (1/6)u^2 + (1/2)u^5$. The Shellsort procedure sorts the polynomial-like expression, and the PDF of the convolution of X and Y is retrieved from this expression. Instead of MGFs, probability generating functions (PGFs) can be used in the process. MGFs were chosen over PGFs since PGFs can only be formed when the discrete distribution has nonnegative integer support.

The method of computing convolutions via their MGFs was abandoned after realizing that Maple can only sort "true" polynomial expressions in which variables are raised to nonnegative integer powers. Thus using MGFs would be too restrictive for a Maple implementation. Maple is unable to sort an expression that has nonintegers in the powers of the polynomial terms. For example, Maple cannot sort the expression $1/3\,(e^t)^{3/2} + 1/6\,(e^t)^{1/2} + 1/2\,(e^t)^{5.5}$. Since the `Convolution` procedure was intended to be used on arbitrary distributions, which could indeed have negative, non-integer supports, the MGF method was removed from the procedure `Convolution`. The extra time involved in checking for appropriate values also affected the algorithm's efficiency.

Unfortunately, for random variables with large finite support sizes, heapsort was also inefficient. Nicol [8] suggested constructing a heap dynamically and sorting the list of sums sequentially, instead of waiting until the list of sums is built and then sorting it. The heap will always contain $\min\{m, n\}$ or fewer entries. After constructing the algorithm for this method, which is called `MovingHeapMethod`, the `Convolution` procedure was tested on random variables with large supports by using the `BruteForceMethod` with insertion sort, the `BruteForceMethod` with heapsort, and the `MovingHeapMethod`. A brief comparison analysis of the three methods suggested that `MovingHeapMethod` yielded the best times for computing convolution of random variables with large supports. Test cases of random variables with increasing support sizes were performed to confirm this assumption.

The test involved generating random numbers between -1 and 1 and making them the support values for the random variables X and Y. The supports of the random numbers were sorted and placed into the second sublist in the list-of-sublists format for the random variables to conform to the system of the APPL language. The probabilities, which had no effect on the efficiency of the different algorithms, were assigned to be equally-likely for all support values; i.e., if X's support consisted of 50 values, then each support value's probability was 1/50. The times for determining the PDF of the convolution of random variables of increasing support sizes using a 266 MHz Pentium II personal computer appears in Table 1.

When either one or both random variables' supports are infinite, either the convolution of their PDFs is computed via the discrete convolution formula or the MGF of their product is determined. If one of the random variables has infinite support, while the other has finite support, the MGF of their product is returned. At this time, APPL does not contain a procedure to convert the MGF of a random variable to its PDF form. In future work, this recognition process may become an APPL procedure.

Two random variables with infinite support does not guarantee that the PDF of their convolution can be determined by the discrete convolution formula. Only tractable summations, such as the convolution formula for two Poisson random variables as in Example 1, can be computed. This means that instead of determining the PDF for the convolution of the PDFs of some random variables, such as a Poisson with parameter $\lambda = 5$ and a geometric with parameter $p = 0.3$, the `Convolution` procedure only computes the product of their MGFs.

Table 1. CPU times (in seconds) for the convolution of random variables X and Y by the `BruteForceMethod` with insertion sort, the `BruteForceMethod` with heapsort, and the `MovingHeapMethod` for arbitrary distributions with arbitrary support values ranging in increasing value from -1 to 1.

Support size of X and Y	`BruteForceMethod` with insertion sort	`BruteForceMethod` with heapsort	`MovingHeapMethod`
50	70.5	10.6	15.3
60	143.1	18.1	24.0
70	313.3	29.1	34.6
80	518.0	45.5	50.0
90	824.0	69.9	69.3
95	1050.5	85.3	80.6
100	1263.5	101.3	93.5
110	2037.6	153.2	123.3
120	2897.4	201.7	163.0
125	3283.5	257.5	173.9
130	–	284.8	201.6
140	–	394.8	236.4
150	–	541.1	320.1
160	–	728.8	377.3
170	–	969.0	454.6
175	–	1127.9	506.5
180	–	1319.1	578.5
190	–	1723.2	671.8
200	–	2210.3	829.0

5. EXAMPLES

The following examples use the algorithm described in Section 3 to determine the PDF of the convolution of independent random variables. Examples for a variety of random variables are provided to illustrate the utility of the algorithm. Returning first to Examples 1–4 introduced in Section 1 of this paper, we can use the `Convolution` procedure to determine their solutions.

Example 1 Revisited. If X and Y are independent Poisson random variables with respective parameters λ_1 and λ_2, compute the PDF of $Z = X + Y$.

Solution. In APPL, define `X` as a Poisson random variable with parameter `lambda1` and `Y` as a Poisson random variable with parameter `lambda2`. The Poisson random variable is also predefined in APPL. The PDF $Z = X + Y$ is found with the statements,

```
> X := PoissonRV(lambda1);
> Y := PoissonRV(lambda2);
> Z := Convolution(X, Y);
```

which returns the PDF of Z as:

$$\left[\left[z \to \frac{(z+1)\, e^{-\lambda_1-\lambda_2}\, (\lambda_2+\lambda_1)^z}{(z+1)!}\right], [0 \ldots \infty], [\text{ “Discrete”, “PDF”}]\right].$$

Using the Maple `simplify` procedure, the resulting PDF after simplification is

$$f(z) = \frac{e^{-\lambda_1-\lambda_2}\, (\lambda_2+\lambda_1)^z}{\Gamma(z+1)}, \qquad z = 0, 1, \ldots,$$

which is easy to recognize in its standard form as

$$f(z) = \frac{e^{-\lambda_1-\lambda_2}\, (\lambda_1+\lambda_2)^z}{z!}, \qquad z = 0, 1, \ldots. \quad \blacksquare$$

We are fortunate in this example that MAPLE can compute the PDF by the discrete convolution formula by simplifying the sum $\sum_{k=0}^{x}(\lambda_1{}^k e^{-\lambda_1}\lambda_2{}^{x-k}e^{-\lambda_2}/k!\,(x-k)!)$. Unfortunately, MAPLE can only simplify certain expressions, so, in some instances, we cannot compute the PDF by the discrete convolution formula.

EXAMPLES 2–4 REVISITED. X and Y are independent discrete random variables with PDFs defined as:

$$f_X(x) = \begin{cases} 0.15, & x = -3, \\ 0.25, & x = -1, \\ 0.1, & x = 2, \\ 0.3, & x = 6, \\ 0.2, & x = 8, \end{cases} \qquad f_Y(y) = \begin{cases} 0.2, & y = -2, \\ 0.1, & y = 1, \\ 0.3, & y = 5, \\ 0.4, & y = 8. \end{cases}$$

Find the PDF of Z.

SOLUTION. Define the random variables `X` and `Y` in APPL's list-of-sublists format. Compute the PDF of $Z = X + Y$ with the following statements,

```
> X := [[0.15, 0.25, 0.1, 0.3, 0.2], [-3, -1, 2, 6, 8], ["Discrete", "PDF"]];
> Y := [[0.2, 0.1, 0.3, 0.4], [-2, 1, 5, 8], ["Discrete", "PDF"]];
> Z := Convolution(X, Y);
```

which returns the PDF of Z as

$$[[0.03, 0.05, 0.015, 0.045, 0.045, 0.01, 0.135, 0.06, 0.04, 0.16, 0.02, 0.04, 0.09, 0.06, 0.12, 0.08]\,,$$
$$[-5, -3, -2, 0, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 16]\,,\ [\text{“Discrete”}, \text{“PDF”}]]\,.$$

EXAMPLE 5. Let X and Y be independent random variables; X assumes three possible values 0, 1, 3 with probabilities 1/2, 3/8, and 1/8, and Y assumes two possible values 0 and 1 with probabilities 1/3, 2/3. Find the PDF of the random variable $Z = X + Y$ [10, p. 136].

SOLUTION. By hand, we can compute the PDF of Z with probability generating functions. The PGFs G of X and Y, respectively, are,

$$G_X(t) = \frac{1}{8}t^3 + \frac{3}{8}t + \frac{1}{2}, \qquad -\infty < t < \infty,$$

$$G_Y(t) = \frac{2}{3}t + \frac{1}{3}, \qquad -\infty < t < \infty.$$

Thus, the PGF of $Z = X + Y$ is,

$$G_Z(t) = \frac{1}{12}t^4 + \frac{1}{24}t^3 + \frac{1}{4}t^2 + \frac{11}{24}t + \frac{1}{6}, \qquad -\infty < t < \infty,$$

and hence, the PDF of Z is,

$$f_Z(z) = \begin{cases} \frac{1}{6}, & z = 0, \\ \frac{11}{24}, & z = 1, \\ \frac{1}{4}, & z = 2, \\ \frac{1}{24}, & z = 3, \\ \frac{1}{12}, & z = 4. \end{cases}$$

In APPL, define `X` and `Y` as list-of-sublists and then apply the `Convolution` procedure to achieve the same result.

```
> X := [[1 / 2, 3 / 8, 1 / 8], [0, 1, 3], ["Discrete", "PDF"]];
> Y := [[1 / 3, 2 / 3], [0, 1], ["Discrete", "PDF"]];
> Z := Convolution(X, Y);
```

Other measures, such as the mean and variance, of a distribution can be found with the use of additional APPL procedures, as seen in the next example.

Example 6

Let X_1 and X_2 be observations of a random sample of size $n = 2$ from a distribution with PDF $f(x) = (x/6)$, $x = 1, 2, 3$. Find the PDF of $Y = X_1 + X_2$, and determine the mean and variance of the sum [11, p. 297].

SOLUTION. The PGF of X_1 and X_2 is

$$G_{X_1}(t) = G_{X_2}(t) = \frac{1}{6}t + \frac{1}{3}t^2 + \frac{1}{2}t^3, \qquad -\infty < t < \infty.$$

Thus, $G_Y(t)$ is

$$G_Y(t) = \frac{1}{4}t^6 + \frac{1}{3}t^5 + \frac{5}{18}t^4 + \frac{1}{9}t^3 + \frac{1}{36}t^2, \qquad -\infty < t < \infty.$$

and $f_Y(y)$ is

$$f_Y(y) = \begin{cases} \frac{1}{36}, & y = 2, \\ \frac{1}{9}, & y = 3, \\ \frac{5}{18}, & y = 4, \\ \frac{1}{3}, & y = 5, \\ \frac{1}{4}, & y = 6. \end{cases}$$

The mean and the variance of Y, respectively, are

$$\mathrm{E}[Y] = G'_Y(t)\Big|_{t=1} = \left[\frac{3}{2}t^5 + \frac{5}{3}t^4 + \frac{10}{9}t^3 + \frac{1}{3}t^2 + \frac{1}{18}t\right]_{t=1} = \frac{14}{3}$$

and

$$G'_Y(1) + G'_Y(1) - [G'_Y(1)]^2 = \frac{10}{9}.$$

In APPL, the mean and variance of Y are computed with the statements,

```
> X := [[x -> x / 6], [1 ..  3], ["Discrete", "PDF"]];
> Y := [[1 / 3, 2 / 3], [0, 1], ["Discrete", "PDF"]];
> Mean(Y);
> Variance(Y);
```

EXAMPLE 7. Find the probability of obtaining a total of 14 in a single toss of four dice [3, p. 230].

SOLUTION. Let X be the PDF $f_X(x) = 1/6$, $x = 1, 2, \ldots, 6$. The PGF G of X is

$$G_X(t) = \frac{1}{6}t^6 + \frac{1}{6}t^5 + \frac{1}{6}t^4 + \frac{1}{6}t^3 + \frac{1}{6}t^2 + \frac{1}{6}t, \qquad -\infty < t < \infty.$$

The PDF of $Z = X_1 + X_2 + X_3 + X_4$ can be found by computing $[G_X(t)]^4$, which is

$$\begin{aligned}[G_X(t)]^4 = {} & \frac{1}{1296}t^{24} + \frac{1}{324}t^{23} + \frac{5}{648}t^{22} + \frac{5}{324}t^{21} + \frac{35}{1296}t^{20} \\ & + \frac{7}{162}t^{19} + \frac{5}{81}t^{18} + \frac{13}{162}t^{17} + \frac{125}{1296}t^{16} + \frac{35}{324}t^{15} + \frac{73}{648}t^{14} \\ & + \frac{35}{324}t^{13} + \frac{125}{1296}t^{12} + \frac{13}{162}t^{11} + \frac{5}{81}t^{10} + \frac{7}{162}t^9 + \frac{35}{1296}t^8 \\ & + \frac{5}{324}t^7 + \frac{5}{648}t^6 + \frac{1}{324}t^5 + \frac{1}{1296}t^4, \qquad -\infty < t < \infty.\end{aligned}$$

Thus, $\Pr(Z = 14) = 73/648$. In APPL, define `X` as a uniform discrete random variable (predefined

in APPL) with parameter 1/6. The procedure `ConvolutionIID(X, n)` computes the PDF of the convolution of `n` iid random variables `X`. This procedure contains a Maple "for loop" which calls `Convolution` `n` times.

```
> X := DiscreteUniformRV(1 / 6);
> Z := ConvolutionIID(X, 4);
> PDF(Z, 14);
```

The APPL procedure `PDF` computes the probability that Z is 14, which is 73/648. ∎

Examples 8–10 are from the article "Getting Normal Probability Approximations without Using Normal Tables" by Thompson [12]. His paper discusses an alternate approach to approximating probabilities involving sums of discrete random variables using the PDF for the normal distribution. He lets S denote the sum of n independent discrete random variables, and assumes that S assumes consecutive integer values. Letting $\mu = \mathrm{E}(S)$ and $\sigma^2 = \mathrm{Var}(S)$, he argues that for sufficiently large values of n, S is approximately normally distributed. Using the standard continuity correction, he gets

$$\Pr(S = s) = \Pr\left(s - 0.5 < N\left(\mu, \sigma^2\right) < s + 0.5\right).$$

Calculating a midpoint approximation using a single subinterval, the *normal PDF approximation* is obtained, which is

$$\Pr(S = s) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(s-\mu)^2/2\sigma^2}.$$

Instead of settling for approximations of the probabilities, we will show how APPL procedures, including `Convolution`, can retrieve exact solutions, while still giving the same insight into the problem.

Example 8. Suppose that $X_1, X_2, \ldots, X_{20}$ are independent and that $X_i \sim \text{Bernoulli}(p_i = (29 + 2i)/100)$, $i = 1, 2, \ldots, 20$. Let $S = \sum_{i=1}^{20} X_i$. (Here, S denotes the total number of successes obtained in a series of independent trials where the probability of success varies from trial to trial.) Give an exact probability table for S, for $s = 2, 3, \ldots, 10$ [12, p. 53].

Solution. Using Thompson's notation, $\mu = 10$ and $\sigma^2 = 2367/500$, and so $\Pr(S = s) \cong (1/\sqrt{2\pi}\sqrt{2367/500}) e^{-(s-10)^2/(2367/250)}$, $s = 0, 1, \ldots, 20$. Using APPL, we obtain the exact distribution of S with the statements:

```
> p := (29 + 2 * 1) / 100;
> S := BernoulliRV(p);
> for i from 2 to 20 do
> p := (29 + 2 * i) / 100:
> X := BernoulliRV(p):
> S := Convolution(S, X):
> od:
> S;
```

Table 2 contains the exact probabilities found with APPL and the normal PDF approximations for $s = 2, 3, \ldots, 10$. ∎

Example 9

There are 20 girls and 30 boys in Group 1, 25 girls and 25 boys in Group 2, and 10 girls and 10 boys in Group 3. If 10 children are randomly chosen from each group and S denotes the total number of girls chosen, give an exact probability table for S, for $s = 7, 8, \ldots, 21$ [12, p. 53–54].

Solution

Let X_1, X_2, and X_3 be the three independent hypergeometric random variables, and let $S = X_1 + X_2 + X_3$. The mean and variance of S are $\mu = \mathrm{E}[S] = \mathrm{E}[X_1] + \mathrm{E}[X_2] + \mathrm{E}[X_3] = 14$ and σ^2

Table 2. Exact probabilities and normal PDF approximations of $\Pr(S = s)$ for $s = 2, 3, \ldots, 10$.

s	True $\Pr(S = s)$ (APPL)	Approximation of true $\Pr(S = s)$	Normal PDF Approximation
2	$\frac{2046587651449892257889307137290 11}{160000000000000000000000000000000000000}$	0.0001	0.0002
3	$\frac{6705810443818611172719629620439 67}{800000000000000000000000000000000000000}$	0.0008	0.0010
4	$\frac{1230630989005121609042060715648 1161}{320000000000000000000000000000000000000}$	0.0038	0.0041
5	$\frac{1313011896141182060942923449706 2639}{100000000000000000000000000000000000000}$	0.0131	0.0131
6	$\frac{1384554599255601609492241990460 5161}{400000000000000000000000000000000000000}$	0.0346	0.0338
7	$\frac{1442918668426172402399749136761 9439}{200000000000000000000000000000000000000}$	0.0721	0.0709
8	$\frac{1931965285930891530250932459049 30293}{160000000000000000000000000000000000000}$	0.1207	0.1202
9	$\frac{6554941445025712560001435444760 7969}{400000000000000000000000000000000000000}$	0.1639	0.1650
10	$\frac{7253130084768895124176352940113025 41}{400000000000000000000000000000000000000}$	0.1813	0.1834

$= \mathrm{Var}(S) = \mathrm{Var}(X_1) + \mathrm{Var}(X_2) + \mathrm{Var}(X_3) = 101/19$ (since X_1, X_2, and X_3 are independent). Table 3 shows the normal PDF approximation values

$$\Pr(S = s) \cong (1/\sqrt{2\pi}\sqrt{101/19})e^{-(s-14)^2/(202/19)},$$

for $s = 7, 8, \ldots, 20$.

Using the APPL `Convolution` procedure, we can readily produce the PDF of S with the statements

```
> X1 := HypergeometricRV(50, 20, 10);
> X2 := HypergeometricRV(50, 25, 10);
> X3 := HypergeometricRV(20, 10, 10);
> Y := Convolution(X3, Convolution(X1, X2));
```

The exact values for $s = 7, 8, \ldots, 21$ are shown in Table 3. ∎

In the next example, APPL is able to find the convolution of a large number (150) of random variables. While not impossible, computing the actual distribution by hand is tremendously tedious and time-consuming.

EXAMPLE 10. Let $S = X_1 + \cdots + X_{150}$, where the Xs are independent, $\Pr(X_i = -1) = \Pr(X_i = 0) = \Pr(X_i = 1) = 1/3$, $i = 1, 2, \ldots, 150$. Find a normal approximation to $\Pr(S = 5)$ [12, p. 54].

SOLUTION. Since the mean and variance of S are $\mu = 0$ and $\sigma^2 = 100$, using the normal PDF approximation we obtain approximately $1/20(\sqrt{2}e^{-1/8}/\sqrt{\pi}) \cong 0.03521$.

The APPL code:

```
> X := [[1 / 3, 1 / 3, 1 / 3], [-1, 0, 1], ["Discrete", "PDF"]];
> S := ConvolutionIID(X, 150);
> PDF(S, 5);
```

yields the exact PDF for S. The statement `PDF(S, 5)` returns

$$\Pr(S = 5) = \frac{160709987007649212790999852367465829596098558279031212787052332840770}{4567759074507740406477787437675267212178680251724974985372646979033929},$$

which is approximately 0.03518.

Table 3. The exact probabilities and normal PDF approximations for $\Pr(S = s)$ for $s = 7, 8, \ldots, 21$.

s	True $\Pr(S = s)$ (APPL)	Approximation of true $\Pr(S = s)$	Normal PDF Approximation
7	$\frac{4641594894759547665}{3082276280132202064912}$	0.0015	0.0017
8	$\frac{97479371530863990}{17512933409842057187}$	0.0056	0.0059
9	$\frac{12613791756912076515}{770569070033050516228}$	0.0164	0.0165
10	$\frac{74849525260411094591}{1926422675082626290570}$	0.0389	0.0384
11	$\frac{57967137494173367365}{770569070033050516228}$	0.0752	0.0742
12	$\frac{2096975232909133615}{17512933409842057187}$	0.1197	0.1188
13	$\frac{22076335771392253895}{140103467278736457496}$	0.1576	0.1575
14	$\frac{317244095646532855}{1843466674720216546}$	0.1721	0.1730
15	$\frac{9955623438355053449}{63683394217607480680}$	0.1563	0.1575
16	$\frac{217921905682010165}{1843466674720216546}$	0.1182	0.1188
17	$\frac{1894259194489549345}{25473357687042992272}$	0.0744	0.0742
18	$\frac{71588441634588035}{1843466674720216546}$	0.0388	0.0384
19	$\frac{10756216836381565}{641205799902684016}$	0.0168	0.0165
20	$\frac{1208983087163529637}{202781334219223820060}$	0.0060	0.0059
21	$\frac{280730797358534065}{162225067375379056048}$	0.0017	0.0017

Since the mass values of the parent populations are adjacent, $\Pr(S = 5)$ can be computed using a combinatorics approach,

$$\Pr(S=5) = \sum_{\substack{\{(p,q,r)|p+q+r=150,\\ 0\le p\le 150\\ 0\le q\le 150\\ 0\le r\le 150\\ -p+r=5\}}}\sum\sum \binom{150}{p,q,r}\left(\frac{1}{3}\right)^p\left(\frac{1}{3}\right)^q\left(\frac{1}{3}\right)^r$$

or equivalently,

$$\Pr(S=5) = \sum_{p=0}^{72}\binom{150}{p,145-20,5+p}\left(\frac{1}{3}\right)^{150},$$

yielding the same result as the APPL code. The true power of APPL is not demonstrated in this particular example because of the adjacent mass values of the parent populations. The APPL

approach allows for unequal, and more importantly, nonadjacent mass values. Also, more than three mass values can be used in the APPL approach. Although a combinatorics method may still be used on these types of examples, the resulting expressions will not be as easy to evaluate as the expression obtained in the sum above. ∎

6. FURTHER WORK

Computer algebra systems make it feasible to determine distributions of convolutions of random variables, especially those random variables with finite arbitrary supports. The algorithm that was constructed (with heaps) to compute the PDF of the convolution was extended to determine the PDF of the product also. Because of negative and positive support values for random variables X and Y, the product algorithm must split the array A into four quadrants before heaping. The quadrants are split based on X's and Y's negative and nonnegative support values. Figure 12 illustrates the evolution of the algorithm in the most general case when both X and Y have negative and positive support values. The product algorithm works from the corners of quadrants two and four toward the center where the quadrants meet, and then, works outward from the center through quadrants one and three. For the product algorithm, the heap can receive values from two competing quadrants at a time. The function `Product` has been implemented and is a part of APPL. The algorithm for the continuous implementation of `Product` is presented in [13].

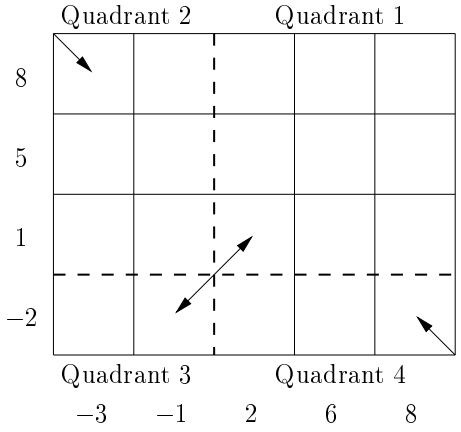

Figure 12. Array A split into four quadrants for the product algorithm.

REFERENCES

1. S. Ross, *A First Course in Probability*, Sixth Edition, Prentice–Hall, Upper Saddle River, NJ, U.S.A., (2002).
2. C.M. Grinstead and J.L. Snell, *Introduction to Probability*, Second Revised Edition, American Mathematical Society, (1997).
3. R.V. Hogg and A.T. Craig, *Mathematical Statistics*, Fifth Edition, Prentice–Hall, Englewood Cliffs, NJ, U.S.A., (1995).
4. E. Parzen, *Modern Probability Theory and Its Applications*, John Wiley & Sons, Inc., New York, (1960).
5. J.A. Woodward and C.G.S. Palmer, On the exact convolution of discrete random variables, *Applied Mathematics and Computation* **83**, 69–77, (1997).
6. M.A. Weiss, *Data Structures and Algorithm Analysis in C++*, Addison–Wesley Publishing Company, Menlo Park, CA, U.S.A., (1994).
7. F.M. Carrano, P. Helman and R. Veroff, *Data Abstraction and Problem Solving with C++: Walls and Mirrors*, Second Edition, Addison–Wesley Longman, Inc., Reading, MA, U.S.A., (1998).
8. D. Nicol, personal communication, (2000).
9. A. Glen, D. Evans and L. Leemis, APPL: A probability programming language, *The American Statistician* **55**, 156–166, (2001).
10. A.A. Sveshnikov, Editor, *Problems in Probability Theory, Mathematical Statistics and Theory of Random Functions*, Dover Publications, Inc., New York, (1968).
11. R.V. Hogg and E.A. Tanis, *Probability and Statistical Inference*, Sixth Edition, Prentice–Hall, Upper Saddle River, NJ, U.S.A., (2000).

12. P. Thompson, Getting normal probability approximations without using normal tables, *The College of Mathematics Journal* **31**, 51–54, (2000).
13. A. Glen, L. Leemis and J. Drew, Computing the distribution of the product of two continuous random variables, *Computational Statistics and Data Analysis* **44**, 451–464, (2004).

APPENDIX 1

DETERMINING CANDIDATE SUMS FOR THE HEAP

THEOREM. *Let $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$ be finite real numbers. Let the $m \times n$ array A be arranged as shown in Figure 13 and have entries $A_{i,j} = y_i + x_j$, $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. Let the set C contain all (i, j) pairs, such that $x_i + y_j \leq c$, where $c > y_1 + x_1$ is a real number. Let P be the path from the northwest corner of A to the southeast corner of A that separates C from C'. The smallest element in C' must occur just to the northeast of a southward followed by an eastward change in direction of the path P.*

y_m	$y_m + x_1$	$y_m + x_2$	$\ldots$	$y_m + x_n$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
y_2	$y_2 + x_1$	$y_2 + x_2$	$\ldots$	$y_2 + x_n$
y_1	$y_1 + x_1$	$y_1 + x_2$	$\ldots$	$y_1 + x_n$
	x_1	x_2		x_n

Figure 13. Array A where $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$.

8	(5)				
5	2	(4)			
1	−2	0	3		
−2	−5	−3	0	(4)	
	−3	−1	2	6	8

Figure 14. Array A corresponds to Example 2. The path P from the northwest corner to the southeast corner of A that delimits the set $C = \{(i, j) \mid y_i + x_j \leq 3\}$ from C' is thickened. The circled entries lie just to the northeast of points in the path where there is a turn from a southward to an eastward direction. These three entries are contained in the cells in C' that hold the smallest entries in C' for $c = 3$.

PROOF. For any cell in C, every cell to the southwest of C is also in C, since $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$. Thus, C must be a finite union of rectangles in A, where each rectangle contains the $(1, 1)$ cell. The next cell to be included in C as c increases is the smallest element in C'. Since C will continue to be a finite union of rectangles when the next cell is added, the smallest element in C' must occur at intersections of the rectangles and the western and southern boundaries of A that occur on P. ∎

As an example, Figure 14 displays the array A for Example 2 with $c = 3$.

APPENDIX 2

ALGORITHM FOR `BRUTEFORCEMETHOD`

The algorithm for the APPL procedure `BruteForceMethod(X, Y)` computes the PDF of the convolution of the PDFs of the two random variables X and Y by the "brute force method" described in Section 2. The support list for the convolution is sorted by a heapsort in the APPL procedure `HeapSort`, which sorts the elements of its first argument, making corresponding swaps to the elements of its second argument. The variables Ω_X and Ω_Y are the supports of the random variables X and Y, respectively.

Procedure `BruteForceMethod(X, Y)` $n \leftarrow |\Omega_X|$
$m \leftarrow |\Omega_Y|$
$s \leftarrow \text{array}[1 \ldots n \cdot m]$
$\text{Probs} \leftarrow \text{array}[1 \ldots n \cdot m]$
For $i \leftarrow 1$ to n
  For $j \leftarrow 1$ to m
    $s_k \leftarrow y_i + x_j$
    $\text{Probs}_k \leftarrow f_Y(y_i) \cdot f_X(x_j)$
return(`HeapSort`(s, Probs))

APPENDIX 3

ALGORITHM FOR `MOVINGHEAPMETHOD`

The algorithm for the APPL procedure `MovingHeapMethod(X, Y)` computes the PDF of the convolution of two random variables X and Y by the "moving heap method" described in Section 2. The additional APPL procedures `PercolateDownHeap`, `RebuildHeap`, and `InsertHeap` are standard heap programs for inserting and restructuring a heap so that it continues to fulfill the properties of a heap.

Procedure `MovingHeapMethod(X, Y)`
$n \leftarrow |\Omega_X|$
$m \leftarrow |\Omega_Y|$
Dimension $s[n \cdot m]$
Dimension $Probs[n \cdot m]$
$s_1 \leftarrow y_1 + x_1$
$Probs_1 \leftarrow f_Y(y_1) \cdot f_X(x_1)$
Dimension $r[m + 1]$
Dimension $c[n + 1]$
$row1col2entry \leftarrow [y_1 + x_2,\ f_Y(y_1) \cdot f_X(x_2)]$
$r_1 \leftarrow 1$
$c_2 \leftarrow 1$
$row2col1entry \leftarrow [y_2 + x_1,\ f_Y(y_2) \cdot f_X(x_1)]$
$r_2 \leftarrow 1$
$c_1 \leftarrow 1$
$r_{n+1} \leftarrow 1$ [Keeps search for new entries inside north border of A]

```
c_{m+1} ← 1                                          [Keeps search for new entries inside east border of A]
H ← [−1 · 10^6, row1col2entry, row2col1entry]
Mimic ← [[0, 0], [1, 2], [2, 1]]                     [Holds the positions of the entries]
PercolateDownHeap(2, 3)                              [Restructures H to fulfill the heap properties]
For q ← 2 to n · m
    RootItem ← H_2
    RootPosition ← Mimic_2
    s_q ← RootItem_1                                 [Root entry placed in sums array s]
    Probs_q ← RootItem_2                             [Root entry's probability placed in probability array Probs]
    a ← RootPosition_1
    b ← RootPosition_2
    r_a ← 0                                          [The root's row becomes inactive]
    c_b ← 0                                          [The root's column becomes inactive]
    size ← |H|
    H_2 ← H_size
    Mimic_2 ← Mimic_size
    H ← [H_1 . . . H_{size−1}]
    Mimic ← [Mimic_1 dots Mimic_{size−1}]
    RebuildHeap(2, size − 1)                         [Restores H as a heap]
    If (r_a = 0) and (c_{b+1} = 0) then              [If the cell just east of
        r_a ← 1                                       the removed entry is inactive,
        c_{b+1} ← 1                                   insert its entry into the heap]
        NewPosition ← [a, b + 1]
        InsertHeap(NewPosition)
    If (r_{a+1} = 0) and (c_b = 0) then              [If the cell just south of
        r_{a+1} ← 1                                   the removed entry is inactive,
        c_b ← 1                                       insert its entry into the heap]
        NewPosition ← [a + 1, b]
        InsertHeap(NewPosition)
return(s, Probs)
```