Comparison of methods for evaluation of the n-fold convolution of an arithmetic distribution

Bjørn Sundt
Department of Mathematics, University of Bergen
Johannes Bruns gate 12, N-5008 Bergen, Norway
Phone: + 47 55 58 28
Fax: + 47 55 58 48 85
E-mail: bsundt@mi.uib.no

David C.M. Dickson
Centre for Actuarial Studies, University of Melbourne
Parkville, VIC 3052, Australia
Phone: + 61 3 9344 4727
Fax: + 61 3 9344 6899
E-mail: ddickson@cupid.ecom.unimelb.edu.au

Abstract: In the present paper we compare De Pril's algorithm for recursive evaluation of the n-fold convolution of an arithmetic distribution with more traditional methods and evaluation by De Pril transforms. The comparison is performed by counting the number of elementary algebraic operations.

Keywords: n-fold convolutions.

1 Introduction

1A. The main purpose of the present paper is to compare De Pril's (1985) algorithm for recursive evaluation of the n-fold convolution of an arithmetic distribution with more traditional evaluation, that is, evaluation directly based on the expression for the convolution. We want to find out how large n should be for it to be more efficient to apply De Pril's method rather than the other method.

1B. Our measure of efficiency is the number of elementary algebraic operations. Like Kuon, Reich, & Reimers (1987) we distinguish between bar operations (summation and subtraction) and dot operations (multiplication and division) as dot operations would normally be more time-consuming than bar operations. These comparisons would give a rough idea of which method is most efficient. However, we stress that for several reasons one should not draw too strong conclusions:

1. By distinguishing between bar and dot operations we have two dimensions so what do we do if one method is more efficient than another with respect to bar operations, but the opposite is the case for dot operations? One solution would be to give bar and dot operations different weights, but how should we choose the weights? To a large extent that would depend on the computer hardware, programming language, and programming style.

2. Some programming languages have strong built-in functions that would be more efficient than programming the individual elementary algebraic operations.

3. Is it really so that a summation is always less time-consuming than a multiplication? Is the summation $a + a$ more efficient than the multiplication $2 \cdot a$? Intuitively one would tend to choose multiplication in such cases. However, in the present paper we shall count multiplications by 2 as bar operations.

4. In addition to algebraic operations, aspects like storage, etc. also ought to be taken into account. Should we always store the value of a product $a \cdot b$ if this product is needed more than once? In our considerations we have done that to reduce the number of multiplications.

5. An algorithm with less algebraic operations could be more complicated to program, and the more complicated a program is, the more time-consuming is the programming and the greater is the danger of errors in

the program. To what extent one should care to minimise the number of algebraic operations, would very much depend on how much the program is to be applied. For a program that is to be used frequently, efficiency becomes more crucial. However, as computers get faster and more powerful, such considerations become less important.

6. All the methods that we present are in principle exact, but rounding errors can occur. Panjer & Wang (1993) discuss numerical stability of recursive methods. In particular they show that De Pril's method is unstable.

1C. Let f be a probability function on the non-negative integers, x a positive integer, and n an integer greater than one. We assume that we need $f^{n*}(y)$ for $y = 0, 1, \ldots, x$.

We do not make any simplifying assumptions, e.g. that $f(y) = 0$ for $y = 0$ or all y greater than some finite value. However, in De Pril's method we divide by $f(0)$, and hence we there have to assume that $f(0) > 0$.

1D. We shall first consider traditional evaluation. This is based on repeated application of

$$f^{(p+q)*}(y) = \left(f^{p*} * f^{q*}\right)(y) = \sum_{z=0}^{y} f^{p*}(z) f^{q*}(y-z). \tag{1.1}$$

Thus, a crucial element will be the convolution of two probability functions f and g, that is,

$$(f * g)(y) = \sum_{z=0}^{y} f(z) g(y-z). \tag{1.2}$$

The number of algebraic operations needed for evaluation of this formula will be studied in Section 2.

In the special case when $g = f$, by brute force evaluation of (1.2) we would perform many of the operations twice. Thus, we can reduce the number of operations considerably. This is the topic of Section 3.

In Section 4 we discuss evaluation of f^{n*} by repeated application of (1.1). An alternative approach for evaluation of f^{n*} is De Pril's (1985) recursive procedure, which will be analysed in Section 5. In Section 6 we consider evaluation by De Pril transforms as discussed in Sundt (1995).

In Section 7 we compare the three approaches. It turns out that De Pril's method is more efficient than the De Pril transform method, and that for most values of n De Pril's method is more efficient than traditional evaluation.

Finally, in Section 8 we briefly consider the situation where we want to evaluate not only f^{n*}, but f^{j*} for all $j \leq n$. In this case traditional evaluation is preferable whereas the De Pril transform method could be preferable in some cases where we want to evaluate f^{j*} for r non-consecutive values of j.

1E. If x is a real number, then, by $[x]$ we shall mean the largest integer less than or equal to x.

We make the convention that $\sum_{i=a}^{b} = 0$ when $b < a$.

2 The convolution of two distributions

For evaluation of $(f * g)(y)$ by (1.2) we need $y+1$ dot operations and y bar operations, that is, for $y = 0, 1, \ldots, x$ we need

$$b(x) = \frac{x(x+1)}{2} \tag{2.1}$$

bar operations, and

$$d(x) = \frac{(x+1)(x+2)}{2} \tag{2.2}$$

dot operations.

3 Simplification for the two-fold convolution

With $g = f$ in (1.2) we obtain

$$f^{2*}(y) = \sum_{z=0}^{y} f(z) f(y-z). \tag{3.1}$$

In particular, for $y = 0$ we obtain

$$f^{2*}(0) = f(0) f(0),$$

from which we see that to evaluate $f^{2*}(0)$ we need one dot operation.

When y is positive, many of the products in (3.1) are equal, and, thus, it seems that we can reduce the number of operations considerably in this special case of (1.2). On the other hand, programming may become more messy, in particular as we have to consider even and odd y's separately.

Let u be a positive integer. We have

$$f^{2*}(2u-1) = \sum_{z=0}^{2u-1} f(z) f(2u-1-z) =$$
$$\sum_{z=0}^{u-1} f(z) f(2u-1-z) + \sum_{z=u}^{2u-1} f(z) f(2u-1-z),$$

and as the two sums in the last expression are equal, we obtain

$$f^{2*}(2u-1) = 2\sum_{z=0}^{u-1} f(z)\, f(2u-1-z)\,. \tag{3.2}$$

Analogously

$$f^{2*}(2u) = 2\sum_{z=0}^{u-1} f(z)\, f(2u-z) + f(u)^2\,. \tag{3.3}$$

Evaluation of $f^{2*}(2u-1)$ by (3.2) involves u bar operations and u dot operations, and evaluation of $f^{2*}(2u)$ by (3.3) involves $u+1$ bar operations and $u+1$ dot operations (recall that we count multiplication by 2 as a bar operation). Thus, evaluation of $f^{2*}(2u-1)$ and $f^{2*}(2u)$ involves $2u+1$ bar operations and $2u+1$ dot operations.

We let $b_2(x)$ and $d_2(x)$ denote the number of bar and dot operations respectively needed to evaluate $f^{2*}(0), f^{2*}(1), \ldots, f^{2*}(x)$ with our present methodology. We see that

$$d_2(x) = b_2(x) + 1.$$

Let v be a positive integer. Summation over u gives that with $x = 2v$ we obtain

$$b_2(x) = \sum_{u=1}^{v} (2u+1) = v(v+2) = \frac{x(x+4)}{4} \tag{3.4}$$

$$d_2(x) = x\left(\frac{x}{4}+1\right) + 1 = \frac{x^2+4x+4}{4}. \tag{3.5}$$

For $x = 2v-1$ it seems most convenient to first take the number of operations as if x were equal to $2v$ and then subtract the number of operations to evaluate $f^{2*}(2v)$. We obtain

$$b_2(x) = v(v+2) - (v+1) = \frac{x^2+4x-1}{4} \tag{3.6}$$

$$d_2(x) = \frac{x^2+4x-1}{4} + 1 = \frac{x^2+4x+3}{4}. \tag{3.7}$$

We have

$$\lim_{v\uparrow\infty} \frac{d_2(2v+1)}{d_2(2v)} = 1,$$

that is, not unexpectedly, the numbers of operations in the odd and even cases are asymptotically equal. We also find

$$\lim_{x\uparrow\infty} \frac{d_2(x)}{d(x)} = \lim_{x\uparrow\infty} \frac{b_2(x)}{b(x)} = \frac{1}{2},$$

that is, asymptotically, evaluation by (3.2) and (3.3) requires half the number of operations required for evaluation by (3.1).

4 Extension to the n-fold convolution

As mentioned in subsection 1D, we can evaluate f^{n*} by repeated application of (1.1). The question is what would be the most efficient way to do this? In Section 3 we saw that evaluation of $(f * g)(y)$ by (1.2) for $y = 0, 1, \ldots, x$ requires asymptotically twice as many algebraic operations for $g \neq f$ as for $g = f$. Thus, it seems that in addition to keeping the number of applications of (1.1) as low as possible we want as many as possible of them with $p = q$. Let us count each usage of (1.1) as 2 when $p \neq q$ and 1 when $p = q$.

The least efficient we could do, would be to use (1.1) with $p = i$ and $q = 1$ for $i = 1, 2, \ldots, n-1$. That would give a count of

$$w(n) = 2(n-1) - 1 = 2n - 3;$$

the deduction of 1 being for the evaluation of f^{2*}.

Let us now describe what we believe to be the optimal strategy. We introduce the binary representation

$$n = 2^{k(n)} + \sum_{i=0}^{k(n)-1} 2^i b_{ni}$$

of n, where $k(n)$ is a positive integer and $b_{ni} \in \{0, 1\}$ for $i = 0, 1, \ldots, k(n)-1$. We first evaluate $f^{2^i *}$ by (1.1) with $p = q = 2^{i-1}$ for $i = 1, 2, \ldots, k(n)$; each of these $k(n)$ applications has count 1. Finally we find f^{n*} by

$$f^{n*} = f^{2^{k(n)}*} * \left(\mathop{*}_{\{i : b_{ni} = 1\}} f^{2^i *} \right),$$

which is evaluated by $\sum_{i=0}^{k(n)-1} b_{ni}$ applications of (1.1), each of which has count 2. Thus, we apply (1.1)

$$a(n) = k(n) + \sum_{i=0}^{k(n)-1} b_{ni} = \sum_{i=0}^{k(n)-1} (b_{ni} + 1)$$

times, and that gives a total count of

$$c(n) = k(n) + 2 \sum_{i=0}^{k(n)-1} b_{ni} = \sum_{i=0}^{k(n)-1} (2b_{ni} + 1) = 2a(n) - k(n). \qquad (4.1)$$

We believe that $c(n)$ is the lowest possible number of counts for evaluation of f^{n*} by repeated application of (1.1).

We trivially have

$$c(n) = a(n) = k(n) \qquad (4.2)$$

n	$k(n)$	$a(n)$	$c(n)$	$w(n)$
2	1	1	1	1
3	1	2	3	3
4	2	2	2	5
5	2	3	4	7
6	2	3	4	9
7	2	4	6	11
8	3	3	3	13
9	3	4	5	15
10	3	4	5	17
11	3	5	7	19
12	3	4	5	21
13	3	5	7	23
14	3	5	7	25
15	3	6	9	27
16	4	4	4	29

Table 4.1: Counts for the n-fold convolution.

when n is a power of two, and

$$c(n) > a(n) > k(n) \tag{4.3}$$

when this is not the case.

When applying the present strategy to evaluate $f^{n*}(0), f^{n*}(1), \ldots, f^{n*}(x)$, we need

$$b_n(x) = k(n)\, b_2(x) + (a(n) - k(n))\, b(x) \tag{4.4}$$

bar operations and

$$d_n(x) = k(n)\, d_2(x) + (a(n) - k(n))\, d(x) \tag{4.5}$$

dot operations.

In Table 4.1 we display $k(n)$, $a(n)$, $c(n)$, $w(n)$ for $n = 1, 2, \ldots, 16$.

5 De Pril's recursion

De Pril (1985) presented the recursion

$$f^{n*}(y) = \begin{cases} \dfrac{1}{f(0)} \displaystyle\sum_{z=1}^{y} \left(\frac{n+1}{y} z - 1 \right) f(z)\, f^{n*}(y-z) & (y = 1, 2, \ldots) \\ f(0)^n. & (y = 0) \end{cases} \tag{5.1}$$

Most programming languages have a power function or routines for exponentials and logarithms that could be applied for evaluation of the initial value

$$f^{n*}(0) = f(0)^n. \tag{5.2}$$

However, such procedures would introduce a new dimension as it is uncertain how they compare to dot and bar operations. When restricting to dot and bar operations, we can find $f^{n*}(0)$ by $n-1$ multiplications by $a(n)$ by optimising like we did in Section 4.

For evaluation of $f(y)$ for $y > 0$, we rewrite the expression in (5.1) as

$$f^{n*}(y) = \frac{1}{s(y)} \sum_{z=1}^{y} h(z,y) f^{n*}(y-z) \tag{5.3}$$

with

$$s(y) = yf(0); \qquad h(z,y) = ((n+1)z - y) f(z),$$

which can be evaluated recursively by

$$s(y) = s(y-1) + f(0) \qquad (y = 2, 3, \ldots) \tag{5.4}$$

$$s(1) = f(0)$$

$$h(z,y) = h(z,y-1) - f(z) \qquad (z = 1, 2, \ldots, y-1) \tag{5.5}$$

$$h(y,y) = nyf(y).$$

Let us first consider the case $y = 1$. To evaluate $h(1,1)$ we need one dot operation, and for $s(1)$ we need no algebraic operations. To evaluate $f^{n*}(1)$ by (5.3) we need two dot operations. Thus, totally we need three dot operations.

Let us now consider $y > 1$. We need two dot operations to find $h(y,y)$, and to find $h(z,y)$ by (5.5) we need one bar operation for each $z = 1, 2, \ldots, y-1$. To evaluate $s(y)$ by (5.4) we need one bar operation. Finally we need $y-1$ bar operations and $y+1$ dot operations to evaluate $f^{n*}(y)$ by (5.3). Thus, totally we need $2y-1$ bar operations and $y+3$ dot operations.

Summing up the number of operations that we have found, we obtain that for evaluation of $f^{n*}(y)$ for $y = 0, 1, \ldots, x$ we need

$$b_r(x) = \sum_{y=2}^{x} (2y-1) = x^2 - 1 \tag{5.6}$$

bar operations and

$$d_r(x) = a(n) + 3 + \sum_{y=2}^{x} (y+3) = \frac{x}{2}(x+7) + a(n) - 1 \tag{5.7}$$

dot operations.

6 Evaluation by De Pril transforms

Sundt (1995) defined the De Pril transform φ_f of f by

$$\varphi_f(y) = \frac{1}{f(0)} \left(y f(y) - \sum_{z=1}^{y-1} \varphi_f(z) f(y-z) \right). \qquad (y = 0, 1, 2, \ldots) \quad (6.1)$$

The De Pril transform determines f uniquely. By solving (6.1) for $f(y)$ we obtain

$$f(y) = \frac{1}{y} \sum_{z=1}^{y} \varphi_f(z) f(y-z). \qquad (y = 1, 2, \ldots) \quad (6.2)$$

Furthermore, Sundt (1995) showed that

$$\varphi_{f^{n*}}(y) = n \varphi_f(y). \qquad (y = 1, 2, \ldots) \quad (6.3)$$

Thus, we can evaluate f^{n*} by first evaluating φ_f by (6.1), then finding $\varphi_{f^{n*}}$ by (6.3), and finally evaluating f^{n*} by (6.2), obtaining the starting value $f^{n*}(0)$ by (5.2).

As argued in Section 5 we need $a(n)$ dot operations and no bar operations to evaluate $f^{n*}(0)$.

Let us now consider $y > 0$. To evaluate $\varphi_f(y)$ by (6.1) we need $y-1$ bar operations and $y+1$ dot operations. To evaluate $\varphi_{f^{n*}}(y)$ by (6.3) we need one dot operation, and to evaluate $f^{n*}(y)$ by (6.2) we need $y-1$ bar operations and $y+1$ dot operations. Thus, we totally need $2y-2$ bar operations and $2y+3$ dot operations to evaluate $f^{n*}(y)$, and by summation over y and adding the operations for evaluation of $f^{n*}(0)$ we obtain that to evaluate $f^{n*}(y)$ for $y = 0, 1, \ldots, x$ we need

$$b_p(x) = \sum_{y=1}^{x} (2y-2) = x(x-1)$$

bar operations and

$$d_p(x) = a(n) + \sum_{y=1}^{x} (2y+3) = x(x+4) + a(n)$$

dot operations.

7 Comparison of the methods

7A. We easily see that $d_p(x) - d_r(x)$ is always positive. On the other hand, $b_p(x) - b_r(x)$ is negative for all $x > 1$, that is, dot and bar operations give opposite conclusions. Let us therefore compare the total number of algebraic operations required for the two methods. We have

$$b_p(x) + d_p(x) - (b_r(x) + d_r(x)) = \frac{x}{2}(x-1) + 2 > 0,$$

that is, totally the De Pril transform method requires more algebraic operations than De Pril's method. Furthermore, as $d_p(x) - d_r(x) > 0$, and our reason for distinguishing between bar and dot operations was that the latter would be more time consuming, we conclude that De Pril's method is more efficient than the De Pril transform method. Thus, we can concentrate on comparing De Pril's method and traditional evaluation. However, we point out that for large n the method of Section 6 will be more efficient than traditional evaluation.

7B. To compare the number of operations needed in De Pril's method and the method of Section 4 we introduce the differences

$$b_{n\Delta}(x) = b_n(x) - b_r(x); \qquad d_{n\Delta}(x) = d_n(x) - d_r(x).$$

By application of (4.4), (3.4), (3.6), (2.1), (4.1), and (5.6) we obtain

$$b_{n\Delta}(x) = \begin{cases} \frac{1}{4}\left((c(n)-4)x^2 + 2(a(n)+k(n))x + 4\right) & (x \text{ even}) \\ \frac{1}{4}\left((c(n)-4)x^2 + 2(a(n)+k(n))x + 4 - k(n)\right) & (x \text{ odd}) \end{cases} \tag{7.1}$$

and by (4.5), (3.5), (3.7), (2.2), (4.1), and (5.7)

$$d_{n\Delta}(x) = \begin{cases} \frac{1}{4}\left((c(n)-2)x^2 + (3c(n)+k(n)-14)x + 4\right) & (x \text{ even}) \\ \frac{1}{4}\left((c(n)-2)x^2 + (3c(n)+k(n)-14)x + 4 - k(n)\right). & (x \text{ odd}) \end{cases} \tag{7.2}$$

From (7.1), (4.2), (4.3), and Table 4.1 we see that for all $n \geq 5$ except for $n = 8$ we have $b_{n\Delta}(x) \geq 0$ for all $x > 0$. For $n = 8$ and $n < 5$ we have $b_{n\Delta}(x) < 0$ except for some small values of x. Thus, we conclude that with respect to bar operations traditional evaluation is preferable when $n = 8$ and $n < 5$ whereas De Pril's method is at least as good for all other values of n.

Let us now turn to dot operations. For all n except 2 and 4 we have $d_{n\Delta}(x) \geq 0$ for all $x > 0$. For $n = 2$ and $n = 4$ we have $d_{n\Delta}(x) < 0$ except for some small values of x. Thus, with respect to dot operations we conclude that traditional evaluation is preferable when $n = 2$ and $n = 4$ whereas De Pril's method is at least as good for all other values of n.

We see that the conclusions with respect to bar and dot operations are consistent except for $n = 3$ and $n = 8$. In both these cases we have $b_{n\Delta}(x) + d_{n\Delta}(x) > 0$ for all x, and by similar reasoning as in subsection 7A we conclude that in both these cases De Pril's method is more efficient than traditional evaluation, that is, we prefer De Pril's method for all values of n except 2 and 4.

8 Evaluation of $f^{2*}, f^{3*}, \ldots, f^{n*}$

Until now we have discussed evaluation of $f^{n*}(0), f^{n*}(1), \ldots, f^{n*}(x)$, and our conclusion was that for most values of n, De Pril's method is preferable to traditional evaluation with regard to the number of algebraic operations. If we also need $f^{j*}(0), f^{j*}(1) \ldots, f^{j*}(x)$ for $j = 2, 3, \ldots, n-1$, the picture changes. Whereas De Pril's method is a recursion in y for $f^{n*}(y)$, in traditional evaluation we also evaluate f^{j*} for some values of $j < n$.

The most efficient way of traditional evaluation of $f^{2*}, f^{3*}, \ldots, f^{n*}$ seems to be to evaluate f^{j*} by the method of Section 2 with $g = f^{(j-1)*}$ when j is odd, and when j is even, as the two-fold convolution of $f^{\frac{j}{2}*}$ by the method of Section 3.

With De Pril's method we will have to perform the recursion (5.3) for each value of j. The only places where it seems possible to obtain some gain, are in evaluation of $f^{n*}(0)$, $s(1)$, and $h(y, y)$.

Without going into further detail we conclude that in this situation, traditional evaluation is preferable to De Pril's method.

Traditional evaluation will also be more efficient than the De Pril transform method. However, the latter method may be more efficient in some cases where we want to evaluate $f^{j*}(0), f^{j*}(1), \ldots, f^{j*}(x)$ for r non-consecutive values of j.

Acknowledgement

The present research was carried out while the first author stayed as GIO Visiting Professor at the Centre for Actuarial Studies, University of Melbourne. We are grateful to W.S. Jewell for useful suggestions in connection with Sections 3 and 5.

References

De Pril, N. (1985). Recursions for convolutions of arithmetic distributions. *ASTIN Bulletin* **15**, 135-139.

Kuon, S., Reich, A. & Reimers, S. (1987). Panjer vs Kornya vs De Pril: A comparison. *ASTIN Bulletin* **17**, 183-191. Letter to the editors. *ASTIN Bulletin* **18**, 113-114.

Panjer, H.H. & Wang, S. (1993). On the stability of recursive formulas. *ASTIN Bulletin* **23**, 227-258.

Sundt, B. (1995). On some properties of De Pril transforms of counting distributions. *ASTIN Bulletin* **25**, 19–31.