Management Science and Operations Research

Harvey J. Greenberg Editor

Computing Poisson Probabilities

BENNETT L. FOX and PETER W. GLYNN

ABSTRACT: We propose an algorithm to compute the set of individual (nonnegligible) Poisson probabilities, rigorously bound truncation error, and guarantee no overflow or underflow. Work and space requirements are modest, both proportional to the square root of the Poisson parameter. Our algorithm appears numerically stable. We know no other algorithm with all these (good) features. Our algorithm speeds generation of truncated Poisson variates and the computation of expected terminal reward in continuous-time, uniformizable Markov chains. More generally, our algorithm can be used to evaluate formulas involving Poisson probabilities.

1. INTRODUCTION

We give an algorithm to compute the set of individual (nonnegligible) Poisson probabilities, needed for example in the applications pointed out in section 1.1. To get finite termination, we clearly have to truncate the Poisson distribution. Unlike previous contributions, we bound the mass in the truncated tails from above and the remaining individual probabilities from below. Given any "reasonable" ε (see section 3), we find a left truncation point L and a right truncation point R such that for a Poisson random variable N with parameter λ we have $P[L \le N \le R] \ge 1 - \varepsilon$ and $R - L = O(\sqrt{\lambda})$. Starting from a mode $m = \lfloor \lambda \rfloor$ we recursively compute weights $w(m-1), w(m-2), \ldots, w(L)$ and $w(m+1), w(m+2), \ldots, w(R)$ such that for some constant α we have $w(i) = \alpha p(i)$ for $L \le i \le R$ where $p(i) = P[N = i]$, $\alpha \doteq W$, and $W = w(L) + \cdots + w(R)$. Our (new) lower bounds let us scale the weights, via the choice of $w(m)$, to guarantee without repeated checking that no underflows or overflows occur. Our (new) upper bounds let us avoid estimating tail masses by summing estimated probabilities in the complementary region, a computation likely to be significantly contaminated by roundoff error. As far as we know, no other published algorithms for computing Poisson probabilities use rigorous, practical bounds such as ours, and so these algorithms are unsatisfactory. Such bounds do not seem readily accessible elsewhere.

The overall work and space complexities are both $O(\sqrt{\lambda})$, which lets us handle all practical λ's. Our algorithm does not restrict λ. Today's desktop computers can easily handle λ in the range 0 to 10^{10} say. We give a crude analysis of roundoff error that suggests that our method is numerically stable. Possibly $w(i)/W$ may underflow for some i, but this is ordinarily irrelevant if the computations are properly arranged. For example, we can compute a weighted sum $w(L)f(L) + \cdots + w(R)f(R)$ and (only) then divide the sum by W. Likewise, the overall error in a computation is ordinarily small even though the individual estimated probabilities are all a bit off. To illustrate, we note

Bennett L. Fox's research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

Peter W. Glynn's research was supported by the National Science Foundation under Grant ECS-8404809 and the U.S. Army under Contract Number DAAG29-80-C-0041.

PROPOSITION 1. Let f be a real-valued function with $\|f\| = \sup\{f(j): j = 0, 1, 2, \ldots\}$ and $P[L \le N \le R] \ge 1 - \varepsilon$. In exact arithmetic,

$$\left| (1/W) \sum_{j=L}^{R} w(j)f(j) - \sum_{j=0}^{\infty} p(j)f(j) \right| \le 2\varepsilon \|f\|.$$

Proof. It suffices to show that $\sum_{j=0}^{\infty} |q(j) - p(j)| \le 2\varepsilon$ where $q(i) = w(i)/W = p(i)/\beta$ with $\beta = p(L) + \cdots + p(R)$. Now

$$\sum_{j=0}^{\infty} |q(i) - p(i)| \le \sum_{i=L}^{R} [q(i) - p(i)] + \varepsilon$$

$$= \sum_{i=L}^{R} p(i)\left[\frac{1}{\beta} - 1\right] + \varepsilon \le 2\varepsilon. \quad \square$$

For large λ, our L is $\lambda - O(\sqrt{\lambda})$ and our R is $\lambda + O(\sqrt{\lambda})$. This verifies that our overall work and space complexities are both $O(\sqrt{\lambda})$. In contrast, the 1982 IMSL routine asks the user to specify a parameter k—not necessarily related to λ—and outputs estimates of $p(0)$, $p(1), \ldots, p(k)$. This amounts to setting $L = 0$. The k specified by any reasonable user will be $O(\lambda)$. Thus, in practice the IMSL routine requires $O(\lambda)$ work.

1.1 Motivation

Gross and Miller [9] and Fox [4] consider problems of this form, in which large values of λ typically occur. Given the $f(j)$'s, the first summation in proposition 1 can be computed in $O(\sqrt{\lambda})$ time; without the left truncation, it would take $O(\lambda)$ time. For the terminal-reward problem considered by Gross and Miller [9], the required $f(j)$'s can be computed in $O(\sqrt{\lambda})$ time using successive (matrix) squaring up to L as Fox [4] details. On the other hand, consider using the weights to compute Erlang's loss formula (e.g., see Gross and Harris with c servers as $w(c)/[w(L) + \cdots + w(c)]$ for $L \le c \le R$. If $L = 0$, the error is zero (ignoring roundoff and assuming no underflow occurs) but computing the weights then takes $O(\lambda)$ time. For $L > 0$ we do not attempt a formal error analysis here, which would require lower as well as upper bounds on the left tail. Our bounds may indicate a reasonable choice of R.

For another example, suppose that we want to generate truncated Poisson variates by (efficiently-implemented) inversion or by the alias method. Simply scale standard uniform variates as $u \leftarrow uW$ and then use the $w(j)$'s directly. Our algorithm reduces the setup time from $O(\lambda)$ for a naive method (with no error bounds) to $O(\sqrt{\lambda})$ with bounds on truncation error. Fox [5] bound the loss in coverage probability for confidence intervals constructed from truncated variates with any distribution. Bratley, Fox, and Schrage [2] argue that truncation avoids statistical anomalies and allows variate generation by (efficiently-implemented) inversion in $O(1)$ marginal time, independently of the truncation point. Inversion is compatible with common random numbers and antithetic variates, but rejection methods typically are not.

1.2 Numerical Stability

The $w(j)$'s are slightly inaccurate due to roundoff error. They are all positive, so no cancellation errors occur when adding them to get W. Thus, the additional roundoff error induced by adding the $w(j)$'s is potentially troublesome only when there are many summands (large λ). If we were simply to add from left to right say, then for example $w(\lceil (m + R)/2 \rceil) \oplus \tilde{W} = \tilde{W}$ in finite-precision floating-point ($\oplus$ = "add") arithmetic for large enough λ. Here $\tilde{W} = w(L) + \cdots + w(\lceil (m + R)/2 \rceil - 1)$. Thus, we will get $W = \tilde{W}$ though $w(\lceil (m + R)/2 \rceil) + \cdots + w(R)$ is not negligible. The cure is to *add small weights first*. This widely applicable principle to attenuate roundoff error is also apparently used in the 1982 IMSL routine to compute Poisson probabilities.

Recursive computation of weights starting from the mode is by itself not a new idea. What is new is its coupling with *a priori* determination of truncation points and *a priori* underflow checks. The obvious choice for $w(m)$ is one. Given the computer's underflow and overflow thresholds, we typically choose $w(m)$ far larger to avoid underflow worries.

Knüsel [10, pages 1028–1029] gives a numerically-stable method to compute $p(i)$ for any *fixed* i, though he gives no corresponding error bounds. It would be inefficient to use that method to compute the set of probabilities $\{p(L), \ldots, p(R)\}$. He also gives a numerically-stable method to estimate the masses in each tail with prescribed *relative* error. These bounds can be transformed into corresponding bounds on *absolute* error. Whether such transformed bounds are looser or tighter than ours is an open question. Our bounds take $O(1)$ time to compute, independently of *all* other parameters. The time and space complexities to compute Knüsel's bounds are unspecified and unclear. Our bounds on tail masses, as estimates of them, probably would have high relative error. In view of proposition 1, this seems irrelevant to the applications cited above; in other applications, however, small relative error may be important. As a check for programming errors, our $w(m)/W$ should be approximately Knüsel's estimate of $p(m)$.

1.3 Overview

In section 2, we give an algorithm to find the weights, which requires a subroutine to find L, R, and $w(m)$ and to check that underflow will not occur. We sketch that subroutine in section 3, based on bounds in sections 4 and 5. Section 6 justifies the bounds in section 5. In section 7 we outline generalizations to other discrete distributions.

2. COMPUTING WEIGHTS

In this section we present an easily-programmed algorithm to compute the weights followed by an analysis of roundoff error.

Algorithm. WEIGHTER (λ, ε, τ, Ω; L, R, $w(L), \ldots, w(R)$, W, F)

```
Inputs:
  Poisson parameter          λ
  error tolerance            ε ≥ 10^-10
  underflow threshold        τ
  overflow threshold         Ω
Outputs:
  left truncation point      L
  right truncation point     R
  weights                    w(L), . . . , w(R)
  total weight               W
  flag                       F

Subroutine required:
  FINDER (λ, ε, τ, Ω; L, R, w(m), F)—see section 3

Comments:
  w(i) ≐ p(i)W
    where p(i) = Poisson probability
  For λ ≥ 400:
    mass left of L ≤ ε/2
    mass right of R ≤ ε/2
  For 0 < λ < 400:
    mass left of L ≤ ε/2   (note: L = 0 for λ ≤ 25)
    mass right of R ≤ ε/2 + 6 × 10^12 τ/Ω—see section 3
  Section 3 explains where the 400 comes from.
  F = "true" if no underflow can occur while computing the weights
  F = "false" indicates that weights are not computed due to potential underflow
  w(i)/W may underflow even when F = "true"

  Initialize:
    Set m ← ⌊λ⌋ (mode)
    Get L, R, w(m), and F from FINDER; if F = "false", exit.
  Down:
    Set j ← m
    While j > L, execute
      w(j − 1) ← (j/λ)w(j)
        j ← j − 1
Up:
  If λ < 400, go to Special
  Else, set j ← m
  While j < R, execute
    w(j + 1) ← (λ/(j + 1))w(j)
      j ← j + 1

Compute W:
  [Comment: We want to compute W ← w(L) + · · · + w(R)]
  [Comment: To attenuate roundoff, we add small terms first.]
    W ← 0
    s ← L
    t ← R
    While s < t, execute
      If w(s) ≤ w(t) then
          W ← W + w(s)
          s ← s + 1
        else
          W ← W + w(t)
          t ← t − 1
    W ← W + w(s)
Exit.
Special:
  If R > 600, set F = "false" and then exit.
    [Comment: Underflow is possible but not certain; section 3 explains where the 600 comes from.]

  Else, set j ← m.
  While j < R, execute
    q ← λ/(j + 1)
    If w(j) > τ/q,
      then set w(j + 1) ← qw(j) and j ← j + 1
    else
      set R ← j
      go to Compute W
Go to Compute W.
```

Computing each new weight takes two floating-point operations, accounting for the 2 in the exponents below. We define the relative roundoff error to be the maximum of the ratio of the computed result to the true result and the reciprocal of that ratio. With multiplication and division, bounds on relative roundoff errors multiply. Let u be the *unit roundoff*. Here $u \doteq 2^{1-b}$ where b is the number of bits in the mantissas of the computer's floating-point numbers. The relative roundoff error accumulated in Down and Up is at worst $O[(1 + u)^{2(m-L)}]$ and $O[(1 + u)^{2(R-m)}]$ respectively, likely gross overestimates. For example, $(1 + 10^{-7})^{1000} \doteq 1.00010$. The implicit proportionality factor associated with estimates of roundoff error depends on whether floating-point arithmetic chops or rounds. It is less than 1.01 in all cases, if we replace u by $10u$.

Since Compute W involves only positive numbers, it follows (from Gill, Murray, and Wright [6, pages 11–12] for example) that the associated relative roundoff error is at worst $O[(1 + u)^{R-L}]$. Consider using double precision to attenuate it. The actual roundoff error is probably much less than the bound, because we add small weights first. Pushing the principle of adding small terms first to the limit, we could put all (initial) summands in a heap, remove the smallest two, insert their sum in the heap, and so on, until the heap empties. In view of Glynn's [7] corollary 1.4 (showing roughly that the mass in the tails does not overwhelm the next summand) such elaborate measures seem unnecessary and the method in WEIGHTER looks good enough.

Even with that corollary, simply terminating when the current weight falls below a heuristic threshold would leave us with no rigorous error bound on the corresponding truncated tail masses. The 1982 IMSL routine outputs estimates of $p(0), p(1), \ldots, p(k + 1)$ with k specified by the user. It does not check whether the user-specified k is reasonable; for example, for large λ, picking $k = \lceil\sqrt{\lambda}\rceil$ or $k = \lambda^2$ is unreasonable. If k is at least of order λ, then the IMSL routine spends most of its time computing estimates of negligible probabilities; it may have to rescale often to avoid underflow.

The roundoff error associated with L and R does not seem severe, but analyzing it would require specification of how the machine computes exponentials. As a hedge against roundoff, divide the nominal ε by 10 say. If the nominal ε is already small, we will see later that this hedge does not have a major impact on L and R.

3. FINDING TRUNCATION POINTS

We now give a recipe to find L and R, given ε, as required by WEIGHTER. The heuristic choice $w(m) = \Omega/10^{10}(R - L)$ assures that $W \le \Omega/10^{10}$. The factor 10^{10} typically prevents overflow when the weights are subsequently used as in proposition 1 for example. To check for underflow, we scale the lower bounds in corollaries 3 and 4 by $\Omega/10^{10}(R - L)$ before comparing them to τ. Based on the discussion below, writing a subroutine FINDER (λ, ε, τ, Ω; L, R, $w(m)$, F) as required by WEIGHTER is straightforward.

1. If $\lambda = 0$, then set $L = R = 0$ and $F =$ "false".
2. If $0 < \lambda < 25$, then $L = 0$. If $e^{-\lambda} < \tau$, set $F =$ "false" and exit. If $0 < \lambda < 400$, find R using corollary 1 of section 4 with $\lambda = 400$. Increase k through the positive integers greater than 3 until the upper bound is less than $\varepsilon/2$. Set $F =$ "true".
3. If $\lambda \ge 400$, use corollary 1 with the actual λ and proceed as above to find R. Evaluate the lower bound in corollary 3 of section 5 multiplied by $\Omega/10^{10}(R - L)$ at the k corresponding to R. If the result is less than τ, set $F =$ "false" and exit. If $\lambda \ge 25$, then find L using corollary 2 of section 4 with the actual λ. Evaluate the lower bound in corollary 4 of section 5 multiplied by $\Omega/10^{10}(R - L)$ at the k corresponding to L. If the result is greater than τ, set $F =$ "true"; else, set $F =$ "false".

From inspection of corollaries 1 and 2, we see that the k's found above are $o([\log(1/\varepsilon)]^{1/2})$, growing very slowly as ε decreases. In corollaries 1 and 2 the factor $\exp(-k^2/2)$ dominates, when $\lambda \ge 25$ say. At $k = 7$, this factor equals 6.2×10^{-11} approximately and we get $R - L \le 20\sqrt{\lambda}$. Suppose that $\varepsilon = 10^{-10}$. One routinely checks that with $k = 7$, the bounds are less than 10^{-10} for $\lambda \ge 25$. If programming in a language like Fortran where storage for the weights must be assigned in advance, a generous rule of thumb is to allow $\max(\lceil 20\sqrt{\lambda} \rceil, 600)$ cells.

When $\lambda = 400$, then corollary 1 applies for $R \le 600$ which corresponds to $k = 7$. This explains the 600 above. If R is not reset in *special*, then the mass to its right is at most $\varepsilon/2$; otherwise, the mass to the right of R is at most $\varepsilon/2 + 600 \times 10^{10}\tau/\Omega$ because of our choice for $w(m)$.

While computing bounds, use the upper bounds on a_λ, b_λ, and $d(k, \lambda)$ given in section 4 and the lower bound on c_m given in section 5. The remaining factor in any particular bound is an exponential, say $\exp(-g(k))$. Multiply the bound on $a_\lambda d(k, \lambda)$, b_λ, or $c_m\Omega/10^{10}(R - L)$ by $\exp(-g(k) + \lfloor g(k) \rfloor)$. We assume (reasonably) that there is no underflow up to this point. To avoid subsequent underflow, multiply the result by e^{-1} as long as the current product is greater than τe or until $\lfloor g(k) \rfloor$ multiplications occur, whichever happens first. For corollaries 1 and 2, if underflow would occur with the next (hypothetical) multiplication, then the bound is less than τ, hence acceptable providing $\varepsilon > 2\tau$ as seems reasonable. For corollaries 3 and 4, however, if underflow would occur, then ϵ is too small.

Again looking at what happens for $k = 7$, in corollary 3 the dominant factor is $\exp(-(\hat{k} + 1)^2/2) \ge 2.4 \times 10^{-14}$ for $\lambda \ge 25$; in corollary 4, the dominant factor is $\exp(-\hat{k}^2/2 - \hat{k}^3/3\sqrt{\lambda}) \ge 2.1 \times 10^{-15}$ for $\lambda \ge 196$. The discussion following corollary 4 indicates that for $k = 7$ we have to deal with bounds no smaller than 10^{-60} for $\lambda < 196$.

Corresponding to $k = 7$, we get $R - L \le \max(\lceil 20\sqrt{\lambda} \rceil, 100)$. In any case, underflow occurs only if a lower bound is less than $10^{10}(R - L)\tau/\Omega$.

Consider τ/Ω for typical computers. According to the 1982 VAX-11 Fortran reference manual, p. 2–7, $\tau/\Omega \sim 10^{-76}$. According to the 1982 CDC Fortran 5 reference manual, p. 1–5, $\tau/\Omega \sim 10^{-615}$. According to the 1981 Itel iAPX 86, 88 User's Manual, p. S-6, for the 8087 Numerical Data Processor, $\tau/\Omega \sim 10^{-75}$ for single precision and $\tau/\Omega \sim 10^{-615}$ for double precision. Probably our checks for underflow and overflow are superfluous in practice, but they are inexpensive to do and guarantee that, when passed, no problems can occur.

4. BOUNDING POISSON TAILS

Let

$$p_\lambda(i) = e^{-\lambda}\lambda^i/i!, \quad i \ge 0$$

$$Q_\lambda(i) = \sum_{j=i}^{\infty} p_\lambda(j)$$

$$T_\lambda(i) = \sum_{j=0}^{i} p_\lambda(j)$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$\Phi(x) = \int_{-\infty}^{x} \phi(t)\, dt$$

$$a_\lambda = (1 + 1/\lambda)e^{1/16}\sqrt{2}$$

$$b_\lambda = (1 + 1/\lambda)e^{1/8\lambda}.$$

For $\lambda \ge 25$, we get $a_\lambda \le 1.57$ and $b_\lambda \le 1.05$.

Glynn [7] proves

PROPOSITION 2. Suppose $\lambda \ge 2$ and $2 \le i \le (\lambda + 3)/2$. Then

$$Q_\lambda(m + i) \le a_\lambda(1 - \exp(-2i/9))^{-1} \cdot \Phi((i - 3)/2)/\sqrt{2\lambda}).$$

PROPOSITION 3. Suppose $\lambda \ge 2$ and $i \ge 2$. Then

$$T_\lambda(m - i) \le b_\lambda\Phi((i - 3/2)/\sqrt{\lambda}).$$

We reparameterize these bounds with the substitutions $(i - 3/2)/\sqrt{2} = k\sqrt{\lambda}$ and $i - 3/2 = k\sqrt{\lambda}$ respec-

tively. This gives

$$Q_\lambda(\lceil m + k\sqrt{2\lambda} + 3/2\rceil) \le a_\lambda d(k, \lambda)\Phi(k)$$
$$T_\lambda(\lfloor m - k\sqrt{\lambda} - 3/2\rfloor) \le b_\lambda \Phi(k)$$

where

$$d(k, \lambda) = 1/(1 - \exp(-(2/9)[k\sqrt{2\lambda} + 3/2]))$$

and $T_\lambda(j) = 0$ for $j < 0$. For $\lambda \ge 25$ and $k \ge 3$, we get $d(k, \lambda) \le 1.007$.

From Abramowitz and Stegun [1, page 932], we get

PROPOSITION 4. If $x > 0$, then $\Phi(x) \le \phi(x)/x$ with error less than $\phi(x)/x^3$.

Apply proposition 4 to Glynn's reparameterized bounds to get

COROLLARY 1. If $\lambda \ge 2$ and $1/2\sqrt{2}\lambda \le k \le \sqrt{\lambda}/2\sqrt{2}$, then

$$Q_\lambda(\lceil m + k\sqrt{2\lambda} + 3/2\rceil) \le a_\lambda d(k, \lambda) e^{-k^2/2}/k\sqrt{2\pi}.$$

COROLLARY 2. If $\lambda \ge 2$ and $k \ge 1/\sqrt{2\lambda}$, then

$$T_\lambda(\lfloor m - k\sqrt{\lambda} - 3/2\rfloor) \le b_\lambda e^{-k^2/2}/k\sqrt{2\pi}.$$

Corollary 1 does not contradict the fact that, for large enough truncation points, the mass in the right Poisson tail is an order of magnitude greater than the mass in the corresponding normal tail. In corollary 1, the truncation point is at most $\lceil m + \lambda/2 + 3/2\rceil$.

5. BOUNDING POISSON PROBABILITIES

We bound the Poisson probabilities $p_\lambda(i)$ from below to guarantee that, properly scaled, they do not underflow for $L_\lambda \le i \le R_\lambda$. By the monotonicity of $p_\lambda(i)$ to the left and to the right of $m = \lfloor\lambda\rfloor$, it suffices to check only $p_\lambda(L_\lambda)$ and $p_\lambda(R_\lambda)$. The programs Finder and Weighter use corollaries 3 and 4 below only for $\lambda \ge 25$. For $0 < \lambda < 25$, we set $L_\lambda = 0$ and $R_\lambda = R_{400}$. The latter is justified since the mass in the right tail decreases with λ. Weighter checks that $R_{400} \le 600$; for ε corresponding to $k = 7$, $R_{400} = 600$. It then assures that properly-scaled probabilities do not underflow, resetting R_λ if necessary. The error bound is then $\varepsilon/2 + 10^{10}(R_{400} - R_\lambda)\tau/\Omega \le \varepsilon/2 + 6 \times 10^{12}\tau/\Omega$. The second term is negligible when $\varepsilon \gg 10^{12}\tau/\Omega$, which holds for $\varepsilon = 10^{-10}$ and the computers considered in section 3.

Let

$$c_m = (1/\sqrt{2\pi m})\exp(m - \lambda - 1/12m).$$

According to Feller [3, page 54], the following bound supplements Stirling's formula:

$$n! < \sqrt{2\pi n}\, n^n e^{-n} e^{1/12n}.$$

It readily follows that

$$p_\lambda(m) \ge c_m.$$

Section 6 proves

PROPOSITION 5. For $i > 0$,

$$p(m + i) \ge p_\lambda(m)\exp(-i(i + 1)/2\lambda)$$
$$\ge c_m \exp(-(i + 1)^2/2\lambda).$$

PROPOSITION 6. For $0 < i \le \lambda/2$,

(i) $$p_\lambda(m - i) \ge p_\lambda(m)\exp\left(\frac{-i(i-1)}{2\lambda} - \frac{i(i-1)(2i-1)}{6\lambda^2}\right)$$
$$\ge c_m \exp\left(\frac{-i^2}{2\lambda} - \frac{i^3}{3\lambda^2}\right).$$

(ii) For $0 < i \le m$,

$$p_\lambda(m - i) \ge c_m\left[1 - \frac{i}{m+1}\right]^i.$$

COROLLARY 3. Let $\hat{k} = k\sqrt{2} + 3/2\sqrt{\lambda}$. Then for $k > 0$,

$$p_\lambda(\lfloor m + k\sqrt{2\lambda} + 3/2\rfloor) \ge p_\lambda(\lfloor m + \hat{k}\sqrt{\lambda}\rfloor)$$
$$\ge c_m \exp(-(\hat{k} + 1)^2/2).$$

COROLLARY 4. Let $\tilde{k} = k + 3/2\sqrt{\lambda}$.

(i) For $0 < \tilde{k} \le \sqrt{\lambda}/2$,

$$p_\lambda(\lceil m - k\sqrt{\lambda} - 3/2\rceil) = p_\lambda(\lceil m - \tilde{k}\sqrt{\lambda}\rceil)$$
$$\ge c_m \exp(-\tilde{k}^2/2 - \tilde{k}^3/3\sqrt{\lambda}).$$

(ii) For $\tilde{k} \le (\sqrt{m+1})/m$,

$$p_\lambda(\lceil m - k\sqrt{\lambda} - 3/2\rceil) \ge p_\lambda(\lceil m - \tilde{k}\sqrt{m+1}\rceil)$$
$$\ge c_m\left(1 - \frac{\tilde{k}}{\sqrt{m+1}}\right)^{\tilde{k}\sqrt{m+1}}$$

(iii) For $\tilde{k} \le (\sqrt{m+1})/m$,

$$p_\lambda(\lceil m - k\sqrt{\lambda} - 3/2\rceil) \ge p_\lambda(0) = e^{-\lambda}.$$

We suggest using (i) when applicable; the bound is then at least $c_m\exp(-2\tilde{k}^2/3)$. If only (ii) and (iii) apply, compute both bounds and use the maximum. Since for m large

$$\left(1 - \frac{\tilde{k}}{\sqrt{m+1}}\right)^{\tilde{k}\sqrt{m+1}} \sim e^{-\tilde{k}^2}, \qquad (*)$$

computing the left side is numerically stable. For example, with $m = 63$ and $\tilde{k} = 7$, (i) does not apply and

$$\left(1 - \frac{7}{8}\right)^{56} \doteq 2.6 \times 10^{-51} \quad \text{[see (ii)]}$$
$$e^{-49} \doteq 5.2 \times 10^{-22} \quad \text{[see (*)]}$$
$$e^{-63} \doteq 4.4 \times 10^{-28} \quad \text{[see (iii)]}.$$

Convergence in (*) is glacial.

For $\lambda \ge 25$, we get

$$c_m \ge 1/5e\sqrt{2\pi m} \ge 0.02935/\sqrt{m}.$$

6. PROOFS OF PROPOSITIONS FIVE AND SIX

We use the following known facts:

$$\log(1 + x) \le x \qquad \text{for } x \ge 0$$
$$\log(1 - x) \le -x - x^2 \qquad \text{for } 0 \le x \le 1/2.$$

Right of mode:

$$p(m+i) = p(m)\exp\left(-\sum_{k=1}^{i} \log[(m+k)/\lambda]\right)$$

$$\geq p(m)\exp\left(-\sum_{k=1}^{i}\log[1+k/\lambda]\right)$$

$$\geq p(m)\exp\left(-\sum_{k=1}^{i}\log[1+k/\lambda]\right)$$

$$\geq p(m)\exp\left(-\sum_{k=1}^{i}k/\lambda\right)$$

$$= p(m)\exp(-i(i+1)/2\lambda).$$

Left of mode:

$$p(m-i) = p(m)\exp\left(\sum_{k-1}^{i}\log[(m-k+1)/\lambda]\right)$$

$$\geq p(m)\exp\left(\sum_{k=0}^{i-1}\log[1-k/\lambda]\right)$$

$$\geq p(m)\exp\left(-\sum_{k=0}^{i-1}\left(\frac{k}{\lambda}+\frac{k^2}{\lambda^2}\right)\right)$$

$$\geq p(m)\exp\left(-\frac{i(i-1)}{2\lambda}-\frac{i(i-1)(2i-1)}{6\lambda^2}\right).$$

$$p(m-i) \geq p(m)\left(\frac{m-i+1}{\lambda}\right)^i$$

$$\geq c_m\left(\frac{m-i+1}{m+1}\right)^i$$

$$= c_m\left[1-\frac{i}{m+1}\right]^i.$$

7. CONCLUDING REMARKS

Our bounds probably can be tightened by more intricate analysis. As they stand, they seem good enough for practical purposes. There are similar tradeoffs between complexity and tightness of error bounds for other discrete distributions such as the binomial and hypergeometric distributions. Finding good tradeoffs for such distributions is a subject for future research. Given appropriate bounds, a good strategy to compute the set of individual, nonnegligible probabilities from these distributions are similar to our strategy for the Poisson distribution:

1. choose an appropriate weight for the mode or the mean
2. find appropriate truncation points L and R from upper bounds on the tails, possibly using a counterpart to *special*
3. find lower bounds on the individual probabilities and check for underflow at L and at R
4. compute weights recursively outwards to L and to R
5. compute the total weight by adding smallest terms first, i.e., inwards.

In this paper we focused on the Poisson distribution because it is most relevant to our interests (see 1.1) among the discrete distributions for which the set of nonnegligible individual probabilities is nontrivial to compute.

REFERENCES

1. Abramowitz, M., and Stegun, I.E. *Handbook of Mathematical Functions*. U.S. Dept. of Commerce, National Bureau of Standards Appl. Math. Series #55, 1972.
2. Bratley, P., Fox, B.L., and Schrage, L.E. *A Guide to Simulation*. 2nd ed. Springer-Verlag, New York, 1987.
3. Feller, W. *An Introduction to Probability Theory and Its Applications, 1*. Wiley, New York, 1968.
4. Fox. B.L. Numerical methods for transient Markov chains. *Technical report*, Cornell University, 1987.
5. Fox, B.L. and Glynn, P.W. Conditional confidence intervals. *Technical report*, Cornell University, 1988.
6. Gill, P.E., Murray, W., and Wright, M.H. *Practical Optimization*. Academic Press, London, 1981.
7. Glynn, P.W. Upper bounds on Poisson tail probabilities. *Operations Research Letters 6*, 1 (March, 1987), 9–14.
8. Gross, D., and Harris, C.M. *Fundamentals of Queueing Theory*. Wiley, New York, 1985.
9. Gross, D., and Miller, D.R. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research 32*, 2 (March–April, 1984), 343–361.
10. Knüsel, L. Computation of the chi-square and Poisson distribution. *SIAM J. Sci. Stat. Comput. 7*, 3 (July, 1986), 1022–1036.

CR Categories and Subject Descriptors: F.2.1 [**Numerical Algorithms and Problems**]; G.3 [**Probability and Statistics**]

General Terms: Algorithms, Bounds

Additional Key Words and Phrases: Overflow, Poisson probabilities, truncation, underflow, variate generation

Received 5/86; revised 1/87; accepted 4/87

Authors' Present Address: Bennett L. Fox, Department of Computer Science, University of Montreal, C.P. 6128, Station "A", Montreal, Quebec H3C 3J7, Canada; Peter W. Glynn, Department of Operations Research, Stanford University, Stanford, CA 94305-4022.