THE COMPUTATION OF AGGREGATE LOSS DISTRIBUTIONS

JOHN P. ROBERTSON

Abstract

Paul R. Halmos recently hailed the fast Fourier transform as one of the 22 most significant developments in mathematics in the last 75 years. This paper provides an application of this tool to the computation of aggregate loss distributions from arbitrary frequency and severity distributions. All necessary mathematics is developed in the paper, complete algorithms are given, and examples are provided. Sufficient details are given to allow implementation in any computer language, and sample APL computer language routines are given. The final section includes a discussion of excess loss distributions where computation is not limited to the fast Fourier transform based algorithm.

I thank Walter R. Stromquist, Joseph L. ("Joth") Tupper, Gary G. Venter, and the Committee on Review of Papers for their help with and suggestions regarding this paper.

1. INTRODUCTION

According to Halmos [1], the fast Fourier transform is one of this century's most significant mathematical developments. This paper presents an algorithm for computing aggregate loss distributions using this device. The algorithm assumes that one knows the claim count distribution, T (the probability distribution of the number of claims that will occur), and the severity distribution of a single claim, $S = S_1 = S_2 = ...$ (the distribution of the amount of a single claim). The algorithm computes the aggregate loss distribution,

$$AGG = S_1 + S_2 + ... + S_T$$

(the distribution of the total amount of claims). The algorithm applies to arbitrary frequency and severity distributions.

As an example, claim counts might be expected to follow a Poisson distribution with mean 10, and severity might be expected to follow some distribution with mean $10,000. This implies that the mean of the aggregate distribution is $100,000 (10 times $10,000). In any given year, the total amount of claims might vary from $100,000 because the actual number of claims might differ from 10, and because individual claims will vary from the $10,000 mean. The aggregate distribution expresses the probabilities of the possible total amounts of claims in the same way the severity distribution expresses the probabilities of the amounts of a single claim.

The algorithm given here is less of a "black box" than some other algorithms presented in these *Proceedings* in the following way. The algorithm, as a matter of course, computes the distribution for the sum of n claims, where n is any number of claims with nonzero probability in the claim count distribution. While the computer routines presented herein do not save these distributions, only trivial programming changes are needed to capture and save these distributions for later use. Capturing these distributions can be useful when the resulting aggregate distribution has unexpected properties and one wants to check that it is being correctly computed, or for other reasons.

The method presented here should be considered to be approximate. Technically, it is an exact method, but it is generally necessary to use an approximation of the severity distribution as input, and this makes the output approximate. The running time for the algorithm is roughly proportional to the number of claims expected. For small numbers of claims, this method seems to be faster than other methods (e.g., Heckman and Meyers [2]), but this advantage disappears as the number of claims grows. The algorithm presented here explicitly computes the entire aggregate distribution up to a specified limit, making it easy to derive any statistics of the aggregate distribution.

A quick overview of the algorithm is as follows. Everything in this summary will be described fully below, as it is not possible to give brief

rigorous definitions of all the concepts used. The severity distribution will be given a discrete representation; that is, the severity distribution will be represented by a vector. The n-fold (discrete) *convolutions* of this vector with itself are computed. The result of these convolutions is very nearly the vector representation of the n-fold sum of the severity distribution with itself. The precise representation of an n-fold sum will be obtained by computing the convolution of this result with another vector (described later) to "spread out" the result a bit more. This representation of the density function for the n-fold sum of claims is multiplied by the probability of there being exactly n claims, and these products are added to get the vector representation of the aggregate distribution.

The *discrete Fourier transform* is used to compute the convolutions, and the *fast Fourier transform* is used for rapid computation of the discrete Fourier transform. Convolutions, discrete Fourier transforms, and fast Fourier transforms are defined and discussed in Section 2. The purpose of this discussion is to introduce these items and to give examples so the main structure of the algorithm will be clear. The technical details are in the appendices. Additionally, Section 2 discusses the vector used to "spread out" the n-fold convolutions of the vector representing the severity distribution. Two tactics used to speed the overall computations are also covered.

Section 3 of the paper walks through the full algorithm. Section 4 gives examples and discusses use of the algorithm. Sufficient details are given throughout the paper that it should be possible to implement the algorithm in any computer language. As an example, various appendices show routines implementing the algorithm in the APL computer language.

2. CONVOLUTIONS AND THE FAST FOURIER TRANSFORM

Convolutions

The distribution of the sum of two random variables is given by the *convolution* of their respective distributions. Heckman and Meyers [2, p. 32] discuss convolutions for the case of continuous random variables. In this case, if X_1 and X_2 are independent continuous random variables with

density functions f and g, then the density of the sum of these two variables, i.e., the density of the random variable $X_1 + X_2$, is given by the convolution of f and g, $f*g$, defined as:

$$(f*g)(x) = \int_0^x f(t)\, g(x-t)\, dt .$$

For the algorithm presented here, the probability distributions for the severity of a single claim and for the sum of n claims will be given certain discrete representations; that is, they will be represented by certain vectors. It will be necessary to compute the convolutions of these vectors. The definition of the convolution of vectors is similar to the above definition of the convolution of continuous functions. Let $\boldsymbol{U} = (u_0, u_1, \ldots, u_{n-1})$ and $\boldsymbol{V} = (v_0, v_1, \ldots, v_{n-1})$ be two vectors of the same length, n. Their discrete convolution, $\boldsymbol{W} = \boldsymbol{U}*\boldsymbol{V}$, is a vector of length n defined by:

$$w_i = \sum_{j=0}^{n-1} u_j\, v_{i-j} ,$$

where $0 \le i \le n-1$ and the indices of the terms v_{i-j} are taken modulo n. For example, if $\boldsymbol{U} = (1, 2, 3)$ and $\boldsymbol{V} = (4, 5, 6)$, then

$$\begin{aligned} \boldsymbol{U}*\boldsymbol{V} &= (1\times4 + 2\times6 + 3\times5,\ 1\times5 + 2\times4 + 3\times6,\ 1\times6 + 2\times5 + 3\times4) \\ &= (31, 31, 28). \end{aligned}$$

This definition of convolution is not exactly what is needed here. The *no-wrap convolution* of $\boldsymbol{U}$ with $\boldsymbol{V}$ is defined to have the following components:

$$w_i = \sum_{j=0}^{i} u_j\, v_{i-j} .$$

That is:

$$w_0 = u_0\, v_0,$$

$$w_1 = u_0\, v_1 + u_1\, v_0 ,$$

$$w_2 = u_0 v_2 + u_1 v_1 + u_2 v_0 ,$$

.

.

.

$$w_{n-1} = u_0 v_{n-1} + u_1 v_{n-2} + \ldots + u_{n-1} v_0 .$$

The no-wrap convolution of (1, 2, 3) with (4, 5, 6) is (1×4, 1×5 + 2×4, 1×6 + 2×5 + 3×4) or (4, 13, 28).

The no-wrap convolution can be visualized, as below, by taking one vector, reversing it, and placing it so that its first element is directly below the first element of the other vector. Then successively shift the vectors together, multiply elements in the same column, and add the products. Repeat this until the vectors are completely aligned.

No-wrap Convolution

		1	2	3			1	2	3		1	2	3
6	5	4			6	5	4			6	5	4	
		1×4					1×5 + 2×4				1×6 + 2×5 + 3×4		

In contrast, for regular convolutions, the bottom vector is wrapped around as shown below:

Regular Convolution

1	2	3	1	2	3	1	2	3
4	6	5	5	4	6	6	5	4
1×4 + 2×6 + 3×5			1×5 + 2×4 + 3×6			1×6 + 2×5 + 3×4		

The analogy of the definition of no-wrap convolution for discrete vectors to the definition of convolution in the continuous case should be clear.

One can obtain the no-wrap convolution of two vectors from a routine that computes (regular) convolutions by padding each of the two vectors to the right with enough zeroes to double the length of each vector, performing the regular convolution with these longer vectors, and then

taking just the left half of the result. For example, the first three components of the convolution of $\boldsymbol{U} = (u_0, u_1, u_2, 0, 0, 0)$ with $\boldsymbol{V} = (v_0, v_1, v_2, 0, 0, 0)$ are:

$$\begin{aligned} w_0 &= u_0 v_0 + u_1 \times 0 + u_2 \times 0 + 0 \times 0 + 0 \times v_2 + 0 \times v_1 \\ &= u_0 v_0 ; \\ w_1 &= u_0 v_1 + u_1 v_0 + u_2 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times v_2 \\ &= u_0 v_1 + u_1 v_0 ; \\ w_2 &= u_0 v_2 + u_1 v_1 + u_2 v_0 + 0 \times 0 + 0 \times 0 + 0 \times 0 \\ &= u_0 v_2 + u_1 v_1 + u_2 v_0 . \end{aligned}$$

The first definition of convolution will always be used (unless otherwise noted), but generally zeroes will be added to the vectors being convolved so as to achieve a no-wrap convolution.

Observe that the definition of convolution is valid when the vector elements are complex numbers.

A note on notation is needed. The vector $\boldsymbol{U}$ has components u_0, u_1, u_2, etc., sometimes denoted $\boldsymbol{U}[0]$, $\boldsymbol{U}[1]$, $\boldsymbol{U}[2]$, etc. In particular, the indices of vector elements start at zero.

The Discrete Fourier Transform

Complex numbers and complex roots of unity are used extensively in what follows. The *primitive n^{th} roots of unity* are

$$\cos(2\pi a/n) + i \sin(2\pi a/n),$$

where a and n are relatively prime and i is $\sqrt{-1}$. The properties of complex numbers needed here are reviewed in Appendix A, and are also given in Baase [3, p. 279] and Aho, Hopcraft, and Ullman [4, p. 252].

Given a complex (or real) vector $\boldsymbol{U}$, the discrete Fourier transform of $\boldsymbol{U}$ is a complex vector of the same length. As in Baase [3, p. 269], for $n \geq 1$, let ω be a primitive n^{th} root of unity, and let $\boldsymbol{F}$ be the $n \times n$ matrix with entries $f_{i,j} = \omega^{ij}$ where $0 \leq i, j \leq n - 1$. The discrete Fourier transform

(DFT) of the n-vector $\boldsymbol{U} = (u_0, u_1, \ldots, u_{n-1})$ is the product $\boldsymbol{FU}$ (with $\boldsymbol{U}$ treated as a column vector). This is a vector of length n with components:

$$\omega^0 u_0 + \omega^0 u_1 + \ldots + \omega^0 u_{n-2} + \omega^0 u_{n-1},$$
$$\omega^0 u_0 + \omega^1 u_1 + \ldots + \omega^{n-2} u_{n-2} + \omega^{n-1} u_{n-1},$$
$$\vdots$$
$$\omega^0 u_0 + \omega^i u_1 + \ldots + \omega^{i(n-2)} u_{n-2} + \omega^{i(n-1)} u_{n-1},$$
$$\vdots$$
$$\omega^0 u_0 + \omega^{n-1} u_1 + \ldots + \omega^{(n-1)(n-2)} u_{n-2} + \omega^{(n-1)(n-1)} u_{n-1}.$$

(Note that the DFT of $\boldsymbol{U}$ depends on the ω chosen.)

Let $\boldsymbol{FU}$ be the DFT of $\boldsymbol{U}$. Given $\boldsymbol{FU}$, $\boldsymbol{U}$ is recovered (i.e., the inverse DFT is applied to $\boldsymbol{FU}$) as easily as $\boldsymbol{FU}$ is computed from $\boldsymbol{U}$. To obtain $\boldsymbol{U}$ from $\boldsymbol{FU}$, compute the DFT of $\boldsymbol{FU}$, divide each resulting term by n, and reverse the order of the last $n-1$ elements of this result.

Using the DFT to Compute Convolutions

The DFT helps compute convolutions because

$$\text{DFT}(\boldsymbol{U}*\boldsymbol{V}) = \text{DFT}(\boldsymbol{U}) \times \text{DFT}(\boldsymbol{V}), \text{ or}$$
$$\boldsymbol{U}*\boldsymbol{V} = \text{INVDFT}(\text{DFT}(\boldsymbol{U}) \times \text{DFT}(\boldsymbol{V})),$$

where INVDFT is the inverse DFT.

Thus, to compute the convolution of two vectors, one can compute the DFT of each vector, multiply the DFTs together pointwise, and compute the inverse DFT. This is known as the convolution theorem, proofs of which are given in Baase [3, p. 278] and Aho, Hopcraft, and Ullman [4, p. 255].

For example, let $\boldsymbol{U}$ be (1, 2, 3), let $\boldsymbol{V}$ be (4, 5, 6), and let $\omega = -0.5 + 0.866i$, a primitive third root of unity. Then $\boldsymbol{FU}$ is $(6, -1.5 - 0.866i, -1.5 + 0.866i)$, $\boldsymbol{FV}$ is $(15, -1.5 - 0.866i, -1.5 + 0.866i)$, and the pointwise product, $\boldsymbol{FU} \times \boldsymbol{FV}$, is $(90, 1.5 + 2.598i, 1.5 - 2.598i)$.

To compute the inverse DFT of this last vector, first apply the (forward) DFT to obtain (93, 84, 93). Then divide each element by 3 and reverse the order of the last two elements, giving (31, 31, 28). This matches the previous computation.

A more thorough example showing the convolution of two vectors representing severity distributions is given in Appendix B. This appendix also discusses the inverse DFT.

The Fast Fourier Transform

The *fast Fourier transform* (FFT) is a particularly fast method for computing the DFT (and the inverse DFT) for long vectors. Appendix C gives Baase's [3, p. 273] version of the complete fast Fourier transform and inverse FFT. Appendix D gives an APL implementation of these algorithms.

Why use such a complicated method to compute convolutions (i.e., using the convolution theorem and FFTs)? This method is used because, for long vectors, it's much faster than more straightforward methods (such as computing directly from the definition of convolution). Knuth [5, Vol. 2, p. 651] discusses the number of calculations needed to compute the fast Fourier transform. Using the FFT to convolve vectors of length 1,024 (2^{10}) gives a gain in speed by a factor of about 60 compared to the "naive method" of convolution. The time needed by the naive method for computing convolutions of vectors of length n is proportional to n^2. More precisely,[1] it is $O(n^2)$. The time needed by the method using the convolution theorem and the fast Fourier transform is proportional to $n\ln(n)(O(n\ln(n)))$, where n is the length of the vectors convolved. For large n, $n\ln(n)$ increases more slowly than n^2.

In the literature there are several different definitions of the DFT and FFT that accomplish the same goals but differ in certain details. Be careful before trying to use the algorithms presented here in conjunction with algorithms that appear in other sources or are available in libraries of

[1] That the number of computations is $O(f(n))$ means that there is a constant c such that the number of computations is less than $c \times f(n)$ for all n greater than some n_0. See Knuth [5, Vol. 1, p. 104].

computer routines. For example, using the forward FFT from another source with the inverse FFT given here could produce erroneous results. Other routines for the DFT and FFT can certainly be used, as long as all the routines used are consistent among themselves.

In addition to Baase [3, p. 268], the fast Fourier transform is discussed from several different viewpoints in Knuth [5, Vol. 2], Press, Flannery, Teukolsky, and Vetterling [6, p. 390], Preparata [7, p. 207], and Aho, Hopcraft, and Ullman [4, p. 257]. Each of these discusses the theory behind the FFT, thereby explaining what the FFT is doing and showing why the FFT is so efficient. Chiu [8] gives an introduction to the fast Fourier transform motivated by the problem of exact multiplication of large integers. A detailed implementation of the fast Fourier transform suitable for Fortran and similar languages is given in Monro [9, p. 153]. Convolution is discussed in Hogg and Klugman [10, p. 42], Feller [11, p. 6], and many other statistics books.

Other methods that use the fast Fourier transform to compute aggregate loss distributions are given by I. J. Good in Borch [12, p. 298] and by Bertram [13, p. 175]. The method presented in the latter is summarized in Bühlmann [14, p. 116].

3. THE ALGORITHM

The Basic Algorithm

The *collective risk model* will be used to model the claims process. That is, the aggregate loss distribution is the distribution:

$$AGG = S_1 + S_2 + \ldots + S_T ,$$

where T is a random variable for the number of claims and each S_i is a random variable for severity. It is assumed that the S_i are identically distributed and are pairwise independent and that the S_i are independent of T. This definition of aggregate loss distribution is the same as that given by Algorithm 3.2 in Heckman and Meyers [2, p. 30]. This model is discussed in Bühlmann [15, p. 54], Beard, Pentikäinen, and Pesonen [16, p. 50],

Patrik and John [17, p. 412], and Mayerson, Jones, and Bowers [18, p. 177].

There are three inputs to the algorithm. The first, denoted M, is the smallest number of claims that has nonzero probability in the claim count distribution. The second, $\boldsymbol{P}$, is a vector giving the probability density function of the claim count distribution. $\boldsymbol{P}(i)$ is the probability that there are exactly $i + M$ claims for $i \geq 0$.

The third input, $\boldsymbol{S}$, is a vector representing the severity distribution as a piecewise uniform distribution. Due to technical considerations involving the fast Fourier transform, the length, n, of $\boldsymbol{S}$ will be an integral power of two; i.e., $n = 2^k$ for some positive integer k. Let L be the maximum size of claim considered. Then each element s_i of $\boldsymbol{S}$ represents the probability that a given claim is at least $\frac{i}{n}L$ but less than $\frac{(i+1)}{n}L$. The probability distribution is uniform across each such interval. In other words, the probability density function, $f(x)$, of the claim size distribution is:

$$f(x) = \begin{cases} s_i \div \left(\frac{L}{n}\right), & \text{for } \frac{i}{n}L \leq x < \frac{i+1}{n}L,\ 0 \leq i \leq n-1; \\ 0, & \text{for } x < 0 \text{ or } x \geq L. \end{cases}$$

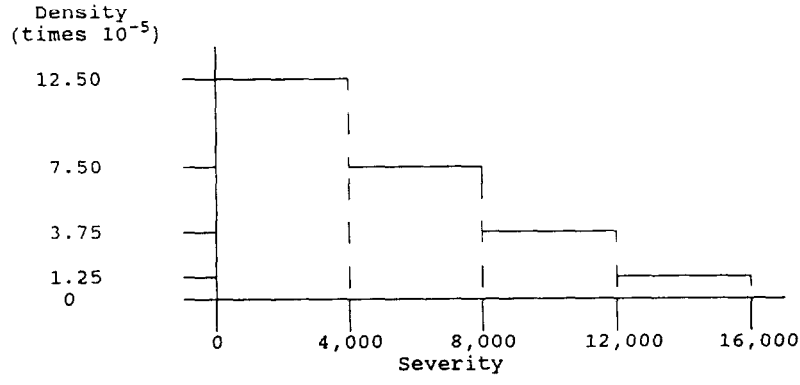
As an example, if L is \$16,000 and $\boldsymbol{S}$ is (0.50, 0.30, 0.15, 0.05), then there is a 50% probability that any given claim is between \$0 and \$4,000, a 30% probability that the claim is between \$4,000 and \$8,000, etc. The density function is uniform over the interval \$0 to \$4,000 at .000125 (= 0.50 ÷ 4,000). Similarly, the density function is uniformly .000075; i.e., (0.30 ÷ 4,000) over the interval \$4,000 to \$8,000, and so on. This is graphed in Figure 1. Note that this specification of the severity distribution is the same as that in Heckman and Meyers [2] but restricted to uniform intervals.

Define $\boldsymbol{S}^i$ inductively as follows. Let $\boldsymbol{S}^1$ be the vector of length $2n$ obtained by catenating n zeroes onto $\boldsymbol{S}$. That is,

$$s_i^1 = \begin{cases} s_i, & \text{if } 0 \leq i \leq n-1; \\ 0, & \text{if } n \leq i \leq 2n-1. \end{cases}$$

FIGURE 1

DENSITY FUNCTION REPRESENTED BY S



Let $S^{*i} = S^1 * S^{i-1}$ for $i \geq 2$.

Let S^i be the same as S^{*i} for the first n elements and be 0 for subsequent elements. Then the first n elements of S^i are the first n elements of the no-wrap convolution of S with itself i times.

Everything will be defined in greater detail below, but the algorithm is simply summarized as computing:

$$\sum_{i=M}^{N} P(i-M) \times A^i * S^i,$$

where

M is the smallest number of claims with nonzero probability; N is the largest number,

$\boldsymbol{P}(i - M)$ is the probability of exactly i claims.

$\boldsymbol{A}^i$ is a vector of "spreads" to be defined later, but, for example, $\boldsymbol{A}^3 = (1/6, 2/3, 1/6, 0, 0, 0, ..., 0)$.

$*$ is the discrete convolution operator, and

$\boldsymbol{S}^i$ is the vector that is the no-wrap convolution of the severity distribution, $\boldsymbol{S}$, with itself i times.

Very roughly speaking, $\boldsymbol{S}^i$ is the density function of the sum of the original severity distribution, $\boldsymbol{S}$, with itself i times. However, it needs to be "spread out" (in a way that will be made precise below). Certain vectors of coefficients, $\boldsymbol{A}^i$, to be defined shortly, will be used to "spread out" the $\boldsymbol{S}^i$. The distribution of exactly i claims is given by $\boldsymbol{A}^i * \boldsymbol{S}^i$. More precisely, $\boldsymbol{A}^i * \boldsymbol{S}^i[j]$ for $0 \le j \le n - 1$ is

$$F^i\left((j+1)\frac{L}{n}\right) - F^i\left(j\frac{L}{n}\right),$$

where $F^i(x)$ is the distribution function for the sum of the severity distribution with itself i times. For $n \le j \le 2n - 1$ the values of $\boldsymbol{A}^i * \boldsymbol{S}^i[j]$ are not meaningful.

To see why the $\boldsymbol{A}^i$ are needed, let $\boldsymbol{S}$ be a uniform distribution on the interval 0 to 1. Let n and L be 4. Then $\boldsymbol{S} = (1, 0, 0, 0)$. Doing the convolutions gives $\boldsymbol{S}^i = \boldsymbol{S}^1 = (1, 0, 0, 0, 0, 0, 0, 0)$ for all i. But the distribution function of the sum of the uniform distribution on [0, 1] with itself should have nonzero probability not only between 0 and 1, but also between 1 and 2. In fact, this distribution should be $(1/2, 1/2, 0, 0, 0, 0, 0, 0)$. Similarly, the sum of this distribution with itself three times should be $(1/6, 2/3, 1/6, 0, 0, 0, 0, 0)$, which has three nonzero terms. $\boldsymbol{A}^2 * \boldsymbol{S}^2$ will be $(1/2, 1/2, 0, 0, 0, 0, 0, 0)$, and $\boldsymbol{A}^3 * \boldsymbol{S}^3$ will be $(1/6, 2/3, 1/6, 0, 0, 0, 0, 0)$.

If $\boldsymbol{S}$ were a discrete distribution, instead of the piecewise uniform distribution used here, the $\boldsymbol{A}^i$ would not be necessary. (That $\boldsymbol{S} = (s_0, s_1, ...)$ is discrete means that if x is $\frac{i}{n}L$, for some i from 0 to $n - 1$, then the

probability that a claim equals x is s_i; for other x, the probability is 0. An example is given in Appendix B.)

Use of a piecewise uniform severity distribution roughly doubles the running time of the algorithm compared to the time required for a similar algorithm using a discrete severity distribution. The use of a piecewise uniform distribution is suggested because the author believes that frequently a piecewise uniform approximation with n vector elements gives a better approximation to the severity distribution than does a discrete approximation with $2n$ vector elements. If the severity distribution is more accurately approximated, then the resulting aggregate loss distribution is more accurately approximated. Also, due to memory limitations in many computers, it is often possible to compute the aggregate distribution using the piecewise uniform approximation with n vector elements, but it is not possible (easily) to compute the aggregate distribution using a discrete approximation with $2n$ vector elements.

In the next subsection the coefficients $\boldsymbol{A}^i$ are defined so they will provide the needed spread. Then, the following two subsections cover two special tactics that substantially speed the running of the algorithm. Then, the full algorithm is discussed.

The Coefficients $\boldsymbol{A}^i$

Define a_j^i by:

$$a_0^i = 1/i! \qquad \text{for } i \geq 1,$$

$$a_j^1 = 0 \qquad \text{for } j \geq 1,$$

$$a_j^i = \left(\frac{1}{i}\right)\left[\,(i-j)\,a_{j-1}^{i-1} + (j+1)\,a_j^{i-1}\,\right] \quad \text{for } i \geq 2, j \geq 1.$$

Table 1 gives the first few values of a_j^i. For example, $a_4^6 = \frac{1}{6}\left(2\,a_3^5 + 5a_4^5\right) = \frac{1}{6}\left(2 \cdot \frac{26}{120} + 5 \cdot \frac{1}{120}\right) = \frac{57}{720}$.

$\boldsymbol{A}^i$ is the vector of length $2n$ whose first n elements are given by the a_j^i, and whose last n elements are zero. For example, $\boldsymbol{A}^4 = (1/24, 11/24, 11/24, 1/24, 0, 0, \ldots, 0)$.

TABLE 1

VALUES OF a_j^i

			j			
i	0	1	2	3	4	5
1	1	0	0	0	0	0
2	1/2	1/2	0	0	0	0
3	1/6	4/6	1/6	0	0	0
4	1/24	11/24	11/24	1/24	0	0
5	1/120	26/120	66/120	26/120	1/120	0
6	1/720	57/720	302/720	302/720	57/720	1/720

The probability that the sum of i unit rectangular distributions is between j and $j+1$ for $i \geq 1$ and $j \geq 0$ is a_j^i. This is the reason these coefficients provide the needed spread. Appendix E shows this and gives a more detailed explanation of the reasons these coefficients are needed.

The numerators of the a_j^i, i.e., $i!a_j^i$, are known as *Eulerian numbers* and are discussed in Graham, Knuth, and Patashnik [19, pp. 253-258]. Feller [11, pp. 26-29] gives formulae useful in working with Eulerian numbers, although he does not mention them by name.

Packing and Unpacking

Two special tactics are applied to make the algorithm run faster. One is to "pack" the severity distribution into a vector, so that the computation of the discrete Fourier transform of a given real vector of length $2n$ is accomplished, instead, by the computation of the discrete Fourier transform of a related complex vector of length n. This tactic roughly doubles the speed of the algorithm (with no effect on accuracy because the DFT of the original vector of length $2n$ is still what is finally computed), and is discussed in this sub-section. The second tactic is to compute the M-fold convolution of the severity distribution with itself using a method which, for M greater than 3, is faster than the naive method that computes $M-1$ convolutions. This tactic is discussed in the next subsection.

The fast Fourier transform operates on vectors of complex numbers, but here it is used only to transform vectors of real numbers. As such, half of the place values are not really being used, because the imaginary parts of the elements of the input vector are all zero. Clever use of certain symmetry properties of discrete Fourier transforms of purely real vectors and purely imaginary vectors, as discussed in Press, et al. [6, p. 398], allows the following.

To transform the length $2n$ vector $\boldsymbol{V} = (v_0, v_1, \ldots, v_{2n-1})$, where each v_i is a real number, rewrite $\boldsymbol{V}$ as $\boldsymbol{PV} = (v_0 + iv_1, v_2 + iv_3, \ldots, v_{2n-2} + iv_{2n-1})$, where i is $\sqrt{-1}$. This is now a complex vector of length n. $\boldsymbol{PV}$ is referred to as the *packed untransformed* vector (it is packed because it is written in a more compact form; it is untransformed because the discrete Fourier transformation has not yet been applied). Compute the discrete Fourier transform of $\boldsymbol{PV}$, and call it $\boldsymbol{FPV}$. $\boldsymbol{FPV}$ is the *packed transformed* vector. Some simple computations on $\boldsymbol{FPV}$, called *unpacking*, yield $\boldsymbol{FV}$, the (unpacked) discrete Fourier transform of $\boldsymbol{V}$. While $\boldsymbol{FV}$ is a vector of length $2n$, if the first $n + 1$ elements of $\boldsymbol{FV}$ are known, then the remaining $n - 1$ elements can be deduced using a formula from Appendix F. Similarly, one can pack the transformed vector in such a way that when the inverse discrete Fourier transform is applied, the untransformed vector appears in the form of $\boldsymbol{PV}$. Note, in particular, that to apply the convolution theorem to real vectors of length $2n$, one can instead work with complex vectors whose lengths never exceed $n + 1$.

Packing and unpacking the untransformed vectors is trivial. Depending on how one represents real and complex vectors this might be a simple rearrangement, or just a redefinition of the meaning of each element of an array. Usually no calculations are needed. Packing and unpacking the transformed vectors does involve some calculation, but not a great amount. Details of the algorithms to pack and unpack are given in Appendix F, and an APL implementation is given in the functions PACK and UNPACK in Appendix G.

Table 2 shows the steps involved in computing the convolution of two real vectors when one packs the vectors.

TABLE 2

CONVOLUTION USING PACKED VECTORS

Vector(s)	Step	Real or Complex	Length	Transf or Untransf	Packed or Unpacked
V, W	Start	Real	$2n$	Untransf	Unpacked
⇓					
PV, PW	Pack	Complex	n	Untransf	Packed
⇓					
FPV, FPW	Apply DFT	Complex	n	Transf	Packed
⇓					
FV, FW	Unpack	Complex	$n+1$	Transf	Unpacked
⇓					
$FU = FV \times FW$	Multiply	Complex	$n+1$	Transf	Unpacked
⇓					
FPU	Pack	Complex	n	Transf	Packed
⇓					
PU	Inv DFT	Complex	n	Untransf	Packed
⇓					
U	Unpack	Real	$2n$	Untransf	Unpacked

While packing and unpacking add four steps to the above, the time saved by transforming vectors of length n instead of $2n$ is more than offsetting. In practice, the untransformed vectors are usually kept packed, thus further reducing the number of steps.

Binary Exponentiation

Before discussing the second special tactic directly, consider an analogous question: how many multiplications are needed to compute 2^{100}? One way to compute 2^{100} is to start with 2, and then repeatedly multiply by 2, doing 99 multiplications. Another way is to use the following formula:

$$2^{100} = (((((\,2^2 \times 2\,)^2)^2)^2 \times 2)^2)^2 .$$

Each operation of squaring is one multiplication and twice an intermediate result is multiplied by 2, so this computes 2^{100} with only eight multiplications.

In general, to compute a^n for $n \geq 1$, one can apply the following algorithm. Express n as a binary number, b, and drop the left-most digit (which is always 1). Set z equal to a. Loop: if there are no digits left in b, then stop; z is a^n. If there is at least one digit remaining in b, square z. If the current left-most digit of b is 1, then multiply z by a. Drop the left-most digit from b. Go back to the step labeled "Loop."

The binary representation of 100 is 1100100. Dropping the first digit gives 100100. Following the steps above, set z to 2, then square, multiply by 2, square, square, square, multiply by 2, square, and square.

This is called a *left-to-right binary method for exponentiation* and is discussed in Knuth [5, Vol. 2, p. 441] (along with even faster methods).

This is used as follows. For some applications of the overall algorithm, the smallest number of claims with nonzero probability, M, will be greater than one. In these cases, this method is used to compute $\boldsymbol{S}^M = \boldsymbol{S} * \boldsymbol{S} * \ldots * \boldsymbol{S}$ (with M factors of $\boldsymbol{S}$, here the $*$ is the no-wrap convolution). That is, M is written as a binary number, and the left-to-right binary method is applied, with no-wrap convolution at each step instead of multiplication. Since convolution is associative, $\boldsymbol{S}^M$ is well defined, and this is a correct way to compute $\boldsymbol{S}^M$.

The Full Algorithm

Now the full algorithm can be described. Figure 2 outlines the algorithm using a flowchart. The notation used is described in the presentation of the complete algorithm, below. A summary of the meaning of each variable is given in Table 3.

The complete algorithm is as follows.

Algorithm for Aggregate Loss Distribution: Let M be the smallest number of claims with nonzero probability, let N be the largest number of such claims ($N \geq 1$ and $N \geq M$), let $\boldsymbol{P}$ be the vector of probabilities of $M, M+1, M+2, \ldots, N$ claims, and let $\boldsymbol{S}$ be the vector representing the density function of the claim severity distribution. The length of $\boldsymbol{S}$ is $n = 2^k$ for some positive integer k. All vectors are indexed starting at 0, so

FIGURE 2

FLOWCHART FOR COMPUTATION OF AGGREGATE LOSS DISTRIBUTION

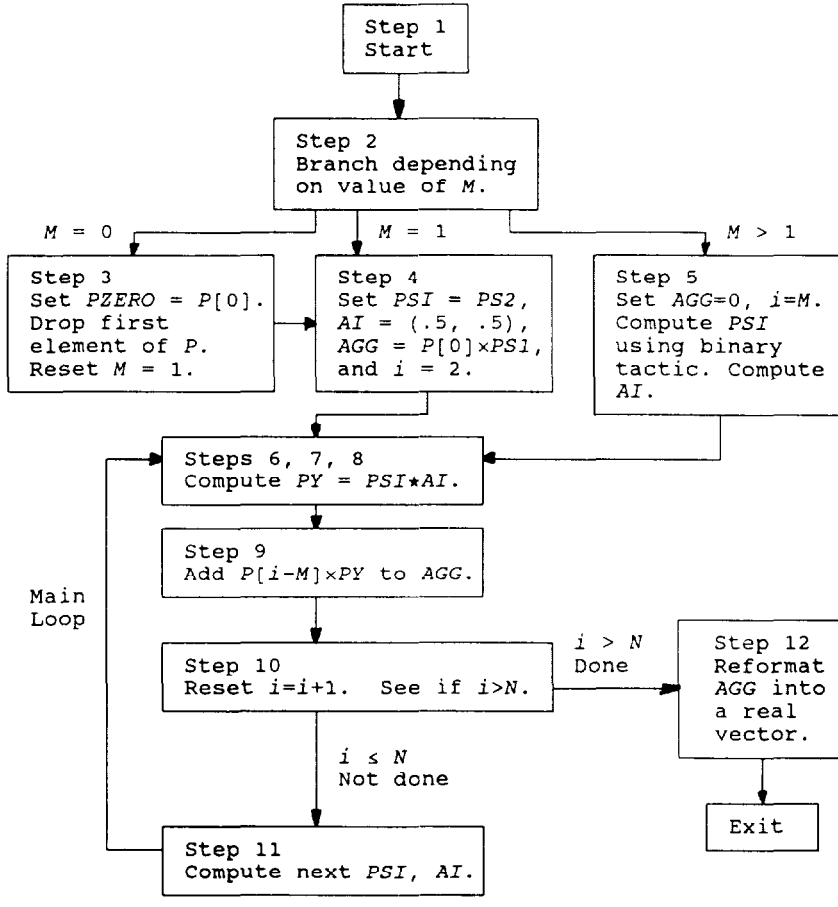


TABLE 3

VARIABLES USED IN THE MAIN ALGORITHM

INPUT VARIABLES:

n Length of vectors which will be subjects of the FFT or inverse FFT. $n = 2^k$ for some positive integer k.

M Smallest number of claims with nonzero probability.

N Largest number of claims with nonzero probability.

P Probabilities of *M* through *N* claims (p_0 = probability of *M* claims, p_1 = probability of $M + 1$ claims, ...).

S Severity distribution. ***S*** is a real vector of length *n*. $\boldsymbol{S} = (s_0, s_1, \ldots, s_{n-1})$

MAJOR VARIABLES:

i Index giving the current number of claims.

PSI Packed severity distribution, $(s_0 + is_1, s_2 + is_3, \ldots s_{n-2} + is_{n-1}, 0, 0, \ldots, 0)$. (Here i is $\sqrt{-1}$.)

FSI Transformed (unpacked) severity distribution.

PSI Packed severity distribution convolved with self *i* times.

SI Severity distribution convolved with self *i* times. (Unpacked ***PSI***.)

FSI Transformed severity distribution convolved with self i times.

AI Vector of spread coefficients, $(a_0^i, a_1^i, \ldots) = \boldsymbol{A}^i$.

AGG Aggregate distribution.

MINOR VARIABLES:

PZERO Probability of zero claims.

BIN Initialized to the binary representation of M, and used in the binary exponentiation tactic.

FSIFLAG Flag to determine whether ***FSI*** has been computed for the current *i*.

X Unpacked ***PSI***.

j Index used in Step 7.

Y Becomes ***AI*X***.

PY Packed ***Y***.

FAI Transformed ***AI***.

FY Transformed ***Y*** = ***FSI*** × ***FAI***.

the indices for ***S*** run from 0 to $n-1$. The result will be ***AGG***, a vector representing the aggregate distribution.

1. [Some initializations.] Set ***AGG*** to be a complex vector of length n all of whose elements are zero. Pack ***S*** into the complex vector ***PSI*** of length n so:

$$\boldsymbol{PSI} = \left(s_0 + is_1, s_2 + is_3, \ldots, s_{n-2} + is_{n-1}, 0, 0, \ldots, 0 \right).$$

   Here i is $\sqrt{-1}$. Set *PZERO*, the probability of exactly 0 claims, to 0.

2. [Branch depending on the value of M.] If M is 0, go to Step 3. If M is 1, go to Step 4. If M is greater than 1, go to Step 5.

3. [Initialize if $M = 0$.] Let *PZERO* be ***P***[0]. Drop the first element from ***P***. Let M be 1.

4. [Initialize if $M = 0$ or 1.] Let ***AGG*** be ***P***[0] times ***PSI***. If N is 1, go to Step 10. Let ***FSI*** be the unpacked DFT of ***PSI***. Let ***PSI*** be the inverse DFT of the packed pointwise product of ***FSI*** with itself. Set the last $n/2$ elements of ***PSI*** to 0. Let ***AI*** be the two-element vector (0.5, 0.5). Let i be 2. Go to Step 6.

5. [Begin procedure if $M > 1$.] Let *BIN* be the binary representation of M. Drop the first (left-most) digit from *BIN*. Let ***FSI*** and ***FSI*** be the unpacked DFT of ***PSI***.

5.1. [Convolve ***PSI*** with itself or ***PSI*** with itself.] Let ***FSI*** be ***FSI*** times itself. Let ***PSI*** be the inverse DFT of the packed ***FSI***. Set the last $n/2$ elements of ***PSI*** to 0. If the first digit of *BIN* is 0, go to Step 5.3.

5.2. [Convolve ***PSI*** with ***PSI***.] Let ***FSI*** be the unpacked DFT of ***PSI***. Let ***FSI*** be ***FSI*** times ***FSI***. Let ***PSI*** be the inverse DFT of the packed ***FSI***. Set the last $n/2$ elements of ***PSI*** to 0.

5.3. [Check whether finished.] Drop the first digit from *BIN*. If there are no digits left, go to Step 5.4 Otherwise, let ***FSI*** be the unpacked DFT of ***PSI***. Go to Step 5.1.

5.4. [Initialize ***AI*** in the case $M > 1$.] Use the formula:

$$a_0^i = 1/i! \qquad \text{for } i \geq 1,$$

$$a_j^1 = 0 \qquad \text{for } j \geq 1,$$

$$a_j^i = \left(\frac{1}{i}\right)\left[(i-j)\,a_{j-1}^{i-1} + (j+1)\,a_j^{i-1}\right] \qquad \text{for } i \geq 2,\ j \geq 1$$

to compute $\boldsymbol{AI} = \boldsymbol{A}^M = \left(a_0^M, a_1^M, \ldots, a_{M-1}^M\right)$. Let i equal M.

6. [Start main loop.] (When this step is reached for the first time, ***FSI*** has been computed, ***PSI*** and ***AI*** have been computed for some i at least 2, and ***AGG*** has been initialized.) Let *FSIFLAG* be 0 (will be used later to determine whether ***FSI*** has been computed). If i is greater than or equal to 100 go to Step 8.

7. [Convolve ***AI*** and ***SI*** without using DFTs. (***SI*** is the severity distribution convolved with itself i times.)] Unpack ***PSI*** and let ***X*** be the first n elements of the result. Let j be 0 and ***Y*** be a (real) vector of length n with all elements 0.

7.1 [Loop.] Let ***Y*** be ***Y*** plus ***AI***[j] times ***X***. Add 1 to j. If j is greater than i minus 1 go to Step 7.2. Drop the last element of ***X*** and add a 0 as the first element. Return to the start of this Step (7.1).

7.2 [Exit Step 7.1.] Add n zeroes to the end of ***Y***, pack and call the result ***PY***. Go to Step 9.

8. [Convolve ***AI*** and ***SI*** using FFTs.] If ***AI*** is of length less than n, add zeroes until a vector of length $2n$ is achieved; otherwise take the first n elements of ***AI*** and append n zeroes. Pack this vector, compute the DFT, unpack the result and assign it to ***FAI***. Compute the DFT of ***PSI***, unpack, and assign the result to ***FSI***. Set *FSIFLAG* to 1. Let ***FY*** be ***FSI*** times ***FAI***. Pack ***FY***, apply the inverse DFT, and assign the result to ***PY***. Set the last $n/2$ (complex) elements of ***PY*** to 0.

9. [Add new packed ***AI*** * ***SI*** (= ***PY***) to ***AGG***.] Let ***AGG*** be ***AGG*** plus ***P***[$i - M$] times ***PY***.

10. [Increment i.] Set i to i plus 1. If i is greater than N, go to Step 12.

11. [Compute next ***AI***, ***PSI***.] If *FSIFLAG* is 0, compute the DFT of ***PSI***, unpack, and assign the result to ***FSI***. (If *FSIFLAG* is 1, it is because ***FSI*** was computed in Step 8.) Let ***FSI*** be ***FSI*** times ***FSI***. Pack ***FSI***, compute the inverse DFT, and assign this to ***PSI***. Let the last $n/2$ elements of ***PSI*** be 0. Compute the next ***AI*** using the formula in Step 5.4. Go to Step 6.

12. [The end.] Let ***AGG*** be the first n elements of ***AGG*** unpacked. Add *PZERO* to ***AGG***[*0*].

The first two steps do some initializations and branch depending on the value of M. If M is 0, the algorithm essentially converts to the case where M is 1. If M is 1 (or 0) the first steps set ***AGG*** to be ***PSI***, compute ***FSI***, and compute ***PSI*** and ***AI*** for $i = 2$. Then comes the main loop. If M is greater than 1, the binary exponentiation tactic is used to compute ***PSI*** for $i = M$. ***AI*** is also computed for $i = M$.

The main loop repeatedly computes the convolution of ***AI*** with ***SI***, multiplies this by ***P***[i - M], and adds the result to ***AGG*** (actually, the untransformed packed result is added to ***AGG***). Note that the convolution of ***AI*** with ***SI*** is the distribution of exactly i claims. If i is less than or equal to N the next ***PSI*** and ***AI*** are computed and one continues with the main loop. (Note that each ***PSI*** is computed from the ***PSI*** for the previous i.) If i is greater than N, one exits the main loop, reformats ***AGG***, and folds in *PZERO*. Observe that ***FSI*** is always recomputed from a current ***PSI*** which has had its "tail" (the last $n/2$ elements) set to zero. This gives the no-wrap convolutions that are needed, instead of regular convolutions.

Sometimes the algorithm computes the convolution of ***AI*** with ***PSI*** through use of the convolution theorem and FFTs, and sometimes it performs this computation directly. For "short" ***AI***, implicitly defined above as having 100 or fewer nonzero terms, it is faster to compute the convolution directly. Once ***AI*** becomes "long," it is faster to use the convolution theorem. In another implementation (using different hardware or software), it might be more efficient to set this cut-off of 100 to a higher or lower number.

Note that if there are fewer than n nonzero terms in $\boldsymbol{AI}$ only the nonzero terms are kept, and one pads to the right with zeroes when a vector of length n is needed. This differs slightly from definitions of $\boldsymbol{AI}$ given earlier.

APL functions implementing the complete algorithm are given in Appendix H.

4. EXAMPLES AND ADDITIONAL DISCUSSION

This section will give some examples of the use of the algorithm, show how parameter uncertainty can be reflected in the aggregate distribution, and discuss the computation of aggregate excess distributions. Comments in the first two areas are specific to this fast Fourier transform algorithm, but comments on the third topic apply generally.

Examples

Three examples of use of this algorithm will be given. The first is simple and is intended to be easy to reproduce in order to test an actual implementation. It is not meant to be realistic. The second example is more typical of actual distributions that arise in practice. The third example is reasonably realistic, but is really meant to illustrate the flexibility of the algorithm.

The first example will compute the distribution of *exactly* five claims, with each claim following a uniform severity distribution. Let $k = 5$, so $n = 32$. (That $k = 5$ has nothing to do with the fact that the distribution of five claims is being computed; this is a coincidence.) The claim count distribution is defined by setting $M = 5$, and making $\boldsymbol{P}$ be a vector with one element, (1). The severity distribution, $\boldsymbol{S}$, is a vector of length 32 (= n), and L is set to 6.4, so each element of $\boldsymbol{S}$ covers a range of 0.2 (= 6.4/32). Letting $\boldsymbol{S}$ be the uniform distribution on [0,1], gives:

$$\boldsymbol{S} = (0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, \ldots, 0).$$

The output is a vector of length 32, $\boldsymbol{AGG}$, a complete listing of which is given in Table 4. Since the sum of five claims, each no greater than 1, cannot exceed 5, only the first 25 elements of the output are nonzero. It is

TABLE 4

AGGREGATE DISTRIBUTION FOR EXAMPLE 1

Index	Range	*AGG*[*i*]	Cumulative Distribution
0	0.0 - 0.2	0.000003	0.000003
1	0.2 - 0.4	0.000083	0.000085
2	0.4 - 0.6	0.000563	0.000648
3	0.6 - 0.8	0.002083	0.002731
4	0.8 - 1.0	0.005603	0.008333
5	1.0 - 1.2	0.012389	0.020723
6	1.2 - 1.4	0.023669	0.044392
7	1.4 - 1.6	0.039749	0.084141
8	1.6 - 1.8	0.059669	0.143811
9	1.8 - 2.0	0.081189	0.225000
10	2.0 - 2.2	0.100816	0.325816
11	2.2 - 2.4	0.114496	0.440312
12	2.4 - 2.6	0.119376	0.559688
13	2.6 - 2.8	0.114496	0.674184
14	2.8 - 3.0	0.100816	0.775000
15	3.0 - 3.2	0.081189	0.856189
16	3.2 - 3.4	0.059669	0.915859
17	3.4 - 3.6	0.039749	0.955608
18	3.6 - 3.8	0.023669	0.979277
19	3.8 - 4.0	0.012389	0.991667
20	4.0 - 4.2	0.005603	0.997269
21	4.2 - 4.4	0.002083	0.999352
22	4.4 - 4.6	0.000563	0.999915
23	4.6 - 4.8	0.000083	0.999997
24	4.8 - 5.0	0.000003	1.000000
25	5.0 - 5.2	0.000000	1.000000
26	5.2 - 5.4	0.000000	1.000000
27	5.4 - 5.6	0.000000	1.000000
28	5.6 - 5.8	0.000000	1.000000
29	5.8 - 6.0	0.000000	1.000000
30	6.0 - 6.2	0.000000	1.000000
31	6.2 - 6.4	0.000000	1.000000

straightforward to check that the result is the sum of five uniform distributions. (Recall that the $\boldsymbol{A}^i$ summarize the sums of uniform distributions.) For example, the sum of the first five elements of the result is .008335, which (up to rounding error) is 1/120, and this agrees with the theoretical sum. Similarly, the sums of the second through fifth sets of five elements are 26/120, 66/120, 26/120, and 1/120. (Ambitious readers can check that the intermediate values are also correct. Feller [11, p. 27] gives the needed formulae.)

The second example is more in line with distributions that arise in practice. In this example, $k = 10$, so $n = 1{,}024$. The claim count distribution is negative binomial with mean 10 and variance 12. For input, M equals 0 and $\boldsymbol{P}$ is a vector of length 42, giving the probabilities of 0 through 41 claims. Actually, for the probability of 41 claims the probability of 41 or more claims is used, so the total of the elements of $\boldsymbol{P}$ is exactly 1.0. The probability of there being 42 or more claims is less than 10^{-10}, so including this in the probability of there being exactly 41 claims is not significant. The values of $\boldsymbol{P}$ are shown in Table 5. This distribution is shown in detail for the benefit of readers who want to reproduce this example.

The severity distribution is a two-parameter Weibull distribution with mean \$10,000 and coefficient of variation 8 (standard deviation divided by mean). The mean and coefficient of variation completely define the Weibull distribution. For the interested reader, note that the parameterization of the Weibull distribution used has distribution function $F(x) = 1 - e^{-(x/b)^c}$ with $b = 454.82609$ and $c = 0.25371$. Any other severity distribution could be used in place of the Weibull, including lognormal, Pareto, or an empirical fit to data. Losses are capped at \$250,000 per loss. An L of \$1,000,000 is chosen to be large enough to cover the highest values needed in the aggregate distribution. $\boldsymbol{S}$ is then a vector of length 1,024 with each element covering a range of \$1,000,000/1,024, or about \$977. As losses are capped at \$250,000, only the first 256 elements of $\boldsymbol{S}$ will be nonzero.

To determine the values of $\boldsymbol{S}$, a variant of a method of Venter [20, p. 21] is used. $\boldsymbol{S}$ is to be a piecewise uniform approximation to the Weibull

TABLE 5

CLAIM COUNT DISTRIBUTION FOR EXAMPLE 2

Number of Claims	Probability of Given Number of Claims	Cumulative Distribution
0	1.09885E-4	1.09885E-4
1	9.15707E-4	1.02559E-3
2	0.00389175	0.00491735
3	0.01124284	0.01616019
4	0.02482795	0.04098814
5	0.04469031	0.08567845
6	0.06827686	0.15395531
7	0.09103581	0.24499112
8	0.10810503	0.35309615
9	0.11611281	0.46920896
10	0.11417760	0.58338656
11	0.10379781	0.68718437
12	0.08793981	0.77512418
13	0.06990088	0.84502506
14	0.05242566	0.89745072
15	0.03728047	0.93473119
16	0.02524198	0.95997317
17	0.01633305	0.97630622
18	0.01013254	0.98643876
19	0.00604397	0.99248273
20	0.00347528	0.99595801
21	0.00193071	0.99788873
22	0.00103849	0.99892722
23	5.41821E-4	0.99946904
24	2.74673E-4	0.99974371
25	1.35505E-4	0.99987922
26	6.51468E-5	0.99994436
27	3.05627E-5	0.99997492
28	1.40079E-5	0.99998893
29	6.27940E-6	0.99999521
30	2.75596E-6	0.99999797
31	1.18536E-6	0.99999915
32	5.00074E-7	0.99999965
33	2.07101E-7	0.99999986
34	8.42617E-8	0.99999994
35	3.37047E-8	0.99999998
36	1.32634E-8	0.99999999
37	5.1381E-9	1.00000000
38	1.9606E-9	1.00000000
39	7.373E-10	1.00000000
40	2.734E-10	1.00000000
41	1.560E-10	1.00000000

distribution. To achieve this, pairs of consecutive elements (s_{2i}, s_{2i+1}) of $\boldsymbol{S}$ are chosen subject to two constraints:

- over each interval $\left[2i \cdot \frac{L}{n}, (2i+2) \cdot \frac{L}{n} \right]$ the integral of $\boldsymbol{S}$ and the integral of the Weibull density are the same, and
- over the same intervals the integral of the first moment distribution of $\boldsymbol{S}$ and the first moment distribution of the Weibull are the same.

If RR and SS are the integral of the density function and the integral of the first moment distribution of the Weibull (or whatever distribution is being approximated) over the above interval then

$$s_{2i} = \left(2i + \frac{3}{2} \right) RR - \frac{SS \cdot n}{L}$$

$$s_{2i+1} = RR - s_{2i}$$

No properties of the Weibull are used in the formulae above; they apply to any distribution, including empirical distributions.

When this method is applied in this particular case, the second element of $\boldsymbol{S}$ becomes negative. Thus, some additional fiddling is done on the first 12 elements of $\boldsymbol{S}$ so that the two constraints are satisfied for these 12 elements taken together, but not pair-wise. Selected values for $\boldsymbol{S}$ (including the first 12 elements) are shown in Table 6. Note in particular that the value of .007072 is the probability of the severity being in the interval \$249,023 to \$250,000 plus the probability of the severity being over \$250,000. This latter probability has been spread over the interval.

Selected values of the resulting aggregate distribution are shown in Table 7. Column (3) is selected elements of the vector ***AGG*** output by the algorithm. Each element, t_i, of ***AGG*** is the (exact) integral of the density function for the aggregate distribution over the interval $\left[i \cdot \frac{L}{n}, (i+1) \cdot \frac{L}{n} \right]$. For the purpose of interpolating values between integral multiples of $L \div n$ it is assumed that the density at each point in an interval is $t_i \div \left(\frac{L}{n} \right)$. Column (4) is the distribution function,

TABLE 6

DISTRIBUTION OF SEVERITY OF SINGLE CLAIM FOR EXAMPLE 2

		Probability Distribution of a Single Claim	
Index	High End of Range*	$S[i]$	Distribution
0	$977	0.716463	0.716463
1	1,953	0.114281	0.830745
2	2,930	0.015000	0.845745
3	3,906	0.015000	0.860745
4	4,883	0.010000	0.870745
5	5,859	0.005000	0.875745
6	6,836	0.005000	0.880745
7	7,813	0.005000	0.885745
8	8,789	0.003000	0.888745
9	9,766	0.003000	0.891745
10	10,742	0.003000	0.894745
11	11,719	0.003000	0.897745
12	12,695	0.004750	0.902494
13	13,672	0.004140	0.906634
14	14,648	0.003884	0.910518
15	15,625	0.003441	0.913959
74	73,242	0.000331	0.973485
75	74,219	0.000321	0.973806
76	75,195	0.000316	0.974122
77	76,172	0.000307	0.974429
78	77,148	0.000302	0.974732
101	99,609	0.000194	0.980249
102	100,586	0.000192	0.980440
103	101,563	0.000187	0.980628
123	121,094	0.000137	0.983819
124	122,070	0.000136	0.983955
125	123,047	0.000133	0.984088
235	230,469	0.000041	0.992213
236	231,445	0.000041	0.992254
237	232,422	0.000040	0.992294
253	248,047	0.000035	0.992893
254	249,023	0.000035	0.992928
255	250,000	0.007072	1.000000
256	250,977	0.000000	1.000000
1,023	1,000,000	0.000000	1.000000

*Each range has width of about 977.

TABLE 7

DISTRIBUTION OF AGGREGATE LOSS AND FIRST MOMENT DISTRIBUTION FOR EXAMPLE 2

		Aggregate Loss Distribution		First Moment Distribution	
Index	High End of Range*	*AGG*[*i*]	Distribution	Density**	Distribution**
(1)	(2)	(3)	(4)	(5)	(6)
0	$977	0.002812	0.002812	1.373	1.373
1	1,953	0.010576	0.013387	15.492	16.865
2	2,930	0.021673	0.035061	52.913	69.778
3	3,906	0.031214	0.066275	106.689	176.467
4	4,883	0.036124	0.102399	158.750	335.217
5	5,859	0.036358	0.138758	195.284	530.501
6	6,836	0.033510	0.172268	212.712	743.213
7	7,813	0.029312	0.201580	214.687	957.900
12	12,695	0.013517	0.294650	165.005	1,884.273
13	13,672	0.012153	0.306802	160.216	2,044.489
14	14,648	0.011327	0.318129	160.388	2,204.877
15	15,625	0.010936	0.329065	165.539	2,370.416
74	73,242	0.003134	0.691222	228.013	16,154.978
75	74,219	0.003082	0.694304	227.219	16,382.197
76	75,195	0.003031	0.697335	226.418	16,608.615
77	76,172	0.002981	0.700316	225.611	16,834.226
78	77,148	0.002932	0.703248	224.797	17,059.023
123	121,094	0.001553	0.798766	187.309	26,306.223
124	122,070	0.001534	0.800300	186.519	26,492.741
125	123,047	0.001515	0.801816	185.732	26,678.473
150	147,461	0.001138	0.834470	167.189	31,076.113
151	148,438	0.001125	0.835595	166.494	31,242.606
152	149,414	0.001113	0.836709	165.802	31,408.408
153	150,391	0.001101	0.837810	165.114	31,573.522
154	151,367	0.001090	0.838900	164.429	31,737.951
235	230,469	0.000520	0.899889	119.608	43,086.466
236	231,445	0.000516	0.900405	119.168	43,205.634
237	232,422	0.000512	0.900916	118.731	43,324.364
253	248,047	0.000453	0.908589	112.035	45,166.396
254	249,023	0.000449	0.909038	111.635	45,278.031
255	250,000	0.000507	0.909546	126.564	45,404.595
256	250,977	0.000830	0.910376	207.973	45,612.569
283	277,344	0.000836	0.949716	231.421	55,893.081
284	278,320	0.000813	0.950530	225.980	56,119.061
285	279,297	0.000792	0.951322	220.846	56,339.908
419	410,156	0.000097	0.989983	39.907	68,828.765
420	411,133	0.000096	0.990080	39.546	68,868.312
1,022	999,023	0.000000	0.999996	0.052	73,804.421
1,023	1,000,000	0.000000	0.999996	0.058	73,804.479

* Each range has width of about 977.

** These columns have been multiplied by the aggregate mean. This table gives selected values of the distributions.

$t_0 + t_1 + \ldots + t_j$, i.e., the cumulative sum of Column (3). Column(4) readily gives the dollar amounts (frequently called *confidence levels* in the context of aggregate loss distributions) associated with given probability levels, and vice versa. For instance, the probability that aggregate losses will be less than or equal to \$250,000 is 91.0%. By simple interpolation, it is seen that \$75,000 corresponds to the 69.7% confidence level. Again through interpolation, the 80% confidence level is \$121,879.

The expected dollars of loss above and below given aggregate limits can be quickly determined. Suppose, for example, an insurer has purchased reinsurance that covers all loss amounts beyond a total of \$250,000. That is, the insurer pays the first \$250,000 of losses (which could be one claim or a number of claims), and the reinsurer pays any losses after the first \$250,000. The insurer's expected loss is:

$$\int_0^{250{,}000} x f(x)\, dx + 250{,}000 \int_{250{,}000}^{\infty} f(x)\, dx$$

where $f(x)$ is the density function of the aggregate distribution. As Columns (5) and (6) in Table 7 will help calculate this integral, these columns are described next.

Each entry in Column (5) is $\int x f(x)\, dx$ over its interval. For instance, under the above assumption that the density function is a constant $0.036358 \div 976.5625$ across the fifth interval, $\int x f(x)\, dx$ over the fifth interval is

$$\frac{(5{,}859.375)^2 - (4{,}882.8125)^2}{2} \times \frac{0.036358}{976.5625}$$

or 195.282. (This is slightly different from 195.284 shown in Column (5) because the Column (3) entry of 0.036358 is used in the above calculation, while more significant digits were used in the calculation of Table 7.) Column (6) is the cumulative sum of Column (5). Thus, Column (6) gives

$$\int_0^{(i+1)\frac{L}{n}} x f(x)\, dx$$

where i is the index of the interval.

Returning to the original question, if losses are capped at an aggregate of \$250,000, Columns (6) and (4) show that expected losses are $45{,}405 + 250{,}000 \times [1 - 0.909546]$, or \$68,019.

The reinsurer is taking both occurrence and aggregate excess of \$250,000, and total expected losses are \$100,000 (10 expected claims times \$10,000 expected loss per claim), so the reinsurer's expected losses are \$31,981 (100,000 - 68,019).

Generally, to compute expected losses for an insurer that retains a given amount per occurrence and retains a given aggregate, use a severity distribution capped at the per occurrence limit, compute the aggregate distribution, and compute the expected retained losses up to the aggregate limit as was just done above.

The third example is a variation on the second example. The main purpose is to show how easy it is to use an arbitrary frequency distribution in the algorithm. For this example, the above frequency distribution is modified to assume that there is a 90% probability that claims will follow the distribution in example 2, and an additional 10% probability there will be exactly 20 claims. The same $\boldsymbol{S}$ as above is used. The modified $\boldsymbol{P}$ is shown in Table 8, and some of the output is given in Table 9.

Note that the severity distribution, $\boldsymbol{S}$, used in examples 2 and 3 is only an approximation to the Weibull distribution. Essentially, having to find a piecewise uniform function to approximate the true severity distribution, with each interval being the same size, $L \div n$, means that $\boldsymbol{S}$ is not going to be precisely the same as the original continuous function. Since $\boldsymbol{S}$ is an approximation to the true severity distribution, the output is an approximation to the true aggregate distribution. Comparisons of aggregate loss distributions computed using this algorithm to aggregate loss distributions

TABLE 8

CLAIM COUNT DISTRIBUTION FOR EXAMPLE 3

Number of Claims	Probability of Given Number of Claims	Cumulative Distribution
0	9.88963E-5	9.88963E-5
1	8.24136E-4	9.23032E-4
2	0.00350258	0.00442561
3	0.01011856	0.01454417
4	0.02234515	0.03688933
5	0.04022128	0.07711060
6	0.06144917	0.13855978
7	0.08193223	0.22049201
8	0.09729453	0.31778654
9	0.10450153	0.42228806
10	0.10275984	0.52504790
11	0.09341803	0.61846593
12	0.07914583	0.69761177
13	0.06291079	0.76052256
14	0.04718309	0.80770565
15	0.03355242	0.84125807
16	0.02271779	0.86397586
17	0.01469974	0.87867560
18	0.00911929	0.88779488
19	0.00543957	0.89323446
20	0.10312775	0.99636221
21	0.00173764	0.99809985
22	9.34641E-4	0.99903449
23	4.87639E-4	0.99952213
24	2.47206E-4	0.99976934
25	1.21955E-4	0.99989129
26	5.86321E-5	0.99994993
27	2.75064E-5	0.99997743
28	1.26071E-5	0.99999004
29	5.65146E-6	0.99999569
30	2.48036E-6	0.99999817
31	1.06682E-6	0.99999924
32	4.50066E-7	0.99999969
33	1.86391E-7	0.99999987
34	7.58356E-8	0.99999995
35	3.03342E-8	0.99999998
36	1.19371E-8	0.99999999
37	4.6243E-9	1.00000000
38	1.7645E-9	1.00000000
39	6.636E-10	1.00000000
40	2.461E-10	1.00000000
41	1.404E-10	1.00000000

TABLE 9

DISTRIBUTION OF AGGREGATE LOSSES FOR EXAMPLE 3

		Probability Distribution of Aggregate Losses	
Index	High End of Range*	***AGG[i]***	Distribution
0	$977	0.002530	0.002530
1	1,953	0.009518	0.012049
2	2,930	0.019506	0.031555
3	3,906	0.028093	0.059647
4	4,883	0.032512	0.092159
82	81,055	0.002900	0.684594
83	82,031	0.002857	0.687451
84	83,008	0.002815	0.690266
85	83,984	0.002773	0.693039
86	84,961	0.002733	0.695772
87	85,938	0.002694	0.698466
88	86,914	0.002655	0.701121
89	87,891	0.002617	0.703738
138	135,742	0.001420	0.797823
139	136,719	0.001405	0.799228
140	137,695	0.001390	0.800617
141	138,672	0.001375	0.801992
142	139,648	0.001360	0.803352
256	250,977	0.000847	0.898052
257	251,953	0.001418	0.899470
258	252,930	0.002059	0.901529
259	253,906	0.002517	0.904046
260	254,883	0.002688	0.906734
297	291,016	0.000691	0.948843
298	291,992	0.000679	0.949521
299	292,969	0.000666	0.950187
300	293,945	0.000654	0.950841
301	294,921	0.000642	0.951482
451	441,406	0.000091	0.989829
452	442,383	0.000090	0.989919
453	443,359	0.000089	0.990009
454	444,336	0.000088	0.990097
455	445,313	0.000087	0.990185
1,021	998,047	0.000000	0.999991
1,022	999,023	0.000000	0.999991
1,023	1,000,000	0.000000	0.999991

* Each range has a width of about 977. This table gives selected values of the aggregate *distribution.*

computed using other methods indicate that the algorithm presented here gives very accurate answers.

Apart from the need to use a severity distribution that is an approximation to the true distribution, the algorithm here is precise in the following sense. Each element of the aggregate loss vector is the exact difference of the distribution function for the exact aggregate loss distribution over the interval that corresponds to the element. In particular, this algorithm is not subject to the convergence difficulties sometimes encountered in certain characteristic function methods when the probability of a maximum loss is high. Of course, there is some potential for rounding error, but most computer languages have a provision for doing calculations to at least 17 decimal place accuracy, and as each element of the result (for $n = 1{,}024$; 20 expected claims) is affected by about 100,000 calculations, the result should be accurate to at least 12 places.

Note also, once Column (3) of Table 7 has been calculated, how simple and fast it is to calculate Columns (4), (5), and (6). Once ***AGG*** has been computed, there is very little computation time needed to determine confidence levels, expected losses subject to an aggregate, or expected losses excess of an aggregate. Also, since the entire aggregate loss distribution (up to some limit) is computed, the computation of any quantity that is related to the aggregate distribution (e.g., expected sliding scale commission for a reinsurance contract) is straightforward and fast.

Computational Considerations

The computational time for this algorithm seems to be roughly proportional to the number of elements in the vector ***P*** that gives the probabilities of the claim counts. The minimum number of claims for which there is a nonzero probability also has some effect on the computing time. But, due to the binary exponentiation tactic, the added computing time increases only as the logarithm to the base 2 of the minimum claim count.

Using APL 9 on a 386SX computer with 2 megabytes of RAM, this algorithm will run with k as high as 10. This makes the maximum length of certain vectors 2^{11}, or 2,048. Using APL 9, adding memory will not allow higher values of k because all arrays active at any given moment

must fit in the workspace available in the first 640K of memory (and this is about 400K because the APL system occupies about 200K).

It is likely that, compared to the computer programs presented in the appendices, the computations can be made more efficient in terms of the amount of memory used. References discuss computing the fast Fourier transform "in-place," which would use less memory than the programs given in the appendices. APL II, or other languages, might allow higher values of k due to better use of memory above the first 640K.

Increasing k by 1 roughly doubles the amount of memory needed, because the longest vectors double in length. Computational time is dominated by the time to compute the fast Fourier transforms, and this time increases by a factor of a bit more than 2 when k is increased by 1. See any of the references given above on the fast Fourier transform for a more precise discussion of the relationship between k and the time of computation.

To capture the distribution of the sum of i claims for any i with nonzero probability in the claim count distribution, just capture the ***PY*** for that i from Step 9 of the main algorithm, given above. When using the same severity distribution but differing claim count distributions to compute several aggregate distributions, the following method might save some time. Capture all the distributions of the sum of exactly i claims that will be needed (the ***PY***s above), and then just apply the probabilities given by the several claim count distributions and add. This can be much faster than recomputing each aggregate distribution from scratch.

Parameter Uncertainty

Patrik and John [17] distinguish *process risk* from *parameter risk* in estimating the distribution of final actual results relative to the estimated results. Essentially, if the frequency and severity distributions used are the best estimates of these distributions, then the calculated aggregate distribution reflects the inherent *process risk* or *process uncertainty*.

The extent to which the correct frequency and severity distributions are not known is termed *parameter risk* or *parameter uncertainty*. Some authors add *specification error* to the list of sources of potential difference

between actual and expected results. *Specification error* refers to the fact that the model being used might not be appropriate. For instance, if it is known that the claim count distribution is Poisson, but the parameter of the Poisson distribution is not known exactly, then estimates of the aggregate distribution are subject to parameter uncertainty. If it is not known whether the claim count distribution is Poisson or some other distribution, then estimates are subject to specification error. Heckman and Meyers [2] discuss incorporation of parameter uncertainty into estimates of aggregate loss distributions.

Parameter Uncertainty for the Claim Count Distribution

To reflect parameter uncertainty in the claim count distribution, one could proceed as follows. First, identify all claim count distributions that might apply, and assign to each claim count distribution the probability that it is the correct distribution. Then, for each claim count distribution (and using some severity distribution), compute the aggregate distribution. Finally, take the weighted average of all these aggregate distributions, according to the probabilities of the claim count distributions. The resulting aggregate distribution reflects the various claim count distributions and the probabilities of those distributions.

For example, one might estimate there is a 20% probability that the claim count distribution is Poisson with mean 10; there is a 50% probability that the claim count distribution is Poisson with mean 20; and there is a 30% probability that the claim count distribution is negative binomial with mean 15 and variance 30. (This is not necessarily a realistic example.) Then, for instance, the probability of total losses being less than $\$X$ in the combined aggregate distribution would be 20% of the probability of losses being less than $\$X$ in the aggregate distribution generated by the Poisson claim count distribution with mean 10, plus 50% of the corresponding probability resulting from the Poisson distribution with mean 20, plus 30% of the corresponding probability from the negative binomial distribution with mean 15 and variance 30.

Fortunately, there is a shortcut that makes it possible to compute the aggregate distribution that reflects the uncertainty regarding the claim count distribution without computing a great number of aggregate distri-

butions. Simply compute the weighted average of the claim count distributions, and then use this distribution in the computation of the aggregate loss distribution. For instance, using the above example, in the claim count distribution used as input to the main algorithm, the probability of i claims would be $0.2\times f(i) + 0.5\times g(i) + 0.3\times h(i)$, where f, g, and h are the probability density functions for the Poisson distribution with mean 10, the Poisson distribution with mean 20, and the negative binomial distribution with mean 15 and variance 30. This combined distribution would be used as the claim count distribution in the algorithm to compute the aggregate loss distribution.

Generally, one will select a family of claim count distributions, and associated probabilities, so that the mean of the combined claim count distribution will be the expected number of claims. The variance of the combined claim count distribution usually will be greater than the variance of the best estimate claim count distribution. The effect of the combined claim count distribution on the variance of the new aggregate distribution can be computed by using the formula for the variance of the aggregate distribution:

$$\mu_N \sigma_S^2 + \mu_S^2 \sigma_N^2 .$$

Here μ_N and σ_N^2 are the mean and variance of the claim count distribution and μ_S and σ_S^2 are the mean and variance of the severity distribution (Mayerson, Jones, and Bowers [18, p. 179]).

A particularly simple situation results if it assumed that the possible claim count distributions are Poisson and that the parameters of these Poisson distributions are distributed according to a gamma distribution with mean λ and variance σ^2. In this case, the resulting overall claim count distribution will be negative binomial with mean λ and variance $\lambda + \sigma^2$. This is discussed in Beard, Pentikäinen, and Pesonen [16, p. 40] and in Heckman and Meyers [2].

As a practical matter, one frequently has a binomial, Poisson, or negative binomial distribution as the best estimate of the claim count distribution. To reflect parameter uncertainty in the claim count distribution used as input to the aggregate loss distribution algorithm, one might, where appropriate, simply use a claim count distribution with the same mean

and a larger variance than the best estimate distribution. For instance, if one's best estimate of the claim count distribution is Poisson with parameter λ then, to reflect parameter uncertainty, one might use a negative binomial distribution with mean λ and variance larger than λ.

The three families of claim count distributions mentioned above are related. For the Poisson distribution, the variance and the mean are the same. The negative binomial has variance greater than the mean. The binomial has variance less than the mean. The Poisson is a limiting case of the negative binomial in that, as the variance of the negative binomial approaches the mean, the negative binomial approaches the Poisson. The Poisson is also a limiting case of the binomial.

In conclusion, parameter uncertainty for the claim count distribution can often be reflected in the computed aggregate loss distribution by choosing an appropriate claim count distribution with the same mean and larger variance than the best estimate distribution. This allows one to reflect parameter uncertainty while computing only one aggregate loss distribution.

Parameter Uncertainty for the Severity Distribution

To reflect parameter uncertainty for the severity distribution, one can proceed in the manner first discussed for the claim count distribution. That is, delineate all the severity distributions that might apply; assign to each a probability; compute the aggregate distribution using each severity distribution; and combine all of these aggregate distributions according to the probabilities of the severity distributions.

Unfortunately, when estimating the effect on aggregate distributions of parameter uncertainty in the severity distribution there is no shortcut quite as efficient as the one for claim count distributions. That is, to reflect parameter uncertainty for the severity distribution, it is not sufficient to use a severity distribution that is the combination of the various severity distributions in the same way that it is possible to use a claim count distribution that is the combination of the several claim count distributions. Later, it will be shown why this last statement is true, but methods of reflecting parameter uncertainty for the severity distribution will be covered first.

Some simplification in the reflection of parameter uncertainty for the severity distribution results if all the possible severity distributions are (or are assumed to be) multiples of some base distribution. More precisely, this assumption is that if F_B is the distribution function of the base severity distribution, B, and if F_Y is the distribution function of any other severity distribution in the family, Y, then there is a constant, c, such that $F_B(cx) = F_Y(x)$. Normally, this is written $B = cY$; it follows that $E(Y) = (1/c)\,E(B)$, and $\text{Var}(Y) = (1/c^2)\,\text{Var}(B)$. Let the constants c be distributed according to a probability distribution with distribution function H and density function h.

Let F_A be the distribution function of the aggregate distribution computed using the base severity distribution B (corresponding to F_B). Then the aggregate distribution reflecting parameter uncertainty, T, is given by

$$T(x) = \int_0^\infty F_A(cx)\, h(c)\, dc.$$

If h has a form such that $h(t)$ and $th(t)$ are easily integrated over arbitrary intervals, and if F_A is piecewise linear, then T, above, is easily computed. For t in the interval $[l_i, u_i]$ let $F_A(t) = a_i + b_i t$. Then

$$\int_0^\infty F_A(cx)\, h(c)\, dc = \sum_{i=0}^{\infty} \left[\int_{l_i/x}^{u_i/x} (a_i + b_i cx)\, h(c)\, dc \right]$$

$$= \sum_{i=0}^{\infty} \left[a_i \int_{l_i/x}^{u_i/x} h(c)\, dc + b_i x \int_{l_i/x}^{u_i/x} ch(c)\, dc \right].$$

As a practical matter, the sums above are not taken to infinity, but rather to a high enough value that sufficient accuracy is achieved. If h has been chosen so that the integrals are easy to compute, T is also easy to compute.

Next is the demonstration, promised above, that to reflect the effect of parameter uncertainty in the severity distribution, it is not sufficient to simply use a severity distribution with a larger variance. To see this,

consider one way a simulation model could be used to estimate the aggregate distribution. Choose a claim count, n, at random from the claim count distribution, N. Then n times draw a random claim severity, s_i, from the severity distribution, S. Compute $s_1 + s_2 + ... + s_n$. This sum gives one "draw" from the aggregate distribution; that is, it gives one observation selected at random from the aggregate distribution. Repeat this process, i.e., make draws from the aggregate distribution, until the statistics of interest for the aggregate distribution are known with sufficient accuracy.

There are two methods one might use to reflect parameter uncertainty for the severity distribution when performing the above simulation. The first method is to choose a severity distribution at random each time a severity is needed. Within a given draw, s_{i+1} would potentially be drawn from a different distribution than the preceding s_i. A second method is to fix a severity distribution each time an n is chosen from N. This one severity distribution is used for all s_i in the sum $s_1 + s_2 + ... + s_n$ corresponding to one draw. Then another n is selected from N, and another severity distribution, possibly different from the severity distribution used in the previous draw, is used, and the process continues.

It is the second method that best reflects parameter uncertainty for the severity distribution. Under this method, only one severity distribution is used for each draw from the aggregate distribution. In contrast, under the first method, in many of the draws from the aggregate distribution, some claim amounts will come from severity distributions with larger-than-average means and some claim amounts will come from severity distributions with smaller-than-average means. The effects of severity distributions with larger-than-average means and severity distributions with smaller-than-average means will tend to cancel each other to some degree. Thus, the first method will tend to produce an aggregate distribution with a smaller variance than is correct. Under the second method, each draw is influenced by only one severity distribution.

The first simulation method corresponds to using a severity distribution that is the composite of the family of severity distributions being used to reflect parameter uncertainty. It is the second simulation method, where each draw is influenced by only one severity distribution, that

corresponds to the methods discussed above for reflecting parameter uncertainty for the severity distribution.

The first and second methods differ fundamentally in the independence assumptions among samples from the severity distribution. A more mathematical discussion of the differences between the two methods, including a more precise discussion of the difference in independence assumptions, is given in Appendix I.

Capped Severity Distributions and Parameter Uncertainty

If a capped severity distribution is being used, e.g., losses are capped at $250,000 per claim, and if parameter uncertainty for the severity distribution is reflected using the method that assumes that all distributions are multiples of each other, then the loss cap becomes variable. In some cases, e.g., where the aggregate distribution of a self-insurance program with a given retention is being computed, it may not be appropriate to allow the loss cap to vary. There does not seem to be a simple way to reflect parameter uncertainty for the severity distribution in such a case.

One approach is to increase the degree of parameter uncertainty reflected in the claim count distribution to a level above that which would otherwise be used, and to not reflect parameter uncertainty in the severity distribution. This approach is not theoretically correct, but, as a practical matter, might be sufficiently accurate. Another approach is to let the cap be essentially variable, and perform tests to determine whether this significantly distorts the results. Finally, and with the greatest accuracy, one can compute a number of aggregate distributions, each using a different severity distribution with the correct cap, and take the weighted average.

Excess Loss Distributions

The results of this subsection apply to all methods used to calculate aggregate loss distributions, not to just the algorithm presented herein. The main results of this subsection appear to be well known, but have not previously appeared directly in actuarial literature. Schumi [21, 22] has presented material similar to this result. Bear and Nemlick [23] present the result in terms of the negative binomial distribution.

Suppose a claim count distribution, a severity distribution, and the corresponding aggregate distribution are specified. In regard to the severity distribution, suppose further that the probability of any given claim being excess of a given attachment point A is α. Suppose it is desired to compute the aggregate distribution for claims excess of A (this A has nothing to do with the vector of spreads $\boldsymbol{A}$ used previously). For example, the aggregate distribution might be based on a Poisson claim count distribution with parameter 1,000 (i.e., the number of expected claims is 1,000) and a Weibull severity distribution with mean \$10,000 and coefficient of variation 8. If A is \$100,000 then α is 0.0197.

One way to compute the excess aggregate distribution (the aggregate distribution for the amount of claims excess of A per claim) is to keep the same claim count distribution (e.g., Poisson with parameter 1,000 in the example) and adjust the severity distribution so that claims less than A become 0 and claims, x, greater than or equal to A become $x - A$. This gives a severity distribution that generally assigns a large probability, namely $1 - \alpha$, to claims being exactly 0.

Another way is to work directly with the excess claim count and severity distributions. The excess claim count distribution is the distribution of the number of excess claims (the distribution of the number of claims exceeding A). The excess severity distribution is the claim severity distribution for the amount of individual claims excess of A, given that a claim is excess.

The main purpose of this subsection is to note that, for certain claim count distributions, the excess claim count distribution is easily determined. Assume the claim count distribution is binomial, Poisson, or negative binomial, respectively, with mean λ and variance σ^2. Suppose the probability of a given claim being excess of the attachment point A is α. Then the excess claim count distribution is binomial, Poisson, or negative binomial, respectively, with mean $\alpha\lambda$ and variance $\alpha\lambda + \alpha^2(\sigma^2 - \lambda)$.

The excess severity distribution is easy to determine if the total distribution and the attachment point are known. Suppose the severity distribution has distribution function F, and the attachment point for excess

claims is A. Then the excess severity distribution has distribution function, H, defined by:

$$H(x) = \frac{F(x + A) - F(A)}{1 - F(A)}, \text{ for } x \geq 0.$$

That is, the portion of the severity distribution function below A is eliminated, and the remaining distribution is rescaled so that $H(0)$ is 0 and $H(x)$ has limit 1 as x tends to infinity.

In the example, the excess claim count distribution is Poisson with parameter 19.7 (= 0.0197 × 1,000). The excess severity distribution is the above Weibull distribution restricted to claims exceeding $100,000. In particular, the probability of a claim being 0 is 0 (not 0.9803).

It should be clear that the excess severity distribution is as claimed above. Appendix J has a proof that the excess claim count distribution is as claimed. An interesting open problem is to find other claim count distributions for which the excess claim count distribution is of the same form as the original claim count distribution, or the excess claim count distribution is otherwise easy to compute.

For readers familiar with the notation in Heckman and Meyers, recall that they parameterize claim count distributions with λ and c. In their method, the parameters for the excess claim count distribution are $\alpha\lambda$ and c.

The above formulae for the mean and variance of the excess claim count distribution hold only if the parameters of the severity distribution are known with certainty. Venter provided the following formulas for the mean and variance of the excess claim count distribution N when α is uncertainly known:

$$E(N) = \lambda E(\alpha),$$

$$\text{Var}(N) = \lambda E(\alpha) + (\sigma^2 - \lambda)E(\alpha)^2 + (\sigma^2 + \lambda^2 - \lambda)\text{Var}(\alpha).$$

Proofs are as follows:

$$E(N \mid \alpha) = \alpha\lambda, \text{ so } E(N) = E(E(N \mid \alpha)) = \lambda E(\alpha).$$

$$\begin{aligned}
\text{Var}(N \mid \alpha) &= \alpha\lambda + \alpha^2(\sigma^2 - \lambda)\text{, so} \\
\text{Var}(N) &= \text{E}(\text{Var}(N \mid \alpha)) + \text{Var}(\text{E}(N \mid \alpha)) \\
&= \lambda\text{E}(\alpha) + (\sigma^2 - \lambda)\text{E}(\alpha^2) + \lambda^2\text{Var}(\alpha) \\
&= \lambda\text{E}(\alpha) + (\sigma^2 - \lambda)[\text{Var}(\alpha) + \text{E}(\alpha)^2] + \lambda^2\text{Var}(\alpha) \\
&= \lambda\text{E}(\alpha) + (\sigma^2 - \lambda)\text{E}(\alpha)^2 + (\sigma^2 + \lambda^2 - \lambda)\text{Var}(\alpha)\text{.}
\end{aligned}$$

These formulae are useful either if α varies from one claim to the next (for example, if the excess distribution is for a set of reinsurance contracts with attachment points that vary by contract), or if it is desired to reflect parameter uncertainty with regard to α.

REFERENCES

[1] Halmos, Paul R., "Has Mathematical Progress Slowed Down?" *American Mathematical Monthly*, August-September 1990,Vol. 97, No. 7, pp. 561-588.

[2] Heckman, Philip E., and Meyers, Glenn G., "The Calculation of Aggregate Loss Distributions from Claim Severity and Claim Count Distributions," *PCAS* LXX, 1983, pp. 22-61.

[3] Baase, Sara, *Computer Algorithms* (Second Edition), Addison-Wesley, 1988.

[4] Aho, Alfred V., Hopcraft, John E., and Ullman, Jeffrey D., *The Design and Analysis of Computer Programs*, Addison-Wesley, 1974.

[5] Knuth, Donald E., *The Art of Computer Programming*, Addison-Wesley; Vol. 1, *Fundamental Algorithms*, Second Edition, 1973; Vol. 2, *Seminumerical Algorithms*, Second Edition, 1981.

[6] Press, William H., Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T., *Numerical Recipes*, Cambridge University Press, 1986.

[7] Preparata, Franco P., "Computational Complexity," *Studies in Computer Science*, Pollack, S. V. (ed.), MAA Studies in Mathematics, The Mathematical Association of America, 1982, Vol. 22.

[8] Chiu, Patrick, "Transforms, Finite Fields, and Fast Multiplication," *Mathematics Magazine*, December 1990, Vol. 63, No. 5, pp. 330-336.

[9] Monro, Donald M., "Complex Discrete Fast Fourier Transform," *Applied Statistics*, Journal of the Royal Statistical Society (Series C), 1975, Vol. 24, No. 1, pp. 153-160.

[10]Hogg, Robert V., and Klugman, Stuart A., *Loss Distributions*, John Wiley & Sons, Inc., 1984.

[11]Feller, William, *An Introduction to Probability Theory and Its Applications* (Second Edition), John Wiley & Sons, Inc., 1971, Vol. II.

[12]Borch, Karl, *The Mathematical Theory of Insurance*, Lexington Books, 1974.

[13]Bertram, Jürgen, "Numerische Berechnung von Gesamtschadenverteilungen," *Blätter der deutschen Gesellschaft Versicherungsmathematik*, October 1981, Band 15.2, pp. 175-194.

[14]Bühlmann, Hans, "Numerical Evaluation of the Compound Poisson Distribution: Recursion or Fast Fourier Transform?" *Scandinavian Actuarial Journal*, 1984, No. 2, pp. 116-126.

[15]Bühlmann, Hans, *Mathematical Methods in Risk Theory*, Springer-Verlag, 1970.

[16]Beard, R.E., Pentikäinen, T., and Pesonen, E., *Risk Theory* (Third Edition), Chapman and Hall, 1984.

[17]Patrik, Gary, and John, Russell, "Pricing Excess-of-Loss Casualty Working Cover Reinsurance Treaties," *Pricing Property and Casualty Insurance Products*, Casualty Actuarial Society Discussion Paper Program, 1980, pp. 399-474.

[18]Mayerson, Allen L., Jones, Donald A., and Bowers, Newton L., Jr., "On the Credibility of the Pure Premium," *PCAS* LV, 1968, pp. 175-185.

[19]Graham, Ronald L., Knuth, Donald E., and Patashnik, Oren, *Concrete Mathematics*, Addison-Wesley, 1989.

[20]Venter, Gary G., "Easier Algorithms for Aggregate Excess," *Casualty Actuarial Society Forum*, Fall 1989, pp. 19-37.

[21]Schumi, Joseph R., "A Method to Calculate Aggregate Excess Loss Distributions," *Casualty Actuarial Society Forum*, Spring 1989, pp. 193-203.

[22]Schumi, Joseph R., "Excess Loss Distributions Over an Underlying Annual Aggregate," *Casualty Actuarial Society Forum*, Fall 1989, pp. 159-172.

[23]Bear, Robert A., and Nemlick, Kenneth J., "Pricing the Impact of Adjustable Features and Loss Sharing Provisions of Reinsurance Treaties," *PCAS* LXXVII, 1990, pp. 60-123.

[24]Hastings, N.A.J., and Peacock, J.B., *Statistical Distributions*, John Wiley & Sons, Inc., 1975.

APPENDIX A

COMPLEX NUMBERS

This is a brief summary of the properties of complex numbers used earlier. More extensive treatments are in Baase [3, p.279], and Aho, Hopcraft, and Ullman [4, p. 252].

In this Appendix, i is $\sqrt{-1}$. Given two complex numbers, $a + bi$ and $c + di$, their sum, difference, product, and quotient are given as:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

$$(a + bi) - (c + di) = (a - c) + (b - d)i$$

$$(a + bi) \times (c + di) = (ac - bd) + (ad + bc)i$$

$$(a + bi) \div (c + di) = \left(\frac{1}{c^2 + d^2} \right) [(ac + bd) - (ad - bc)i\ ]$$

The *complex conjugate* of $a + bi$ is $a - bi$, sometimes denoted $(a + bi)^*$.

A number ω is a *primitive n^{th} root of unity* if $\omega^n = 1$ and $\omega^j \neq 1$ for any positive j less than n. If ω is a primitive n^{th} root of unity, then:

$$\omega^{j*} = 1 \div \omega^j$$

$$\sum_{k=0}^{n-1} (\omega^j)^k = \begin{cases} 0 & \text{for } j \text{ not a multiple of } n \\ n & \text{for } j \text{ a multiple of } n \end{cases}$$

Let $\boldsymbol{F}$ be the $n \times n$ *matrix with entries* $\boldsymbol{F}_{jk} = \omega^{jk}$. This matrix $\boldsymbol{F}$ plays a key role in the discrete Fourier transform (DFT). $\boldsymbol{F}^2$ is:

$$\begin{bmatrix} n & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & & 0 & n \\ 0 & 0 & 0 & & n & 0 \\ \cdot & & & \cdot & & \cdot \\ \cdot & & & \cdot & & \cdot \\ \cdot & & & \cdot & & \cdot \\ 0 & 0 & n & & 0 & 0 \\ 0 & n & 0 & \dots & 0 & 0 \end{bmatrix}$$

Appendix B shows why this makes the inverse of the DFT so simple.

APPENDIX B

CONVOLUTION EXAMPLE

This Appendix provides an example of the use of the convolution theorem to compute the sum of two severity distributions. Two vectors are used to represent severity distributions, and the convolution of these vectors represents the sum of the two severity distributions.

Example

The first severity distribution, $\boldsymbol{U}$, has probability 4/5 of a claim amount of \$0, probability 1/10 of a claim amount of \$100, probability 1/20 of a claim amount of \$200, and probability 1/20 of a claim amount of \$300. The second severity distribution, $\boldsymbol{V}$, has probability 3/5 of a claim amount of \$0, probability 1/5 of a claim amount of \$100, probability 1/10 of a claim amount of \$200, and probability 1/10 of a claim amount of \$300. These are represented as vectors as follows:

$$\boldsymbol{U} = [4/5,\ 1/10,\ 1/20,\ 1/20,\ 0,\ 0,\ 0,\ 0],$$

$$\boldsymbol{V} = [3/5,\ 1/5,\ 1/10,\ 1/10,\ 0,\ 0,\ 0,\ 0]\ .$$

These representations have been padded with zeroes to the right so that no-wrap convolutions can be computed. (They are not what is used in the body of the paper for the main algorithm. These representations are being used only to give an example of the use of the convolution theorem.)

As $\boldsymbol{U}$ and $\boldsymbol{V}$ are vectors of length 8, ω must be a primitive eighth root of unity. Let ω be $\cos(\pi/4) + i \sin(\pi/4)$. This ω can also be written $\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2} i$ or, approximately, 0.7071067810 + 0.7071067810i. (Here i is $\sqrt{-1}$.) The matrix $\boldsymbol{F}$, with entries ω^{jk} for j, k from 0 to 7, is:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \%1 & i & \%2 & -1 & \%3 & -i & \%4 \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & \%2 & -i & \%1 & -1 & \%4 & i & \%3 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \%3 & i & \%4 & -1 & \%1 & -i & \%2 \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & \%4 & -i & \%3 & -1 & \%2 & i & \%1 \end{bmatrix}$$

where %1 is $\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i$, %2 is $-\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}i$, %3 is $-\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$, and %4 is $\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}i$. These are ω, ω^3, ω^5, and ω^7, respectively.

The convolution theorem states that

$$\boldsymbol{U}*\boldsymbol{V} = \text{INVDFT}(\text{DFT}(\boldsymbol{U}) \times \text{DFT}(\boldsymbol{V}))$$

where DFT is the discrete Fourier transform, and INVDFT is the inverse DFT.

The discrete Fourier transform of $\boldsymbol{U}$ is the matrix product $\boldsymbol{F}\cdot\boldsymbol{U}$ where $\boldsymbol{U}$ is treated as a column vector. Thus DFT($\boldsymbol{U}$), or $\boldsymbol{F}\cdot\boldsymbol{U}$, is approximately:

[1.0, .8353553391 + .1560660172i, .7500000000 + .0500000000i, .7646446610 + .0560660172i, .7000000000, .7646446610 − .0560660172i, .7500000000 − .0500000000i, .8353553391 − .1560660172i] .

Similarly, DFT($\boldsymbol{V}$), or $\boldsymbol{F}\cdot\boldsymbol{V}$, is approximately:

[1.0, .6707106781 + .3121320343i, .5000000000 + .1000000000i, .5292893219 + .1121320343i, .4000000000, .5292893219 -.1121320343i, .5000000000 - .1000000000i, .6707106781 - .3121320343i].

DFT($\boldsymbol{U}$) × DFT($\boldsymbol{V}$) is

[1.0, .5115685425 + .3654163056i, .3700000000 + .1000000000i, .3984314575 + .1154163056i, .2800000000, .3984314575 -.1154163056i, .3700000000 - .1000000000i, .5115685425 - .3654163056i].

For example, the second element of the vector just above is .5115685425 + .3654163056i, which is

$$(.8353553391 + .1560660172i) \times (.6707106781 + .3121320343i).$$

To compute the inverse DFT of DFT($\boldsymbol{U}$) × DFT($\boldsymbol{V}$), one first computes the DFT of DFT($\boldsymbol{U}$) × DFT($\boldsymbol{V}$); divides each term of the result by 8; and inverts the order of the last seven terms. The DFT of DFT($\boldsymbol{U}$) × DFT($\boldsymbol{V}$), or $\boldsymbol{F}\cdot$ (DFT($\boldsymbol{U}$) × DFT($\boldsymbol{V}$)) , is

[3.840000000, 0.0, .0400000000, .0800000000, .2000000000, 1.040000000, 1.040000000, 1.760000000].

Dividing by 8 gives

[0.480, 0.0, 0.005, 0.010, 0.025, 0.130, 0.130, 0.220].

Reversing the order of the last 7 terms gives

[0.480, 0.220, 0.130, 0.130, 0.025, 0.010, 0.005, 0.0].

Thus, for the sum of the distributions $\boldsymbol{U}$ and $\boldsymbol{V}$ there is a probability of 0.480 of a total claim amount of \$0, a probability of 0.220 of a total claim amount of \$100, a probability of 0.130 of a total claim amount of \$200, etc. This can be readily verified by direct computation of these probabilities.

The Inverse DFT

This subsection will justify the method used above to compute the inverse DFT. Suppose we have computed DFT($\boldsymbol{W}$), which is $\boldsymbol{F}\cdot\boldsymbol{W}$, for some vector $\boldsymbol{W}$. Then DFT(DFT($\boldsymbol{W}$)) is $\boldsymbol{F}\cdot$ ($\boldsymbol{F}\cdot\boldsymbol{W}$). Matrix multiplication is

associative, so this is the same as $(\boldsymbol{F} \cdot \boldsymbol{F}) \cdot \boldsymbol{W}$ or $\boldsymbol{F}^2 \cdot \boldsymbol{W}$. But $\boldsymbol{F}^2$ (for the example above) is

$$\begin{bmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This is just 8 times the matrix $\boldsymbol{R}$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This matrix reverses the order of the last seven elements of any vector to which it is applied.

Thus $\boldsymbol{F}^2 \cdot \boldsymbol{W}$ is just 8 times $\boldsymbol{W}$ with the last seven terms reversed. Dividing by 8 and reversing the last seven terms restores $\boldsymbol{W}$.

Alternatively, $\boldsymbol{F}^{-1}$ is just $(1/8) \times \boldsymbol{R} \times \boldsymbol{F}$, where $\boldsymbol{R}$ is the matrix of 1s and 0s above.

APPENDIX C

THE FAST FOURIER TRANSFORM AND INVERSE

The fast Fourier transform is presented by Baase [3, p. 273] as follows[2]. Using a language similar to Modula-2 and Pascal:

Input: The n-vector $\boldsymbol{P} = (p_0, p_1, ..., p_{n-1})$, where $n = 2^k$ for some $k > 0$.

Output: *transform*, the discrete Fourier transform of $\boldsymbol{P}$.

Comment: We assume that *omega* is an array containing the nth roots of 1: $\omega^0, \omega, ..., \omega^{(n/2)-1}$. π_k is a permutation on $\{0, 1, ..., n-1\}$ (described below).

```
procedure FFT (P: RealArray; n: integer;
        var transform: ComplexArray);

var     l: integer;     {the level number}
        num: integer;   {the number of values to be computed at
                         each node at level l }
        t: integer;     {the index in transform for the first of
                         these values for a particular node}
        j: integer;     {counts off the pairs of values to be
                         computed for that node }
        m: integer;     {used to pick out the correct entry from
                         omega }
begin
        for t := 0 to n - 2 by 2 do
                transform [t] := p[π_k(t)] + p[π_k(t + 1)];
                transform [t + 1] := p[π_k(t)] - p[π_k(t + 1)]
end     {for t };

        {The main computation}
          m := n/2; num := 2;
        {Begin triply-nested loop}
        for l := k-2 to 0 by -1 do
```

[2] Reprinted with permission of the publisher.

```
        m := m/2; num := 2*num;
      for t := 0 to (2^l - 1)num by num do
      for j := 0 to (num/2)-1 do
          xPOdd := omega[mj]*transform[t+num/2+j];
          transform[t + num/2 + j] := transform[t + j] - xPOdd;
          transform[t + j] := transform[t + j] + xPOdd;
        end { for j }
        end { for t }
      {end of body of outer for loop }
      end { for l }
end { FFT }
```

Now, what is π_k? Let t be an integer between 0 and $n-1$, where $n = 2^k$. Then t can be represented in binary by $[b_0\, b_1 ... b_{k-1}]$, where each b_j is 0 or 1. Let $rev_k(t)$ be the number represented by these bits in reverse order, i.e., by $[b_{k-1} ... b_1\, b_0]$. Then $\pi_k(t) = rev_k(t)$. (As an example, $\pi_3(3) = 6$ because 011 reversed is 110.)

The inverse fast Fourier transform is computed as follows. Apply the regular (forward) fast Fourier transform to the vector. Divide each element of the result by n. Reverse the order of the last $n-1$ elements (i.e., the first element stays in place and the order of the other elements is reversed).

Baase gives further discussion of the fast Fourier transform, including some analysis of the number of computations needed.

APPENDIX D

APL PROGRAMS FOR FFT AND INVFFT ALGORITHMS

This Appendix contains the functions FFT and INVFFT. Before running these, INIT must be run to initialize certain global variables. FFT and INVFFT do not modify these global variables, so INIT needs to be run only when the global variables have to be changed. INIT calls INITOMEGA and INITPIK, also listed here.

All the APL functions presented in this paper assume □IO has been set to zero.

Complex vectors of length n are represented as two-by-n arrays; e.g., the vector $(a + bi, c + di, ...)$ is represented by:

$$\begin{pmatrix} a & c & \dots \\ b & d & \dots \end{pmatrix}$$

```
FFT:

 [0]  R←FFT P;Z;INDX;INDXP1;T1;T2;M;NUM;L;
        OMIND
 [1]  ⍝ REFERENCE SARA BAASE, COMPUTER
        ALGORITHMS, P 273 FF
 [2]  ⍝ ASSUMES YOU HAVE RUN INIT
 [3]  ⍝ INPUT IS P PER BAASE, N PER BAASE IS
        GLOBAL VARIABLE
 [4]  ⍝ OUTPUT IS TRANSFORM PER BAASE
 [5]  R←INITR
 [6]  R[;INDXE]←P[;PIK[INDXE]]+P[;PIK[INDXO]]
 [7]  R[;INDXO]←P[;PIK[INDXE]]-P[;PIK[INDXO]]
 [8]  ⍝ NOW HAVE INITIALIZED R (= TRANSFORM)
 [9]  M←⌊0.5+N÷2 ◇ NUM←2 ◇ L←K-2
[10]  LLOOP:M←⌊0.5+M÷2 ◇ NUM←⌊0.5+2×NUM
[11]  T2←,⍉(⍳NUM÷2)∘.+NUM×⍳N÷NUM
[12]  T1←T2+NUM÷2
[13]  OMIND←(N÷2)⍴M×⍳N÷2×M
[14]  Z←(2,N÷2)⍴(,-⌿OMEGA[;OMIND]×R[;T1]),
        ,+⌿OMEGA[;OMIND]×⊖R[;T1]
[15]  R[;T1]←R[;T2]-Z ◇ R[;T2]←R[;T2]+Z
[16]  L←L-1
[17]  →(L≥0)⍴LLOOP


INVFFT:

 [0]  R←INVFFT X
 [1]  R←FFT X
 [2]  R[;INVINDX]←⌽R[;INVINDX] ⍝ REVERSE
        ORDER OF LAST N-1 ELEMENTS
 [3]  R←R÷N
```

```
INIT:

[0]  INIT KK
[1]  K←KK
[2]  N←⌊0.5+2*K
[3]  ⎕IO←0
[4]  INITOMEGA K
[5]  INITPIK K
[6]  INITR←(2,N)⍴0
[7]  INDXE←2×⍳N÷2
[8]  INDXO←INDXE+1
[9]  INVINDX←1+⍳N-1
[10] TAIL←(N÷2)+⍳N÷2

INITOMEGA:

[0]  INITOMEGA K
[1]  N←⌊0.5+2*K
[2]  OMEGA←(2,N÷2)⍴○(⍳N÷2)×2÷N
[3]  OMEGA[0;]←2○OMEGA[0;]
[4]  OMEGA[1;]←1○OMEGA[1;]
[5]  OMEGA2←(2,N+1)⍴○(⍳N+1)×2÷2×N
[6]  OMEGA2[0;]←2○OMEGA2[0;]
[7]  OMEGA2[1;]←1○OMEGA2[1;]

INITPIK:

[0]  INITPIK K;N
[1]  N←⌊0.5+2*K
[2]  PIK←2⊥⍉((K⍴2)⊤⍳N)
```

APPENDIX E

PROOF THAT $A^i * S^i$ IS THE DISTRIBUTION OF i CLAIMS

This Appendix gives a proof sketch that $A^i * S^i$ is the probability distribution of the sum of exactly i claims. More precisely, it shows that if A^i, S^i, n, and L are defined as in the main body of this paper, and X is the probability distribution of exactly i claims, then the probability that X is between $j \times \left(\frac{L}{n}\right)$ and $(j+1) \times \left(\frac{L}{n}\right)$ is $(A^i * S^i)[j]$.

Consider first the case where $L = n$ and $S = (1, 0, 0, ..., 0)$. This makes S a uniform distribution on the unit interval [0, 1]. In this case the first n terms of $A^i * S^i$ are:

$$a_0^i, a_1^i, a_2^i, \ldots a_{n-1}^i .$$

Let F^i be the distribution function for the sum of i mutually independent random variables uniformly distributed over [0, 1]. Let $b_j^i = F^i(j+1) - F^i(j)$. It needs to shown that $a_j^i = b_j^i$.

Combining equation 9.1 and Theorem 1 of Section I.9 of Feller [11, p. 27],

$$b_j^i = F^i(j+1) - F^i(j) = \frac{1}{i!} \sum_{v=0}^{i+1} (-1)^v \binom{i+1}{v} (j+1-v)_+^i ,$$

where x_+^i is x^i if $x \geq 0$ and x_+^i is zero if $x < 0$. This yields:

$$b_j^i = \frac{1}{i!} \sum_{v=0}^{j} (-1)^v \binom{i+1}{v} (j+1-v)^i$$

because if $j < i+1$ then $(j+1-v)_+$ is zero for terms with $v > j$, and if $j > i+1$ then $\binom{i+1}{v}$ is zero for $v > i+1$.

It is easy to see that $a_0^i = b_0^i = 1/i!$ for $i > 1$, and $a_j^1 = b_j^1 = 0$ for $j \geq 1$. To complete the proof that $a_j^i = b_j^i$ it suffices to show that the b_j^i satisfy the recursion relation used to define the a_j^i.

To this end, for $i > 1$ and $j > 0$ consider:

$$z = \left(\frac{1}{i} \right) \left[(i-j)\, b_{j-1}^{i-1} + (j+1)\, b_j^{i-1} \right].$$

It needs to be shown that z equals b_j^i. This is done by plugging into the above formula the expression for b_j^i as a sum, and rearranging terms:

$$i!\, z = (i-j)\,(i-1)!\, b_{j-1}^{i-1} + (j+1)\,(i-1)!\, b_j^{i-1}$$

$$= (i-j) \sum_{\tau=0}^{j-1} (-1)^\tau \binom{i}{\tau} (j-\tau)^{i-1} + (j+1) \sum_{v=0}^{j} (-1)^v \binom{i}{v} (j+1-v)^{i-1}$$

$$= (i-j) \sum_{v=0}^{j} (-1)^{v-1} \binom{i}{v-1} (j+1-v)^{i-1} +$$

$$(j+1) \sum_{v=0}^{j} (-1)^v \binom{i}{v} (j+1-v)^{i-1}$$

$$= (j+1)\,(1)\,(1)\,(j+1)^{i-1} +$$

$$\sum_{v=1}^{j} \left[(-1)\,(i-j) \binom{i}{v-1} + (j+1) \binom{i}{v} \right] (-1)^v\, (j+1-v)^{i-1}$$

$$= (-1)^0 \binom{i+1}{0} (j+1-0)^i +$$

$$\sum_{v=1}^{j} \left[\frac{(j-1)\, v}{i+1} + \frac{(j+1)\,(i+1-v)}{i+1} \right] \left(\frac{(i+1)!}{v!\,(i+1-v)!} \right) (-1)^v\, (j+1-v)^{i-1}$$

$$= (-1)^0 \binom{i+1}{0} (j+1-0)^i + \sum_{v=1}^{j} (j+1-v) \binom{i+1}{v} (-1)^v (j+1-v)^{i-1}$$

$$= \sum_{v=0}^{j} (-1)^v \binom{i+1}{v} (j+1-v)^i .$$

Thus, $z = b_j^i$. This establishes the main result for this case.

For the general case, consider the positive "quadrant" of $\mathbb{R}^i$, i.e, the points $(x_0, x_1, ..., x_{i-1})$ such that each x_j is greater than or equal to zero. Divide this space into cubes with edge length $L \div n$ in the obvious way. Assign a density to each cube as follows. If the cube's vertex closest to the origin is $\left(v_0 \frac{L}{n}, v_1 \frac{L}{n}, ..., v_{i-1} \frac{L}{n} \right)$, assign a density of $(s_{v_0} s_{v_1} s_{v_2} \dots s_{v_{i-1}}) \div \left(\frac{L}{n} \right)^i$ where the s_{v_k} are elements of the vector representing the severity distribution if $v_k \le n-1$, and $s_{v_k} = 0$ if $v_k \ge n$. As the volume of every cube is $\left(\frac{L}{n} \right)^i$, the integral of this density over the cube is $s_{v_0} s_{v_1} s_{v_2} \dots s_{v_{i-1}}$. Now consider the integral of these densities between the parallel $(i-1)$-planes:

$$x_0 + x_1 + \dots + x_{i-1} = k \frac{L}{n}, \text{ and}$$

$$x_0 + x_1 + \dots + x_{i-1} = (k+1) \frac{L}{n} .$$

This integral is the probability that the sum of i claims will have a value between $k \frac{L}{n}$ and $(k+1) \frac{L}{n}$.

This is also the k^{th} term of $\boldsymbol{A}^i * \boldsymbol{S}^i$, as will now be shown. The m^{th} element of $\boldsymbol{S}^i$ is the sum of all $s_{j_0} \times s_{j_1} \times \ldots \times s_{j_{i-1}}$ such that $j_0 + j_1 + \ldots + j_{i-1} = m$. For instance, if i is 3, s_2^3 is:

$$s_2 s_0 s_0 + s_0 s_2 s_0 + s_0 s_0 s_2 + s_1 s_1 s_0 + s_1 s_0 s_1 + s_0 s_1 s_1 = 3 s_0^2 s_2 + 3 s_0 s_1^2.$$

Each of the cubes associated with the m^{th} element of $\boldsymbol{S}^i$ (under the inverse of the above association of cubes with densities) has its vertex closest to the origin on the plane $x_0 + \ldots + x_{i-1} = m\frac{L}{n}$. Each cube also has its vertex farthest from the origin on the plane $(m+i)\frac{L}{n}$. The planes "m," "$m+1$," ..., "$m+i$" divide each of these cubes into the proportions given by the $\boldsymbol{A}^i$. Getting back to the planes "k" and "k+1", considering all the cubes that have some portion between these two planes, the integral of the density between these two planes is $(\boldsymbol{A}^i * \boldsymbol{S}^i)[k]$. The probability that the sum of the i distributions will be between k and $k+1$ is given by the same integral. This establishes the overall result.

APPENDIX F

PACKING AND UNPACKING

This Appendix provides the formulae for packing and unpacking transformed vectors. This treatment essentially follows that of Press, Flannery, Teukolsky, and Vetterling [6, p. 398]. APL programs to implement these routines are in Appendix G.

Unpacking a transformed vector is discussed first. Assume that one starts with a real (untransformed) vector, $\boldsymbol{U}$, of length $2n$ and packs it into a complex vector, $\boldsymbol{PU}$, of length n, as discussed in the main body of the paper. Then the FFT is applied to $\boldsymbol{PU}$ to obtain a vector $\boldsymbol{PH}$ that is the packed transformation of $\boldsymbol{PU}$. The next step is to unpack $\boldsymbol{PH}$ to obtain the FFT of $\boldsymbol{U}$.

The result of unpacking $\boldsymbol{PH}$ will be a complex vector of length $n + 1$. One might think the result would be a complex vector of length $2n$ since the goal is to obtain the FFT of $\boldsymbol{U}$ which is of length $2n$. If $\boldsymbol{R}$ is the (length $2n$ for the moment) FFT of $\boldsymbol{U}$, then:

$$r_{2n-j} = r_j^* \text{ for } 1 \le j \le n ,$$

where * denotes complex conjugation. Thus, from the first $n + 1$ terms (0 to n) it is easy to derive the remaining terms of $\boldsymbol{R}$.

Append to the end of $\boldsymbol{PH}$ the first element of $\boldsymbol{PH}$, making $\boldsymbol{PH}$ a complex vector of length $n + 1$. Let $\boldsymbol{PH2}$ be the complex conjugate of the "reverse" of $\boldsymbol{PH}$; i.e., $\boldsymbol{PH2}[j] = \boldsymbol{PH}[n - j]^*$ for $0 \le j \le n$. Define $\boldsymbol{PH3}$ by:

$$\boldsymbol{PH3}[j] = -i(\boldsymbol{PH}[j] - \boldsymbol{PH2}[j]) \times \omega^j,$$

where i is $\sqrt{-1}$ and ω is a $2n^{\text{th}}$ root of unity (such that ω^2 is the n^{th} root of unity used in the FFT). Finally, $\boldsymbol{R}$ is half the sum of $\boldsymbol{PH}$, $\boldsymbol{PH2}$, and $\boldsymbol{PH3}$.

The steps for packing a transformed vector ***R*** (of length $n+1$), to ready it for application of the inverse FFT, are almost the same steps as for unpacking. Let ***R2*** be the complex conjugate of the "reverse" of ***R***. Define ***R3*** by:

$$\boldsymbol{R3}[j] = i(\boldsymbol{R}[j] - \boldsymbol{R2}[j]) \times \omega^{-j},$$

where i and ω are as immediately above. The final result is the first n terms of half the sum of ***R***, ***R2***, and ***R3***.

APPENDIX G

APL ROUTINES FOR PACKING AND UNPACKING

```
UNPACK:

 [0]  R←UNPACK H;H2;H3
 [1]  ⍝ UNPACKS TRANSFORMED DATA.  ASSUMES X
        IS THE RESULT OF
 [2]  ⍝ APPLYING THE FFT TO A LENGTH 2N REAL
        VECTOR WHICH HAD BEEN
 [3]  ⍝ PACKED INTO A 2×N COMPLEX ARRAY.
 [4]  ⍝ RESULT IS A 2 × N+1 ARRAY
 [5]  H←H,H[;0]
 [6]  H2←⌽H
 [7]  H2[1;]←-H2[1;]
 [8]  H3←H-H2
 [9]  H3←(2,N+1)⍴(,-⍉H3×OMEGA2),,+⍉H3×⊖OMEGA2
[10]  H3←⊖H3
[11]  H3[1;]←-H3[1;]
[12]  R←0.5×H+H2+H3

PACK:

 [0]  R←PACK X;X2;X3
 [1]  ⍝ PACKS TRANSFORMED VARIABLE
 [2]  X2←⌽X
 [3]  X2[1;]←-X2[1;]
 [4]  X3←X-X2
 [5]  X3←(2,N+1)⍴(,+⍉X3×OMEGA2),,-⍉OMEGA2×⊖X3
 [6]  X3←⊖X3
 [7]  X3[0;]←-X3[0;]
 [8]  R←0.5×X+X2+X3
 [9]  R← 0 ¯1 ↓R
```

OMEGA2 (the $2n^{th}$ roots of unity) is generated by INIT and INITOMEGA in Appendix D.

APPENDIX H

APL FUNCTIONS FOR THE COMPLETE ALGORITHM

This Appendix gives an APL function, AGGDISTR, that implements the full algorithm. AGGDISTR calls a number of subroutines. The subroutines FFT and INVFFT are listed in Appendix D, and the subroutines PACK and UNPACK are listed in Appendix G. The only other subroutine needed is MULT, listed below. Before running AGGDISTR it is necessary to run INIT, which sets certain global variables. INIT calls INITOMEGA and INITPIK; these three programs are given in Appendix D.

```
AGGDISTR:

 [0]  AGGDISTR;M;M2;P;S;PS1;PSI;PZERO;FS1;
        FSI;AI;I;BIN;FSIFLAG;X;J;Y;PY;FAI;FY
 [1]  ⎕IO←0
 [2]  ⍝ ----
 [3]  'Input the smallest number of claims
        with non-zero probability,'
 [4]  'M:'
 [5]  ' '
 [6]  M←⎕
 [7]  ' '
 [8]  →((M≥0)∧(M=⌊0.5+M))⍴SKIP1 ⍝ M must be a
        non-negative integer.
 [9]  'M, ',(⍕M),' is not a non-negative
        integer.  Stopped.'
[10]  →0
[11]  SKIP1:' '
[12]  'Input densities of claim frequency
        distribution.  These should'
[13]  'be the probabilities of M, M+1, M+2,
        ... claims.'
[14]  ' '
[15]  P←,⎕
[16]  →(((⍴P)≠1)∨(P[0]≠0))⍴SKIP2 ◇ ⍝ IF ONLY
        ONE NUMBER IS INPUT AND IT IS ZERO,
        THEN EXIT.
[17]  'Only one number was input, and it is '
        ,(⍕P),'.  Stopped.'
[18]  →0
[19]  SKIP2:' '
[20]  'Input vector for severity distribu-
        tion.  Must be of length ',(⍕N),'.'
[21]  ' '
[22]  S←,⎕
```

```
[23]  →((ρS)=N)ρSKIP3 ◇ ⍝ IF S IS NOT OF
         LENGTH N, THEN EXIT.
[24]  'Length of S is ',(⍕ρS),'.  Should be '
         ,(⍕N),'.  Stopped.'
[25]  →0
[26]  SKIP3:' '
[27]  ⍝ ----
[28]  AGG←(2,N)ρ0
[29]  PS1←⍉(N,2)ρS,(Nρ0) ◇ ⍝ 'PACK' S.
[30]  S←0 ◇ ⍝ FREE UP SPACE
[31]  PZERO←0 ◇ ⍝ INITIALIZE - WILL BE RESET
         IF M=0.
[32]  ⍝ ----
[33]  ⍝ THREE CASES ARE CONSIDERED, M=0, M=1,
         OR M>1.
[34]  →(M=0)ρMEQ0
[35]  →(M=1)ρMEQ1
[36]  →(M>1)ρMGT1
[37]  MEQ0:PZERO←P[0] ◇ P←,1↓P ◇ M←1
         ⍝ CONVERT TO CASE M=1
[38]  MEQ1:M2←ρP
[39]  AGG←P[0]×PS1
[40]  →(M2=1)ρENDIT
[41]  FS1←UNPACK FFT PS1
[42]  PSI←INVFFT PACK FS1 MULT FS1
[43]  PSI[;TAIL]←0
[44]  AI← 0.5 0.5 ◇ I←2 ◇ PS1←0
[45]  →MAINLOOP
[46]  ⍝ ----
[47]  ⍝ ----
[48]  MGT1: ⍝ START BINARY POWER TRICK.
[49]  BIN←,1↓((1+⌊2⊛M)ρ2)⊤M ⍝ EXPRESS M AS
         BINARY VECTOR, DROP FIRST TERM
[50]  FSI←FS1←UNPACK FFT PS1
[51]  BINLOOP:FSI←FSI MULT FSI ◇ PSI←INVFFT
         PACK FSI ◇ PSI[;TAIL]←0
[52]  'BINLOOP ',(⍕BIN),' ',⍕⎕TS
[53]  →((1↑BIN)=0)ρSKIP
[54]  FSI←UNPACK FFT PSI ◇ FSI←FSI MULT FS1 ◇
         PSI←INVFFT PACK FSI ◇ PSI[;TAIL]←0
[55]  SKIP:BIN←,1↓BIN ◇ →(0=ρBIN)ρEXIT
[56]  FSI←UNPACK FFT PSI
[57]  →BINLOOP
[58]  EXIT: ⍝ THIS IS THE EXIT FROM THE
         BINARY POWER TRICK.
[59]  ⍝ ----
[60]  AI←,1 ◇ I←1 ⍝ SET AI, INDEX
[61]  ALOOP:I←I+1 ◇ AI←(1÷I)×(ΦAI)+AI←(1+⍳I)×
         AI,0
[62]  →(I<M)ρALOOP
[63]  M2←¯1+M+(ρP) ◇ PS1←0
[64]  ⍝ ----
[65]  ⍝ ----
[66]  MAINLOOP:'MAINLOOP ',(⍕I),' ',⍕⎕TS
[67]  FSIFLAG←0
```

```
[68]  ⍝ ----
[69]  →(I≥100)⍴BETA
[70]  ⍝ IF HERE, WANT TO CONVOLUTE AI WITH
         PSI WITHOUT FFT'S
[71]  X←N↑,⍉PSI
[72]  J←0 ◇ Y←0
[73]  LOOPB:Y←Y+AI[J]×X
[74]  J←J+1 ◇ →((J>I-1)∨(J>N-1))⍴ENDLOOPB
[75]  X←0,¯1↓X ⍝ DROP LAST ELEMENT OF X, ADD
         A ZERO TO THE FRONT.
[76]  →LOOPB
[77]  ENDLOOPB:PY←⍉(N,2)⍴Y,N⍴0 ◇ X←Y←0
[78]  →GAMMA
[79]  ⍝ ----
[80]  BETA:FAI←UNPACK FFT⍉(N,2)⍴(N↑AI,N⍴0),
         N⍴0
[81]  FSIFLAG←1
[82]  FSI←UNPACK FFT PSI
[83]  FY←FSI MULT FAI
[84]  PY←INVFFT PACK FY
[85]  PY[;TAIL]←0 ◇ FY←0
[86]  ⍝ ----
[87]  GAMMA:AGG←AGG+P[I-M]×PY ◇ PY←0
[88]  I←I+1 ◇ →(I>M2)⍴ENDIT
[89]  ⍝ ----
[90]  →(FSIFLAG=1)⍴SKIP4
[91]  FSI←UNPACK FFT PSI
[92]  SKIP4:FSI←FSI MULT FS1
[93]  PSI←INVFFT PACK FSI
[94]  PSI[;TAIL]←0
[95]  ⍝ ----
[96]  AI←(1÷I)×(ΦAI)+AI←(1+⍳I)×AI,0
[97]  →MAINLOOP
[98]  ENDIT:⎕TS ◇ AGG←N↑,⍉AGG
[99]  AGG[0]←AGG[0]+PZERO
[100] ' '
[101] '***  REMEMBER, RESULT IS IN AGG.  ***'
[102] ' '

MULT:

 [0]  Z←X MULT Y
 [1]  Z←(⍴X)⍴(,-⌿X×Y),,+⌿X×⍉Y
```

APPENDIX I

PARAMETER UNCERTAINTY FOR THE SEVERITY DISTRIBUTION

We will show mathematically that it is fundamentally impossible to reflect parameter uncertainty for the severity distribution by computing the aggregate distribution using a severity distribution with a larger variance than the best-estimate severity distribution. This discussion is based on suggestions by Venter.

Parameter uncertainty for the severity distribution is reflected by choosing a distribution a with mean 1 and variance greater than 0, and computing the aggregate distribution:

$$AGG = aS_1 + aS_2 + ... + aS_T .$$

Here AGG, S_i, and T are the aggregate distribution, the severity distribution, and the claim count distribution, defined earlier. The above equation is written to indicate that one sample from T is associated with one sample from a and multiple samples from S. The above equation could also be written

$$AGG = a(S_1 + S_2 + ... + S_T) .$$

A general fact about variance (for arbitrary/independent distributions X and Y) that will be used is:

$$\mathrm{Var}(XY) = \mathrm{Var}(X)\,\mathrm{Var}(Y) + (\mathrm{E}(X))^2\,\mathrm{Var}(Y) + (\mathrm{E}(Y))^2\,\mathrm{Var}(X). \quad (I.1)$$

Also, recall that

$$\mathrm{Var}(S_1 + S_2 + ... + S_T) = \mu_T\sigma_S^2 + \mu_S^2\sigma_T^2 \quad (I.2)$$

Here μ_T and μ_S are the means of the claim count and severity distributions, and σ_T^2 and σ_S^2 are the variances of the respective distributions.

Set

$$X = a,\ Y = S_1 + S_2 + ... + S_T,\ \text{and}\ XY = AGG$$

and substitute in equation I.1 above. This gives

$$\text{Var}(AGG) = \text{Var}(a)\,\text{Var}(S_1 + S_2 + \ldots + S_T) + (\text{E}(a))^2\,\text{Var}(S_1 + S_2 + \ldots S_T) + (\text{E}(S_1 + S_2 + \ldots + S_T))^2\,\text{Var}(a).$$

Denote $\text{Var}(a)$ by σ_a^2, note that $\text{E}(S_1 + S_2 + \ldots + S_T)$ is $\mu_T\mu_S$, recall that $\text{E}(a)$ is 1, and substitute using equation I.2 to obtain

$$\text{Var}(AGG) = \left(1 + \sigma_a^2\right)\left(\mu_T\,\sigma_S^2 + \mu_S^2\,\sigma_T^2\right) + \left(\mu_T\,\sigma_S\right)^2\sigma_a^2.$$

Dividing by $(\mu_T\mu_S)^2$ gives a formula for the square of the coefficient of variation for the aggregate distribution:

$$\frac{\text{Var}(AGG)}{(\mu_T\mu_S)^2} = (1 + \sigma_a^2)\left[\frac{\sigma_S^2}{\mu_T\mu_S^2} + \frac{\sigma_T^2}{\mu_T^2}\right] + \sigma_a^2 . \qquad \text{(I.3)}$$

Now consider what happens as the mean of the claim count distribution, i.e., the expected number of claims, increases towards infinity. For the moment, assume there is no parameter uncertainty for the claim count distribution. The term $\frac{\sigma_S^2}{\mu_T\mu_S^2}$ tends to zero as μ_T increases. If the claim count distribution is Poisson, or is negative binomial with a fixed "probability of success" parameter, then the term $\frac{\sigma_T^2}{\mu_T^2}$ also tends to zero. (If the negative binomial is parameterized so the density function is $f(x) = \binom{y + x - 1}{x} p^y (1-p)^x$ with $y > 0$ and $0 < p < 1$, then p is the probability of success parameter.)

Thus, for any fixed severity distribution, the limit of the square of the coefficient of variation of the aggregate distribution, as μ_T goes to infinity, is equal to σ_a^2. In a practical sense, this means that, as the expected number of claims becomes large, the effect of the claim count distribution and the severity distribution on the coefficient of variation of the aggregate distribution becomes minimal. The coefficient of variation of the aggregate distribution is determined by the parameter uncertainty for the severity distribution.

In particular, this shows that there is a fundamental difference between the effect on the aggregate distribution of parameter uncertainty in the severity distribution and the effect of using a severity distribution with a greater variance. If a severity distribution, S', with a larger variance is substituted for the best-estimate severity distribution, S, and $AGG = S'_1 + S'_2 + \ldots + S'_T$ is computed (this is the same as setting a to a constant 1), then as μ_T goes to infinity, the coefficient of variation of the aggregate distribution tends to zero, and not to some positive value as above.

The two approaches differ in the independence assumptions regarding the samples from the severity distribution. If the severity distribution is diffused, then each draw from the severity distribution is a combination of an independent draw from the best-estimate severity distribution and an independent draw from the distribution used to reflect parameter uncertainty for the severity distribution. Under the correct method of reflecting parameter uncertainty, each draw is still an independent draw from the severity distribution, but there is only one draw from the distribution reflecting parameter uncertainty for each draw from the claim count distribution.

Equation I.3 shows that diffusing the severity distribution is not an adequate method for the recognition of parameter uncertainty for the severity distribution. Equation I.3 can also show the effect of parameter uncertainty for the claim count distribution, and the remainder of this appendix gives a brief discussion of that effect.

Apart from one quick comment at the end, only one method of representing parameter uncertainty for the claim count distribution will be considered here. It will be assumed that the best-estimate claim count distribution is Poisson with parameter λ. That is, the best-estimate claim count distribution, N, has probability density function

$$P(N = x \mid \lambda) = \frac{\lambda^x}{x!} e^{-\lambda}.$$

Parameter uncertainty is incorporated by assuming that λ follows a gamma distribution, denoted Λ, with probability function

$$P(\Lambda = \lambda) = \frac{(\lambda/b)^{c-1} e^{-\lambda/b}}{b\Gamma(c)}.$$

Above, Γ is the gamma function and b and c are parameters. The distribution Λ has mean bc and variance b^2c. The parameter b is the scale parameter because changing b by some factor has the effect of multiplying the distribution uniformly by that same factor. The parameter c is the shape parameter.

Straightforward computations show that the claim count distribution, T, that results from compounding the Poisson distribution with the gamma distribution is a negative binomial distribution with probability function:

$$P(T = t) = \binom{t+c-1}{t} \left(\frac{1}{b+1}\right)^c \left(\frac{b}{b+1}\right)^t.$$

Here b and c are the parameters from the gamma distribution. This negative binomial distribution has mean bc and variance $(bc)(b+1)$. Thus σ_T^2/μ_T^2 is $\frac{1}{c}\left(1 + \frac{1}{b}\right)$.

One way to reflect a constant degree of parameter uncertainty in the claim count distribution while increasing the mean is to allow b to increase but to hold c constant. This maintains a constant percentage of uncertainty regarding the mean of the claim count distribution. In the Equation I.3, the term $\frac{\sigma_S^2}{\mu_I \mu_S^2}$ tends to zero as μ_I increases (as it did above). But now the term $\frac{\sigma_I^2}{\mu_I^2}$ tends to a positive limit, namely $1/c$. Thus, Equation I.3 shows that the coefficient of variation of the aggregate distribution has a component due to the parameter uncertainty in the claim count distribution that does not drop below a certain minimum no matter how large the mean of the claim count distribution becomes.

A little more generally, as μ_T goes to infinity, the limit of the square of the coefficient of variation of the aggregate distribution is

$$\left(1+\sigma_a^2\right)\left(\lim_{\mu_T\to\infty}\frac{\sigma_T^2}{\mu_T^2}\right)+\sigma_a^2 .$$

The severity distribution and the best-estimate claim count distribution have no effect on this limit. This limit depends only on the amount of parameter uncertainty reflected in each of these distributions.

APPENDIX J

PROOFS OF FORMULAE FOR EXCESS CLAIM COUNT DISTRIBUTIONS

This appendix proves that if a claim count distribution is binomial, Poisson, or negative binomial, with mean λ and variance σ^2, and the probability of a claim being excess of some attachment point is α, then the excess claim count distribution is binomial, Poisson, or negative binomial, respectively, with mean $\alpha\lambda$ and variance $\alpha\lambda + \alpha^2(\sigma^2 - \lambda)$. For each of the three types of claim count distributions, it is shown that the selection of claim counts from the given distribution with mean λ and variance σ^2, followed by selection of excess claims with probability α (under a binomial process) gives a distribution of the same type with mean $\alpha\lambda$ and variance $\alpha\lambda + \alpha^2(\sigma^2 - \lambda)$.

The Poisson case is considered first. Here $\lambda = \sigma^2$, so σ^2 will not appear. The total distribution will be T and the excess distribution will be X. For the total distribution:

$$P(T = t) = \frac{\lambda^t}{t!} e^{-\lambda}.$$

Given t claims in the total distribution, the distribution of excess claims is:

$$P(X = x \mid T = t) = \binom{t}{x} \alpha^x (1 - \alpha)^{t-x}.$$

$P(X = x)$ is the sum over all t of $P(T = t) \times P(X = x \mid T = t)$; i.e.,

$$P(X = x) = \sum_{t=x}^{\infty} \frac{\lambda^t}{t!} e^{-\lambda} \binom{t}{x} \alpha^x (1 - \alpha)^{t-x}.$$

Letting $i = t - x$ so $t = x + i$ this becomes:

$$P(X = x) = \sum_{i=0}^{\infty} \frac{\lambda^{x+i}}{(x+i)!} e^{-\lambda} \binom{x+i}{x} \alpha^x (1 - \alpha)^i.$$

This is a "ground up" computation of the claim count distribution for aggregate claims. The starting point was the distribution of total claims, T, and then excess claims were selected according to a binomial process to get the distribution of excess claims.

It is necessary to show that this sum is $\frac{(\alpha\lambda)^x}{x!} e^{-\alpha\lambda}$.

But:

$$e^{-\alpha\lambda} e^{-\lambda+\lambda-\alpha\lambda} = e^{-\lambda} e^{\lambda(1-\alpha)} = e^{-\lambda} \sum_{i=0}^{\infty} \frac{\lambda^i (1-\alpha)^i}{i!}$$

$$\Rightarrow \frac{\alpha^x \lambda^x}{x!} e^{-\alpha\lambda} = \sum_{i=0}^{\infty} \frac{\lambda^x \lambda^i}{(x+i)!} e^{-\lambda} \frac{(x+i)!}{x!\, i!} \alpha^x (1-\alpha)^i$$

$$\Rightarrow \frac{(\alpha\lambda)^x}{x!} e^{-\alpha\lambda} = \sum_{i=0}^{\infty} \frac{\lambda^{x+i}}{(x+i)!} e^{-\lambda} \binom{x+i}{x} \alpha^x (1-\alpha)^i$$

as was to be shown.

Now much the same is done for the negative binomial. According to Hastings and Peacock [24, p. 92], the negative binomial has density function:

$$P(Y = y) = \binom{x+y-1}{y} p^x q^y$$

where x and p are parameters and $q = 1 - p$. This distribution has mean xq/p and variance xq/p^2. If parameters x and p result in a mean of λ and a variance of σ^2, then parameters of x and $p/(p + \alpha - \alpha p)$ result in mean $\alpha\lambda$ and variance $\alpha\lambda + \alpha^2(\sigma^2 - \lambda)$. Let T be the total claim count distribution, and let Y be the excess claim count distribution. Then:

$$P(T = t) = \binom{x+t-1}{t} p^x q^t$$

$$P(Y = y \mid T = t) = \binom{t}{y} \alpha^y (1-\alpha)^{t-y}$$

$$P(Y = y) = \sum_{t=y}^{\infty} \binom{x+t-1}{y} p^x q^t \binom{t}{y} \alpha^y (1-\alpha)^{t-y}$$

$$P(Y = y) = \sum_{i=0}^{\infty} \binom{x+y+i-1}{i+y} p^x q^{i+y} \binom{i+y}{y} \alpha^y (1-\alpha)^i$$

where $i + y$ was substituted for t to get from the next-to-last equation to the last equation.

It is necessary to show that the right-hand side of this equation is equal to:

$$\binom{x+y-1}{y} \left(\frac{p}{p+\alpha-\alpha p}\right)^x \left(\frac{\alpha-\alpha p}{p+\alpha-\alpha p}\right)^y$$

which is the density function for the negative binomial with the parameters for the excess distribution.

Now note that the Maclaurin series for $1/(1-z)^a$ is given by:

$$\frac{1}{(1-z)^a} = \sum_{i=0}^{\infty} \binom{a+i-1}{i} z^i .$$

Substitute $a = x + y$ and $z = (1-p)(1-\alpha) = q(1-\alpha)$ to get $1 - z = p + \alpha - \alpha p$ and:

$$\left(\frac{1}{p+\alpha-\alpha p}\right)^{x+y} = \sum_{i=0}^{\infty} \binom{x+y+i-1}{i} q^i (1-\alpha)^i .$$

Multiply the left-hand side of the above equation by:

$$K = \frac{p^x \alpha^y q^y (x+y-1)!}{(x-1)!\, y!}$$

and multiply the right-hand side by $K \dfrac{(i+y)!}{(i+y)!}$, rearrange, and obtain:

$$\binom{x+y-1}{y}\left(\frac{p}{p+\alpha-\alpha p}\right)^x\left(\frac{\alpha-\alpha p}{p+\alpha-\alpha p}\right)^y$$

$$= \sum_{i=0}^{\infty}\binom{x+y+i-1}{i+y}p^x q^{i+y}\binom{i+y}{y}\alpha^y(1-\alpha)^i,$$

as was to be shown.

Finally, consider the binomial. According to Hastings and Peacock [24, p. 36], the binomial has density function:

$$\mathrm{P}(X=x) = \binom{n}{x}p^x q^{n-x},$$

where n and p are parameters and $q = 1 - p$. This distribution has mean np and variance npq. If parameters n and p result in a mean of λ and a variance of σ^2, then parameters of n and αp result in mean $\alpha\lambda$ and variance $\alpha\lambda + \alpha^2(\sigma^2 - \lambda)$. As above, let T be the total claim count distribution, and let X be the excess claim count distribution. Then:

$$\mathrm{P}(T=t) = \binom{n}{t}p^t q^{n-t},$$

$$\mathrm{P}(X=x \mid T=t) = \binom{t}{x}\alpha^x(1-\alpha)^{t-x},$$

$$\mathrm{P}(X=x) = \sum_{t=x}^{n}\binom{n}{t}p^t q^{n-t}\binom{t}{x}\alpha^x(1-\alpha)^{t-x}.$$

Let $t = x + i$, so:

$$P(X = x) = \sum_{i=0}^{n-x} \binom{n}{x+i} p^{x+i} q^{n-x-i} \binom{x+i}{x} \alpha^x (1-\alpha)^i$$

It is necessary to show that the right-hand side of the above equation is equal to:

$$\binom{n}{x} (\alpha p)^x (1-\alpha p)^{n-x}.$$

Begin with an equality due to the binomial theorem:

$$(z+1)^{n-x} = \sum_{i=0}^{n-x} \binom{n-x}{i} z^i.$$

Now let $z = \frac{p(1-\alpha)}{1-p}$, so $z + 1 = \frac{1-\alpha p}{1-p}$. Substituting gives:

$$\left(\frac{1-\alpha p}{1-p}\right)^{n-x} = \sum_{i=0}^{n-x} \binom{n-x}{i} \left(\frac{p(1-\alpha)}{1-p}\right)^i.$$

$$\frac{(1-\alpha p)^{n-x}}{(n-x)!} = \sum_{i=0}^{n-x} \frac{p^i (1-\alpha)^i (1-p)^{n-x}}{i!\,(n-x-i)!\,(1-p)^i}.$$

Multiply the left side by $K = \frac{n!\,p^x\alpha^x}{x!}$ and the right side by $K \cdot \frac{(x+i)!}{(x+i)!}$ and rearrange to get:

$$\frac{n!}{x!\,(n-x)!} p^x \alpha^x (1-\alpha p)^{n-x} =$$

$$\sum_{i=0}^{n-x} \frac{n!}{(x+i)!\,(n-x-i)!} p^x p^i (1-p)^{n-x-i} \frac{(x+i)!}{x!\,i!} \alpha^x (1-\alpha)^i$$

or:

$$\binom{n}{x}(\alpha p)^x (1-\alpha p)^{n-x} = \sum_{i=0}^{n-x}\binom{n}{x+i} p^{x+i} q^{n-x-i}\binom{x+i}{x}\alpha^x (1-\alpha)^i$$

which completes the proof.