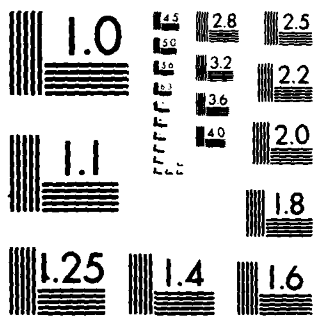
AD-A111 883 PURDUE UNIV LAFAYETTE IN SCHOOL OF INDUSTRIAL ENGINEERING F/G 12/1
POISSON RANDOM VARIATE GENERATION.(U)
DEC 81 B SCHMEISER, V KACHITVICHYANUKUL N00014-79-C-0832
UNCLASSIFIED RR-81-4 NL

END
DATE
FILMED
4-82
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

ADA111883

POISSON RANDOM VARIATE GENERATION

Bruce Schmeiser

Voratas Kachitvichyanukul

School of Industrial Engineering
Purdue University
West Lafayette, Indiana 47907

Research Memorandum 81-4

August 1981

(Revised December 1981)

This research was supported by Office of Naval Research Contract N00014-79-C-0832.

ABSTRACT

Approximate algorithms have long been the only available methods for generating Poisson random variates when the mean is large. A new algorithm is developed which is exact, has execution time which is insensitive to the value of the mean, and is valid whenever the mean is greater than ten. This algorithm is compared to the three other algorithms which have been developed recently for generating Poisson variates when the mean is large. Criteria used are set-up time, marginal execution time, memory requirements, and lines of code. New simple tight bounds on Poisson probabilities contribute to the speed of the algorithm, but are useful in a more general context. In addition, a survey of Poisson variate generation algorithms is given.

1. INTRODUCTION

We consider algorithms for generating random variates from the Poisson mass function

$$f_p(x) = e^{-\mu} \mu^x / x! \qquad x=0,1,2,...$$

$$= 0 \qquad \text{elsewhere,}$$

where μ denotes the expected value of the random variable X. In Section 2 existing algorithms for Poisson variate generation are surveyed. A new algorithm, PTPE, is developed in Section 3. Computational results are shown in Section 4. The validity of PTPE is discussed in the Appendix.

2. LITERATURE SURVEY

Each of the four fundamental approaches to variate generation: inverse transformation, special rties, composition, and acceptance/rejection, (Schmeiser [18]) has been used as the basis for existing algorithms, which we briefly survey here. U(0,1) is used to denote the uniform distribution over the unit interval.

Probably the most basic approach for generating random variates of any kind is the inverse transformation.

Algorithm PINV

1. Generate $u \sim U(0,1)$, set $x \leftarrow 0$, $p \leftarrow e^{-\mu}$.
2. If $u \leq p$, then return x.
3. Set $x \leftarrow x + 1$, $u \leftarrow u-p$, $p \leftarrow p\mu/x$, and go to 2.

When more than one variate is to be generated for a fixed value of μ, PINV may be modified to save the initial value of p in Step 1 and the cumulative probabilities implicit in Step 3. Either way, the execution time per variate increases proportionally with μ. Fishman [11] developed algorithm PIF which executes in time proportional to $\mu^{1/2}$ by performing the inverse transformation beginning at the mode and searching either increasingly or decreasingly for values of x. To begin the search at the mode, both the cumulative probability $p\{X \leq \mu\}$ and probability $p\{X = \mu\}$ are stored. Fishman stored these probabilities for $\mu=1,2,\ldots,100$ to six decimal places, but the size and accuracy of the table could easily be modified. The cumulative probabilities are calculated recursively as in PINV. Snow [20] suggested explicitly storing the cumulative probabilities and using binary search to determine x. Chen and Asau [7] proposed an index table approach (for any discrete distribution) which searches the cumulative probabilities quickly by beginning near the appropriate value, and Atkinson [5] included an algorithm based on index tables in his computational results.

Special properties have been the basis for several Poisson algorithms. The best known and simplest is based on the exponential inter-event times of the homogeneous Poisson point process.

<u>Algorithm PMUL</u>

1. Set $x \leftarrow 0$, $s \leftarrow 1$, $p \leftarrow e^{-\mu}$.
2. Generate $u \sim U(0,1)$, and set $s \leftarrow su$.
3. If $s \leq p$, then return x.
4. Set $x \leftarrow x + 1$, and go to 2.

As with <u>PINV</u> the execution time increases proportionally with μ and storing the initial value of p for future use is reasonable when the value of μ does not change each time a variate is generated. Note that <u>PINV</u> is faster than <u>PMUL</u> for all values of μ whenever the generation of a U(0,1) variate requires more time than the total time required for a division, subtraction and a storage move. The authors have seen no implementation where <u>PMUL</u> was faster than <u>PINV</u>.

In addition to the inverse transformation methods, composition can be used as the basis for Poisson algorithms. Composition, or probability mixing, is used in variate generation by returning a variate from $f_i(x)$ with probability p_i when $f(x) = \sum_{i=1}^{n} p_i f_i(x)$, where n may be finite or infinite and each $f_i(x)$ is either a discrete probability mass function or a density function. Let I be a Poisson random variable with mean λ, $\lambda \geq \mu$. Then a binomial random variable, arising from I trials, each having probability of success μ/λ, has a Poisson distribution with mean μ. The proof is direct by noting

$$f_p(x) = \sum_{i=x}^{\infty} [e^{-\lambda}\lambda^i/i!][\binom{i}{x}(\mu/\lambda)^x ((\lambda-\mu)/\lambda)^{i-x}] \qquad x=0,1,2,\ldots .$$

The advantage to this composition approach is that $e^{-\mu}$ does not need to be calculated during setup. Usually $\lambda = 1$ (Ahrens and Dieter [2] and Fishman [11]) with the resulting algorithm being used to supply x from the fractional portion of μ when μ is not integer. A reasonable implementation for $\mu \leq 1$ "thins" a Poisson variate with unit mean. Using PINV to generate the variate with a mean of one yields

Algorithm PTH ($\mu \leq 1$)

1. Generate $u \sim U(0,1)$, set $x \leftarrow 0$, $k \leftarrow 0$, $p \leftarrow .367879441171$.
2. If $u \leq p$, then return x.
3. Set $k \leftarrow k+1$, $u \leftarrow u-p$, $p \leftarrow p/k$. Generate $v \sim U(0,1)$.

   If $v \leq \mu$, then set $x \leftarrow x+1$. Go to 2.

Fishman [11] gives the algorithm in a form assuming the cumulative probabilities for $\mu = 1$ are tabled. A similar algorithm can be created by incrementing x in Step 4 of PMUL with probability μ and initializing $p \leftarrow .367879441171 = e^{-1}$ in Step 1. The idea of thinning is related to the result by Bolshev [6] discussed later in this section. Lewis and Shedler [14] have developed an algorithm for nonhomogeneous Poisson point processes which is also related.

Ahrens and Dieter [2] proposed algorithm PG which uses relationships between the Poisson, gamma and binomial distributions to generate Poisson variates in time increasing with $\ln(\mu)$. In their computational results, the execution time is greater than for other algorithms unless the mean is quite large. However, newer algorithms for gamma generation (see, e.g., Cheng [8], Schmeiser and Lal [19]) and binomial generation (see, e.g., Devroye and Naderisamani [10]) make this algorithm more competitive.

Ahrens and Dieter [2] also developed a third algorithm based on composition. In the Ahrens and Dieter algorithm PT, a triangular density is used to return the variate most of the time. The other parts of the distribution are more time consuming but occur infrequently. The execution time increases with $\mu^{1/2}$.

The acceptance/rejection algorithm is the basis for three recent Poisson generation algorithms, all of which have execution times which do not increase (and in fact decrease slightly) as $\mu \to \infty$. The acceptance/rejection algorithm centers on a function $t(x)$ which majorizes $f(x)$, the density function from which variates are to be generated. The density function $r(x) = t(x) / \int_{-\infty}^{\infty} t(y)dy$ is proportional to $t(x)$. The acceptance/rejection algorithm is

1. Generate $x \sim r(x)$.

2. Generate $v \sim U(0,1)$.

3. If $v \leq f(x)/t(x)$, then return with x as the generated variate.

   Otherwise, go to Step 1, thereby rejecting x.

The selection of any function $t(x)$ satisfying $t(x) \geq f(x)$ for all $x \in (-\infty,\infty)$ yields a valid algorithm. Whether the algorithm is good depends upon the speed of performing Step 1, the difficulty in evaluating the ratio in Step 3, and the expected number of iterations required to generate one variate. Atkinson [5] proposes algorithm PA which uses a logistic majorizing function and Devroye [9] proposes algorithm IP which uses a normal majorizing function for the body of the distribution and exponential distribution for the right tail. Algorithm PA uses tabled values for $x!$ for $x=0,1,\ldots,200$. Algorithm IP uses preliminary comparisons to avoid calculating $x!$ so often that when evaluation of $x!$ is required, it is performed explicitly as $x=x(x-1)\ldots(3)(2)$. Ahrens and Dieter [3] develop an algorithm based on a double exponential majorizing function.

Kronmal and Peterson [13] describe the "acceptance/complement" method, which is a composition approach which requires one region to be generated using acceptance/rejection. Set-up time can be reduced by forcing the probability of rejection to be equal to the probability of generating a variate from the second composition region. Ahrens and Dieter [4] develop an acceptance/complement algorithm, KPOISS, based on the normal distribution, that dominates their earlier algorithm in [3].

Four approaches which provide variates which are approximately Poisson have been proposed. Atkinson [5] includes the approach developed in Marsaglia [15] and Norman and Cannon [16] which is based on composition and tabling many values. It inherently requires $P\{X=x\}$ to be truncated, although the amount of truncation may be limited by increasing the table size. This algorithm could be considered when memory is not a problem, a small error is acceptable, and many Poisson variates are to be generated for a fixed value of μ.

The second approximate approach is to use a normal approximation to the distribution. Pak [17] discusses the normal approximation to the distribution of X, $(X + .375)^{1/2}$, and $(X - 1/24)^{1/3}$, where X is the Poisson random variable.

The third approximate approach is to use Walker's [22] alias method. The method requires truncation of the right tail of the distribution, memory requirements increase linearly with the mean, and set-up time is substantial for large values of the mean. An alias algorithm was the fastest method for generating Poisson variates according to Atkinson [5]. This approach could be made exact by using a composition framework to obtain the tail values.

The fourth approximate procedure is based on an exact result by Bolshev [6]: If $(X_1, X_2, \ldots, X_n)$ is a multinomial random vector with parameters γ and $p_i = 1/n$ for $i=1,2,\ldots,n$ and γ is a Poisson random variable with mean $n\mu$, then $X_1, X_2, \ldots, X_n$ are independent Poisson random variables each with mean μ. Tadikamalla [21] suggested using the normal distribution to generate γ, noting that the error can be made arbitrarily small by selecting n large. Despite the constant execution time of generating γ from the normal distribution, the algorithm's execution time as implemented by Tadikamalla increases linearly with μ. However, the existence of a multinomial algorithm with execution time robust to $\gamma = \sum_{i=1}^{n} X_i$ would make the use of Bolshev's result very appealing. Note that Bolshev's result can be used to create an exact algorithm by generating γ by algorithm PA, IP, KPOISS, or PTPE.

3. ALGORITHM PTPE

The Poisson random variate generation algorithm PTPE is developed in this section. Generation of variates is via acceptance/rejection, based on

$$f(x) = \mu^{(y-M)} M!/y! \qquad -0.5 < x < \infty \qquad (1)$$

$$= 0 \qquad \text{elsewhere,}$$

where $M = < \mu >$, $y = < x + .5 >$, and $< s >$ denotes the integer portion of s. The function $f(x)$ is constructed by rescaling the Poisson probability function $f_p(y)$ by the value of the function at the mode M. This specific scaling has three advantages:

1. $f(M) = 1$ for all μ, thereby reducing set-up time.
2. Machine accuracy evaluation of $f(y)$ requires fewer terms of Stirling's approximation than does $f_p(y)$, because the errors in M! and Y! tend to cancel.
3. $f(x)$ is *numerically* stable.

Although details will remain, specification of the majorizing function $t(x)$ and minorizing function $b(x)$ defines the basic structure of the algorithm as shown in Figure A. The majorizing function is

$$t(x) = \begin{cases} k_L \exp[-\lambda_L(x_L - x - .5)] & -\infty \le x \le x_L-.5 \\ (1 + c) - |M - x + .5|/p_1 & x_L-.5 < x \le x_R-.5 \\ c \exp[-\lambda_R(x + .5 - x_R)] & x > x_R-.5 \end{cases} \qquad (2)$$

and the minorizing function is

$$b(x) = \begin{cases} 1 - |M - x + .5|/p_1 & x_L-.5 \le x \le x_R-.5 \\ 0 & \text{elsewhere.} \end{cases} \qquad (3)$$

The constants k_L, λ_L, λ_R, c, p_1, x_L, and x_R are defined as functions of μ in the set-up step of the algorithm. Proposition 1 in the Appendix addresses the validity of $t(x)$ as a majorizing function of $f(x)$.

Figure A about here

Composition based on four regions (subdensities) is used to generate variates from the density function proportional to $t(x)$. Region 1, which is the area under $b(x)$, is triangular with zero probability of rejection. Region 2 contains the two parallelograms which can be generated as uniform variates. Regions 3 and 4 are negative exponential. p_1, p_2, p_3, and p_4 in the set-up step are the cumulative values needed to randomly select the region to be used in each iteration. The probability of selecting each region is proportional to its area.

Algorithm PTPE ($\mu \geq 10$)

Step 0. (Set-up constants as function of μ. Execute whenever the value of μ changes.)

$$M = \langle\mu\rangle, \quad p_1 = \langle 2.195\sqrt{M} - 2.2\rangle + 0.5,$$
$$c = 0.133 + 8.56/(6.83 + \mu),$$
$$x_M = M + 0.5, \quad x_L = x_m - p_1, \; x_R = x_m + p_1,$$
$$a = (\mu - x_L)/\mu, \quad \lambda_L = a(1 + a/2),$$
$$a = (x_R - \mu)/x_R, \quad \lambda_R = a(1 + a/2),$$
$$p_2 = p_1(1 + 2c),$$
$$p_3 = p_2 + (0.109 + 8.25/(10.86 + \mu))/\lambda_L$$
$$p_4 = p_3 + c/\lambda_R.$$

Step 1. (Begin logic to generate next variate. Generate u for selecting the region. If region 1 is selected, generate triangularly distributed variate and return.)

Generate $u \sim U(0,p_4)$, $v \sim U(0,1)$.

If $u > p_1$, go to 2. Otherwise return $y = \langle x_M - p_1 v + u \rangle$.

Step 2. (Region 2. Parallelograms. Check whether Region 2 is used. If so, generate y uniformly in $[x_L - .5,\ x_R - .5]$ and go to Step 5 for acceptance/rejection comparison.)

If $u > p_2$, go to 3.

Otherwise $x = x_L + (u - p_1)/c$,

$v = vc + 1 - |M - x + 0.5|/p_1$.

If $v > 1$, go to 1.

Otherwise set $y = \langle x \rangle$ and go to 5.

Step 3. (Region 3, Left tail)

If $u > p_3$, go to 4.

Otherwise set $y = \langle x_L + \ln(v)/\lambda_L \rangle$,

If $y < 0$, go to 1.

Otherwise set $v = v(u - p_2)\lambda_L$ and go to 5.

Step 4. (Region 4, Right tail)

Set $y = \langle x_R - \ln(v)/\lambda_R \rangle$,

$v = v(u - p_3)\lambda_R$.

<u>Step 5</u>. (Acceptance/Rejection comparison)

5.0 (Test for method of evaluating f(y))

If $M \geq 100$ and $y > 50$, go to 5.2.

5.1 (Evaluate f(y) via the recursive relationship

$f(y)=f(y-1)\mu/y$. Start the search from the mode.)

```
F = 1.0
If M < y,
  Then set I = M and
          Repeat
               I=I+1
               F=Fμ/I
          until I=y.
  Otherwise
          If M > y,
            Then set I=y and
                    Repeat
                         I=I+1
                         F=FI/μ
                    until I=M
          Endif.
Endif
If v > F, go to 1.
Otherwise return y.
```

5.2 (Squeezing, check the value of $\ln$ v against upper and lower bounds of $\ln f(y)$.)

$x = y$.

$q = (\mu-x)/x$.

$U_B = x-\mu+(x+.5)q(1+q(-.5+q/3)) + .00084$.

$A = \ln(v)$.

If $A > U_B$, go to 1.

$D = (x+.5)\, q^4/4$.

If $q < 0$, $D = D/(1+q)$.

If $A < U_B - D - .004$, return y.

5.3 (Perform final acceptance/rejection test by using the expression of $\ln f(y)$ derived from the Stirling's formula.)

If $A > [(M+.5)\ln(M/\mu) + (x+.5)\ln(\mu/x - M + x$

$+ (1./M - 1./x)/12.$

$+ (1./x^3 - 1./M^3)/360.]$ go to 1.

Otherwise return y.

<u>Remark 1</u>.

In Step 1, Region 1 is selected. Since Region 1 lies entirely under $f(x)$, the probability of rejection is zero. Since $u \sim U(0,p_1)$, then $u/p_1 \sim U(0,1)$. The triangularly distributed variates are generated as the sum of two independent uniform variates, denoted w and v. Then

$$y = M + 0.5 + (w+v-1)p_1$$

$$= x_M + wp_1 - (1-v)p_1.$$

Since $v \sim U(0,1)$, then $(1-v)$ is also $U(0,1)$. Replacing w by u/p_1 and $(1-v)$ by v yields the expression used in Step 1.

Remark 2.

In Region 2, x is uniformly distributed between $M-p_1$ and $M+p_1$. Since u is uniformly distributed between p_1 and p_2 in this region, $w = (u-p_1)/(p_2-p_1)$ is $U(0,1)$. From the setup, p_2-p_1 is equal to $2cp_1$. Substituting into $x = x_L + 2wp_1$ yields the expression for x used in Step 2. The expression for v results in $v \sim U(b(x), b(x)+c)$, where $b(x) = 1 - |M-x+.5|/p_1$ is the triangle of Region 1.

Remark 3.

In Step 3, x is the negative of a negative exponential random variate. The upper bound of x is x_L and the mean is $x_L - 1/\lambda_L$. Similarly in Step 4, x is negative exponentially distributed with lower bound x_R and mean $x_R + 1/\lambda_R$.

In Region 3, the accept/reject variate $v \sim U(0,t(x))$, is

$$v = wk_L \exp[-\lambda_L(x_L - x - .5)] \qquad \text{where } w \sim U(0,1).$$

The exponential variate x can be generated as $x = x_L - 0.5 + \ln(v')/\lambda_L$, where $v' \sim U(0,1)$. Then $v = wk_L \exp[-\lambda_L(-\ln(v')/\lambda_L)] = wk_L v'$. Replacing w by $(u-p_2)/(p_3-p_2)$ and (p_3-p_2) by k_L/λ_L yields the result in Step 3. A similar derivation leads to the expression used in Step 4.

Remark 4.

In Step 5.0, a test is made to select the method of evaluating $f(y)$. The criteria used here is based on both M and y. For small values of M and y, direct calculation using the recursive formula is faster than evaluating the bounds derived from the Stirling's formula. Step 5.1 is similar to algorithm PF by Fishman with $f(y)$ in place of $f_p(y)$, but requires no tabled constants. In Step 5.2, a preliminary test is made by comparing $\ln(v)$ against upper and lower bounds of $\ln f(y)$. The expressions of $\ln f(y)$ and its bounds are given in the Appendix.

Remark 5.

The idea underlying the setup for Region 3 is to pass the majorizing function $t_1(x)$ through the point $f(x_L-.5)$ and $f(x_L-1.5)$. This same approach is used in Region 4 using $f(x_R-0.5)$ and $f(x_R+0.5)$. The result is $t_1(x)$ in Figure B.

Figure B about here

This exact set-up requires six logarithms and two exponential operations. These operations are slow and can be avoided. The setup in PTPE uses the majorizing function $t(x)$ as shown in Figure B, which does not require higher order operations. The use of $t(x)$ increases the probability of rejection slightly, but the gain in efficiency by avoiding higher order operations in the setup is significant in the cases where the value of mean μ changes often. That these exponential tails majorize $f(x)$ is proved in the Appendix.

Remark 6.

The expected number of U(0,1) values required to generate a Poisson variate is $2(p_4)(e^{-\mu}\mu^M/M!)$, where M is the integer portion of μ and $p_4 = \int_{-\infty}^{\infty} t(x)dx$, as defined in Step 0. The derivation is straightforward. The expected number of iterations is

$$\int_{-\infty}^{\infty} t(x)dx \Big/ \int_{-\infty}^{\infty} f(x)dx = p_4 \Big/ \left[(M!/e^{-\mu}\mu^M) \int_{-\infty}^{\infty} f_p(x)dx \right]$$

$$= p_4(e^{-\mu}\mu^M/M!).$$

Multiplying by the two U(0,1) values per iteration yields the result.

<u>Remark 7</u>.

All four fundamental concepts are included in <u>PTPE</u>. The overall structure of <u>PTPE</u> is acceptance/rejection. The inverse transformation is used to select the region, to generate uniformly distributed variates for Region 2, and to generate exponentially distributed variates for Regions 3 and 4. The use of four regions is composition. The special property that the sum of two independent U(a,b) random variables has a triangular distribution is used in Region 1.

4. COMPUTATIONAL EXPERIENCE

The four algorithms for which execution time approaches a constant as $\mu \to \infty$, <u>PA</u>, <u>IP</u>, <u>KPOISS</u>, and <u>PTPE</u>, are compared here in terms of setup times, marginal execution times, lines of code, and memory requirements. The Ahrens and Dieter algorithm in [3] is dominated by <u>KPOISS</u> and not discussed here. All four algorithms were implemented in FORTRAN using the MNF compiler on Purdue University's CDC 6500 computer. The uniform (0,1) variates were generated using RANF, which is intrinsic in the MNF compiler.

For each combination of μ and algorithm, four replications of 3000 variates were timed. The execution times shown in Table 1 are the averages of the replication averages and are accurate to within one unit in the last decimal place. The accuracy may also be assessed by comparing the last four lines in the table, for which most of the differences in times are due to random variation rather than to changes in distribution shape.

The marginal execution times, shown under the heading "Fixed Mean" in Table 1, favor PTPE. The execution times for setting up the algorithm and generating one variate, shown under the heading "Incremented Means" in Table 1, were obtained by incrementing μ by 10^{-9} with each variate generated. Because KPOISS requires little more than a square root calculation to set up, it is competitive with PTPE when the mean changes with each variate generated.

Since IP and KPOISS require normal variates, their times are sensitive to the normal variate generation algorithm used. We used algorithm KR (see Kinderman and Ramage [12]) which is the fastest FORTRAN level algorithm available. For those who have a faster assembler language normal generator available, the times for KPOISS and IP would be less. Of course, all four algorithms would be faster if coded in assembler language. Another comment concerns PA. The approximation given by Atkinson [5] for the constant c is $c = .767 - 3.36/\mu$, which is inaccurate when $\mu < 30$. The poor approximation causes the relatively large execution times of PA for small values of μ.

Table 1. Comparison of Algorithms

	Fixed Mean				Incremented Mean			
μ	PTPE	KPOISS[a]	IP[a]	PA	PTPE	KPOISS	IP	PA
10[b]	.33	.38	.66	1.41	.35	.45	1.34	1.78
25	.29	.37	.62	1.03	.50	.45	1.31	1.41
100	.24	.36	.58	.90	.48	.44	1.27	1.27
250	.22	.35	.57	.89	.44	.43	1.26	1.26
1000	.20	.34	.55	.86	.42	.42	1.25	1.24
10,000	.20	.34	.54	.87	.42	.42	1.23	1.24
1,000,000	.20	.34	.54	.87	.41	.41	1.23	1.23
Memory Require-ments	279	309	282	146				
Lines of Code	59	64	64	17				

[a] Using KR for the normal random variates.

[b] Times for KPOISS are for $\mu = 10 + \epsilon$.

The execution times were compared using a slower U(0,1) generator. All times in Table 1 increased by about .1 except for IP which had increases of about .2.

The execution times were also compared using the FTN compiler. For large values of μ, KPOISS required only 56% more time than PTPE (compared to 75% under MNF) for fixed means. For variable means KPOISS was 9% faster than PTPE (compared to 0% under MNF).

Note that several exact algorithms are faster than the four algorithms compared here for small values of the mean (approximately $\mu < 50$).

While the number of lines of FORTRAN code is only a crude measure of the goodness of an algorithm, it can be important both in terms of the effort to implement the algorithm and to verify that the algorithm is working properly. PA, PTPE, KPOISS, and IP required 17, 59, 64 and 64 lines of code, respectively. This does not include the nine lines for the routine used to evaluate $\ln(x!)$ needed by PA nor the 58 lines of the KR normal variate generator used here by IP and KPOISS. Algorithms PA, PTPE, IP, and KPOISS require 146, 279, 282, and 309 words of memory, respectively, again not including required support routines.

REFERENCES

1. Abramowitz, M. and Stegun, I.A., (1964). Handbook of Mathematical Functions, National Bureau of Standard, Applied Mathematics Series, 55.

2. Ahrens, J.H. and Dieter, U. (1974). "Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions", Computing, 12, 233-246.

3. Ahrens, J.H. and Dieter, U. (1980a). "Sampling from Binomial and Poisson Distributions: A Method with Bounded Computation Times", Computing, 25, 193-208.

4. Ahrens, J.H. and Dieter, U. (1980b). "Computer Generation of Poisson Deviates from Modified Normal Distributions", Technical Report, University of Kiel, West Germany.

5. Atkinson, A.C. (1979). "The Computer Generation of Poisson Random Variables", Applied Statistics, 28, 1, 29-35.

6. Bolshev, L.N. (1965). "On a Characterization of the Poisson Distribution and Its Statistical Applications", Theory of Probability and its Applications, 10, 446-456.

7. Chen, H.C. and Asau, Y. (1974). "On Generating Random Variates from an Empirical Distribution", AIIE Transactions, 6, 163-166.

8. Cheng, R.C.H. (1978). "Generating Beta Variates with Non-Integral Shape Parameters", Communications of the ACM, 21, 4, 317-322.

9. Devroye, L. (1980). "The Computer Generation of Poisson Random Variables", Technical Report, McGill University, Canada.

10. Devroye, L. and Naderisamani, A. (1980). "A Binomial Random Variate Generator", Technical Report, McGill University, Canada.

11. Fishman, G.S. (1976). "Sampling from the Poisson Distribution on a Computer", Computing, 17, 147-156.

12. Kinderman, A.J. and Ramage, J.G. (1976). "Computer Generation of Normal Random Variables", Journal of the American Statistical Association, 71, 356, 893-896.

13. Kronmal, R.A. and Peterson, A.V. Jr. (1981). "A Variant of the Acceptance-Rejection Method for Computer Generation of Random Variables", Journal of the American Statistical Association, 76, 374, 446-451.

14. Lewis, P.A.W. and Shedler, G.S. (1979). "Simulation of Nonhomogeneous Poisson Processes by Thinning", Naval Research Logistics Quarterly, 26, 3, 403-413.

15. Marsaglia, G. (1963). "Generating Discrete Random Variables in a Computer", Communications of the ACM, 6, 37-38.

16. Norman, J.E. and Cannon, L.E. (1973). "A Computer Program for the Generation of Random Variables from any Discrete Distribution", Journal of Statistical Computation and Simulation, 1, 331-348.

17. Pak, C.H. (1975). "The Generation of Poisson Random Variates", Journal of the Korean Institute of Industrial Engineering, 1, 1, 87-92.

18. Schmeiser, B.W. (1980). "Random Variate Generation: A Survey", Simulation with Discrete Models: A State of the Art View, ed. Oren, Shub, and Roth, IEEE.

19. Schmeiser, B.W. and Lal, R. (1980). "Squeeze Methods for Generating Gamma Variates", Journal of the American Statistical Association, 75, 371, 679-682.

20. Snow, R.H. (1968). "Algorithm 342: Generator of Random Numbers Satisfying the Poisson Distribution", Communications of the ACM, 11, 12, 819-820.

21. Tadikamalla, P.R. (1979). "A Simple Method for Sampling from the Poisson Distribution", Working paper 365, Graduate School of Business, University of Pittsburgh.

22. Walker, A.J. (1977). "An Efficient Method for Generating Discrete Random Variables with General Distributions", ACM Transactions on Mathematical Software, 3, 253-256.

APPENDIX: PROPERTIES OF b(x) AND t(x)

Four inequalities used in algorithm PTPE are discussed here. Proposition 1 considers $b(x) \leq f(x) \leq t(x)$, which is necessary for the acceptance/rejection parts of PTPE. In addition, in PTPE f(x) is squeezed by upper and lower bounds which are proved valid in Propositions 2 and 3, respectively.

Results 1-3, stated below without proof, are necessary for the proofs of Propositions 1, 2, and 3. All follow from the Taylor series expansion of the logarithm (see e.g., Abramowitz and Stegun [1]).

Result 1. If $a \leq b$, then $\ln(b/a) \geq q + q^2/2$, where $q = (b-a)/b$.

Result 2. For all $a > 0$ and $b > 0$, $\ln(b/a) \leq q - q^2/2 + q^3/3$, where $q = (b-a)/a$.

Result 3. For all $a > 0$ and $b > 0$, $\ln(b/a) \geq q - q^2/2 + q^3/3 - \Delta q^4/4$, where $q = (b-a)/a$ and $\Delta = 1$ if $a \leq b$ and $\Delta = (1+q)^{-1}$ if $a>b$.

Lemma 1 is used in the proof of Proposition 1.

Lemma 1. For all $\mu > 0$,

$$f_{M+1-\varepsilon}(x) \leq f_\mu(x) \leq f_M(x) \qquad \text{if} \quad x=0,1,2,\ldots,M$$

and

$$f_{M+1-\varepsilon}(x) \geq f_\mu(x) \geq f_M(x) \qquad \text{if} \quad x=M,M+1,\ldots,$$

where $M = \langle\mu\rangle$ and $f_\mu(x) = \mu^{x-M} M!/x!$.

__Proof__. The ratio $f_\mu(x)/f_M(x) = (\mu/M)^{x-M}$. Since $M \le \mu$, the right side inequalities follow. Similarly, the left side inequalities follow from $f_{M+1-\epsilon}(x)/f_\mu(x) = ((M+1-\epsilon)/\mu)^{x-M}$. ||

__Proposition 1__. For $\mu \ge 10$ and $x\epsilon\ (-\infty,\infty)$, $b(x) \le f(x) \le t(x)$, where $t(x)$ and $b(x)$ are defined in Equations (2) and (3), $f(x)$ is defined in Equation (1), and specific constants are defined in Step 0 of algorithm __PTPE__.

__Proof__. The proof is trivial for $x\epsilon\ (-\infty,-.5)$, since $f(x) = 0$.

Consider $x\epsilon\ (-.5,x_L-.5)$. Since $x_L > 0$,

$$(x_L/x_L)(x_L/(x_L-1))\ldots(x_L/\langle x+1.5\rangle) \ge 1$$

which implies

$$x_L^{x_L-\langle x+.5\rangle} \ge x_L!/\langle x+.5\rangle!.$$

Then

$$x_L^{x_L-\langle x+.5\rangle} \ge (x_L/\mu)^{(x+.5)-\langle x+.5\rangle}\, x_L!/\langle x+.5\rangle!$$

since $x_L < \mu$ and $(x+.5) \ge \langle x+.5\rangle$.

Direct algebra yields

$$x_L^{x_L-(x+.5)} \ge \mu^{\langle x+.5\rangle-M-(x_L-M)+(x_L-(x+.5))}\, x_L!/\langle x+.5\rangle!$$

which implies

$$(\mu^{x_L-M} M!/x_L!)(x_L/\mu)^{x_L-(x+.5)} \ge \mu^{\langle x+.5\rangle-M} M!/\langle x+.5\rangle!$$

which implies

$$f(x_L)\exp[(x_L-(x+.5))\ln(x_L/\mu)] \ge f(x). \qquad \text{(A-1)}$$

Applying Result 1 to $\ln(x_L/\mu) = -\ln(\mu/x_L)$ yields

$$f(x_L)\ \exp[-\lambda_L(x_L-(x+.5))] \geq f(x), \qquad \text{(A-2)}$$

where $\lambda_L = a_L + a_L^2/2$ with $a_L = (\mu-x_L)/\mu$.

The majorizing function used in the algorithm, valid for $\mu \geq 10$, is

$$k_L\ \exp[-\lambda_L(x_L-(x+.5))] \geq f(x), \qquad \text{(A-3)}$$

where $k_L = .109 + 8.25/(10.86 + \mu)$. Inequality (A-3) requires $f(x_L) \leq k_L$ for all $\mu \geq 10$. Since $k_L \leq M$, Lemma 1 implies that only integer values of μ need be considered. The inequality was numerically verified for $\mu = 10,11,\ldots,10000$. The proof that $f(x_L) \leq k_L$ for $\mu \in [10000,\infty)$ is based on showing $f(x_L) \leq z_1(x_L) \leq z_2(x_L) \leq k_L$, where $z_1(x_L) = \exp[-(x_L-\mu)^2/2\mu]$ and $z_2(x_L) = \exp[-(2.195\ \sqrt{\mu}-3.2)^2/(2\mu)]$. The left inequality is from the normal majorizing function used by Ahrens and Dieter [4] for all $x \leq \langle \mu-1.1484 \rangle$. The center inequality follows from $-(x_L-\mu) = -\langle 2.195\ \sqrt{\mu} -2.2 \rangle^2 \leq -(2.195\ \sqrt{\mu} -3.2)^2$. The right inequality follows from $z_2(10000) = .0964 < \min_\mu k_L = .109$ and that for all $\mu \in [10000,\infty)$, $z_2(x_L)$ is a decreasing function of μ. That $z_2(x_L)$ decreases follows from $d\ \ln z_2(x_L)/d\mu = -3.512\mu^{-3/2} + 5.12\mu^{-2}$ which is negative for all $\mu > 2.1254$.

Similar logic for $x \in (x_R -.5,\infty)$ leads to

$$c\ \exp[-\lambda_R(x+.5-x_R)] \geq f(x), \qquad \text{(A-4)}$$

where $c = .133+8.56/(6.83+\mu)$ for all $\mu \geq 10$.

Now consider $x \in [x_L-.5,x_R-.5]$, for which $b(x) \leq f(x) \leq t(x)$ must be satisfied, where $b(x)= 1 - |M-x+.5|/p_1$ and

$t(x) = (1+c) - |M-x+.5|/p_1$. Again, $\mu \in [10,10000]$ was checked numerically; using $\mu = M+1-\varepsilon$ when $x \leq M$ and $\mu = M$ when $x \geq M$ for $b(x) \leq f(x)$ and $\mu = M$ when $x \leq M$ and $\mu = M+1-\varepsilon$ when $x \geq M$ for $f(x) \leq t(x)$, as indicated by Lemma 1. For $\mu \geq 10{,}000$, consideration of limiting values and the asymptotic value of .133 for c indicates the inequality is satisfied. ||

Lemmas 2, 3, and 4 are needed for the proofs of Propositions 2 and 3, which are upper and lower bounds on f(x), respectively.

Lemma 2. For $M = \langle \mu \rangle$, $(M+.5)\ln(M/\mu) \leq M - \mu$.

Proof. Substituting $x=M/\mu$ into the well-known inequality $\ln x \leq x-1$ and multiplying by M+.5 yields $(M+.5)\ln(M/\mu) \leq (M/\mu)(M-\mu)+(M-\mu)/(2\mu)$. Since $(M-\mu)/(2\mu) \leq 0$ and $0 \leq M/\mu \leq 1$, the result is obtained. ||

Lemma 3. For $\mu \geq \mu^*$ and $M = \langle \mu \rangle$,

$$M - \mu + g(\mu^*) \leq (M+.5)\ln(M/\mu),$$

where $g(\mu^*) = (\langle \mu^* \rangle + .5)[\ln(\mu^*/(\mu^* + .5))] + .5$.

Proof. The proof shows that $g(\mu^*)$ minimizes

$$g(\mu) = \min_{\substack{\mu^* < \mu \\ M=\langle\mu\rangle}} [(M+.5)\ln(M/\mu)-(M-\mu)].$$

First consider $\min_{M \leq \mu < M+1} g(\mu)$. Setting $dg(\mu)/d\mu = 0$ and checking that $d^2g(\mu)/d\mu^2 > 0$ yields $\mu = M+.5$. The problem is now to find the value of M which minimizes $(M+.5)\ln(M/(M+.5))+.5$ subject to $M \geq \langle\mu^*\rangle$. Since the function increases with M, as can be seen graphically or by evaluating derivatives, the optimal value is $M = \langle \mu^* \rangle$. ||

Lemma 4. Consider

$$\delta(M,y) = (M^{-1}-y^{-1})/12 - (M^{-3}-y^{-3})/360 + (M^{-5}-y^{-5})/1260,$$

$$\delta_L(M^*,y^*) = -(12y^*)^{-1} - (360M^{*3})^{-1} - (1260y^{*5})^{-1},$$

and

$$\delta_U(M^*,y^*) = (12M^*)^{-1} + (360y^{*3})^{-1} + (1260M^{*5})^{-1}.$$

If $M \geq M^*$ and $y \geq y^*$, then $\delta_L(M^*,y^*) \leq \delta(M,y) \leq \delta_U(M^*,y^*)$.

Proof. The lower and upper bounds are obtained directly by minimizing term by term for $\delta_L(M^*,y^*)$ and maximizing term by term for $\delta_U(M^*,y^*)$. ||

Proposition 2. Consider

$$U_b = y-\mu+(y+.5)q(1+q(-.5+q/3))+\delta_U(M^*,y^*)$$

where $q = (\mu-y)/y$. If $M \geq M^*$ and $y \geq y^*$, then $U_b \geq \ln f(y)$.

Proof. The proof algebraically simplifies $\ln f(y)$, which is evaluated using Stirling's Formula. Further simplification results from inequalities on relatively insignificant terms.

$$\ln f(y) = (y-M)\ln \mu + \ln M! - \ln y!$$

$$= (y-M)\ln \mu$$

$$+ [(M+.5)\ln M - M + \ln \sqrt{2\pi} + (12M)^{-1}$$

$$- (360M^3)^{-1} + (1260M^5)^{-1} + o(M)^{-7}]$$

$$- [(y+.5)\ln y - y + \ln \sqrt{2\pi} + (12y)^{-1}$$

$$- (360y^3)^{-1} + (1260y^5)^{-1} + o(y)^{-7}]$$

$$= [(y+.5) - (M+.5)]\ln \mu + (M+.5)\ln M - (y+.5)\ln y$$

$$- M + y + \delta(M,y)$$

$$= (M+.5)\ln(M/\mu) + (y+.5)\ln(\mu/y) - M + y + \delta(M,y) \qquad \text{(A-5)}$$

where

$$\delta(M,y) = (M^{-1}-y^{-1})/12 - (M^{-3}-y^{-3})/360 + (M^{-5}-y^{-5})/1260 + o((M-y)^{-7}).$$

Applying Lemma 2 to the first term, Result 2 to $\ln(\mu/y)$ and Lemma 4 to $\delta(M,y)$ in Equation (A-5) yields the result.

<u>Proposition 3</u>. For $M \geq M^*$ and $y \geq y^*$,

$$U_b - D + g(\mu^*) - \delta_U(M^*,y^*) - \delta_L(M^*,y^*) \leq \ln f(y),$$

where $D = (y+.5)q^4\Delta/4$, $\Delta = 1$ if $q > 0$ and $\Delta = (1+q)^{-1}$ if $q < 0$, and $q = (\mu-y)/y$.

<u>Proof</u>. $U_b - D + g(\mu^*) - \delta_U(M^*,y^*) - \delta_L(M^*,y^*)$

$$= [y - \mu + (y+.5)q(1+q(-.5+q/3)) + \delta_U(M^*,y^*)] + g(\mu^*)$$

$$- [(y+.5)q^4\Delta/4] - \delta_U(M^*,y^*) + \delta_L(M^*,y^*)$$

$$= y - \mu + (y+.5)[q-q^2/2+q^3/3-q^4\Delta/4] + g(\mu^*) + \delta_L(M^*,y^*). \qquad \text{(A-6)}$$

From Lemma 3, $g(\mu^*) \leq (M+.5)\ln(M/\mu) - M + \mu$; from Lemma 4, $\delta_L(M^*,y^*) \leq \delta(M,y)$; and from Result 3 applied to $\ln(\mu/y)$, Equation (A-6) is less than $\ln f(y)$.||

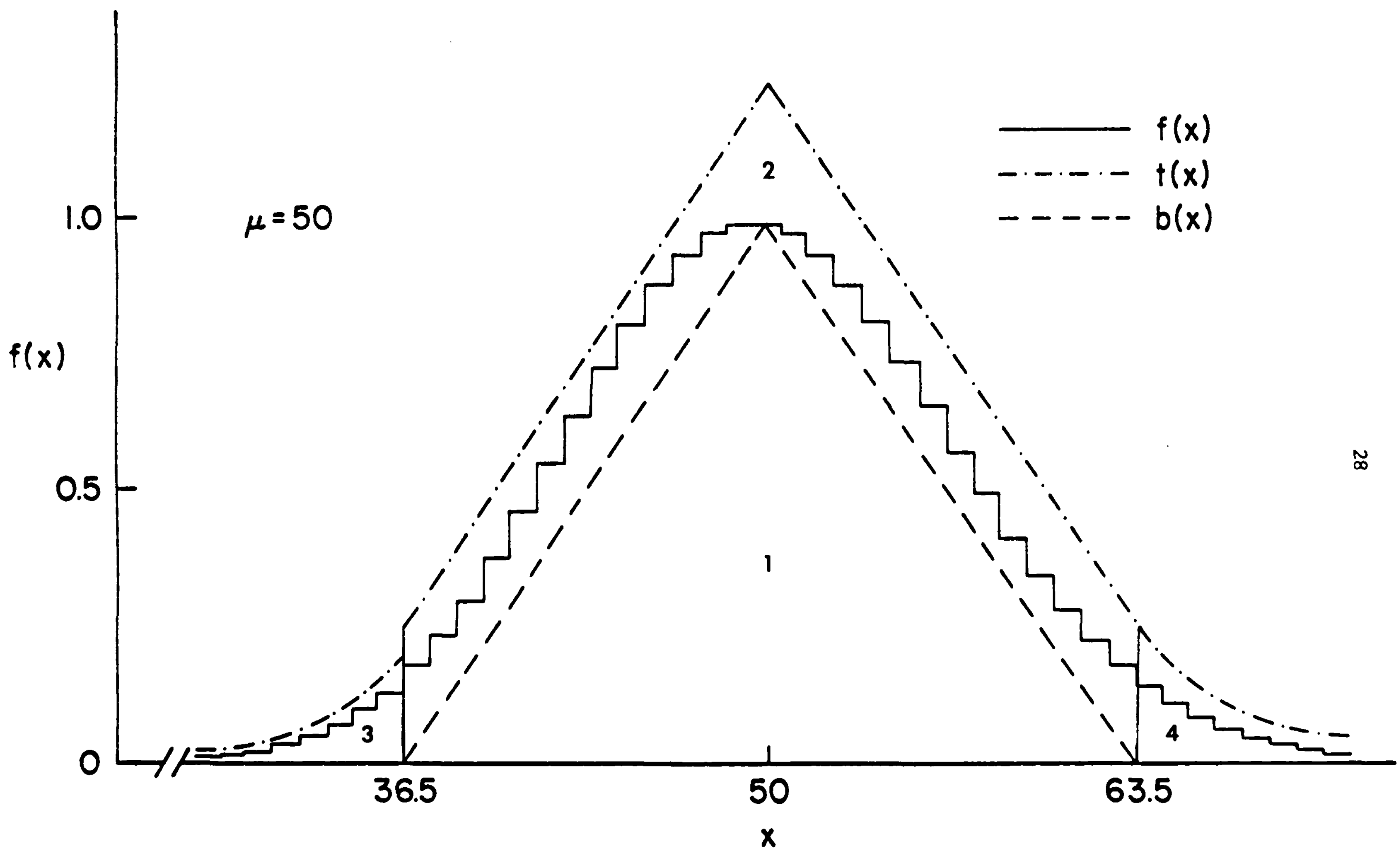


FIGURE A. SET-UP OF ALGORITHM PTPE

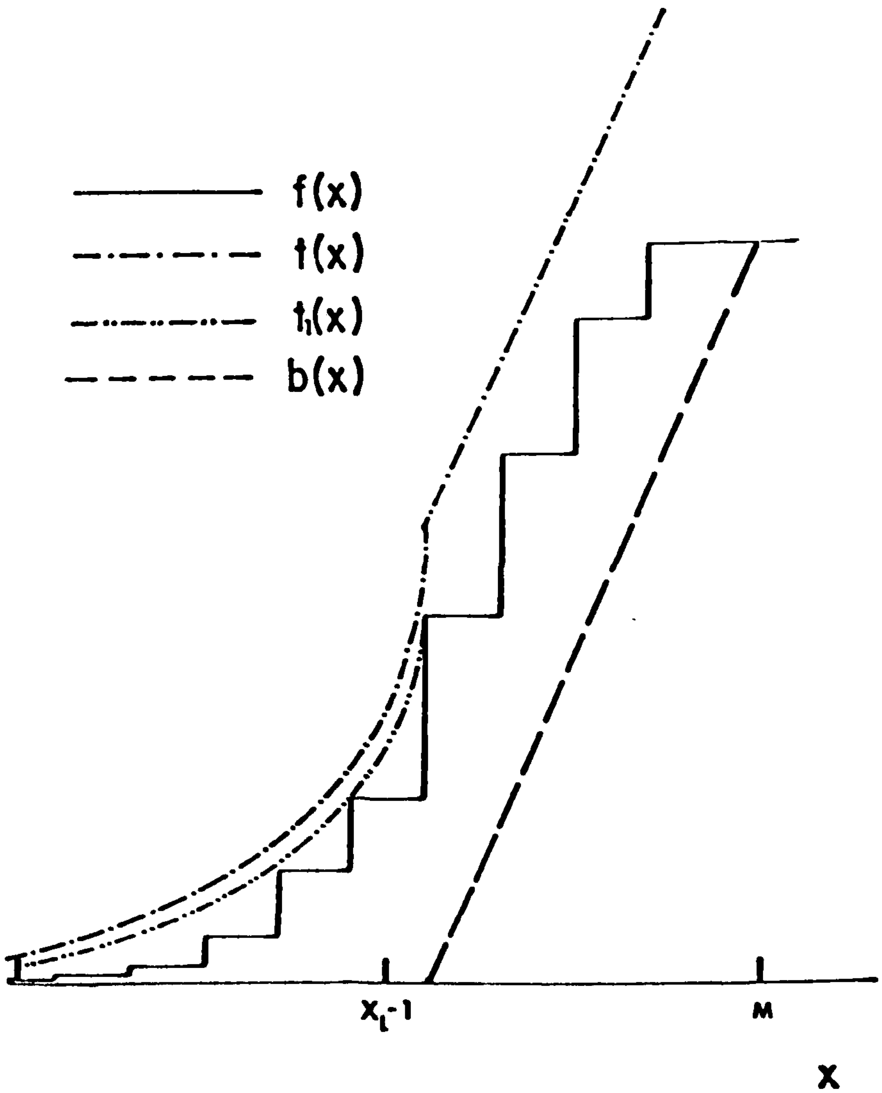


FIGURE B. COMPARISON OF EXACT AND FAST SET-UP.

Computer codes used to obtain the computational results of section 4, "Poisson Random Variate Generation" by Bruce Schmeiser and Voratas Kachitvichyanukul.

```
C     THIS IS THE MAIN PROGRAM TO TEST VARIOUS METHODS
C     OF GENERATING POISSON RANDOM VARIATES
C
C     BRUCE SCHMEISER AND VORATAS KACHITVICHYANUKUL
C     SCHOOL OF INDUSTRIAL ENGINEERING
C     PURDUE UNIVERSITY,      APRIL, 1980
C
C
      DIMENSION NAME(5),XXMU(13)
      DATA NAME/'DUMY','PTPE','KPOISS','IP','PA'/
      DATA XXMU/10.,25.,50.,50.5,100.,250.,500.,1000.,3000.,
     1          5000.,10000.,50000.,1000000./
      N=3000
      ISEED=0
      WRITE(6,1000)
 1000 FORMAT(1H1)
      DO 400 L=1,13
      XMU=XXMU(L)
      II=0
      TMEAN=XXMU(L)
      TVAR=XXMU(L)
      STE=SQRT(TVAR/N)
      WRITE(6,3000) N,TMEAN,TVAR,STE
 3000 FORMAT(3X,'*****************************************************'
     1   /3X,'::::: SAMPLE SIZE  =  'I5,'  :::::'/
     2   /11X,'TIME         MEAN          VARIANCE       STD ERROR'/
     3        ' TRUE               ',3F15.3)
      DO 100 I=1,5
      SUMT=0.0
      DO 150 J=1,4
      SUM=0.0
      SUM2=0.0
      CALL SECOND(T1)
C
      DO 300 K=1,N
      GO TO (1,2,3,4,5),I
    1 CONTINUE
      GO TO 200
    2 CALL PTPE(XMU,ISEED,II)
      GO TO 200
    3 CALL KPOISS(XMU,ISEED,II)
      GO TO 200
    4 CALL IP(XMU,ISEED,II)
      GO TO 200
    5 CALL PA(XMU,ISEED,II)
  200 SUM=SUM+II
      SUM2=SUM2+II*II
  300 CONTINUE
C
      CALL SECOND(T2)
      TIME=1000.*(T2-T1)/N
      SUMT=SUMT+TIME
      AVGT=SUMT/J
      XMEAN=SUM/N
      VAR=SUM2/N-XMEAN*XMEAN
      WRITE(6,2000) NAME(I),TIME,AVGT,XMEAN,VAR
 2000 FORMAT(1X,A5,2F6.3,F15.3,F15.3)
  150 CONTINUE
      PRINT,'    ---------------------------------------------'
  100 CONTINUE
  400 CONTINUE
C
      STOP
      END
```

```
      SUBROUTINE PTPE(XMU,ISEED,JX)
C
C     POISSON RANDOM VARIATE GENERATOR
C        XMU   : MEAN    (XMU .GE. 10)
C        ISEED : RANDOM NUMBER SEED
C        JX    : RANDOMLY GENERATED OBSERVATION
C
C     BRUCE W. SCHMEISER AND VORATAS KACHITVICHYANUKUL
C     PURDUE UNIVERSITY,   SEPTEMBER  1980.
C     REVISED          JULY, 1981
C     METHOD    :  ACCEPTANCE-REJECTION VIA FOUR REGION COMPOSITION
C     AUXILIARY REQUIRED SUBPROGRAM  :
C                UNIFORM (0,1) RANDOM NUMBER GENERATOR
C
      DATA YMU/-1./
      IF(XMU.EQ.YMU) GO TO 2
C
C*****SETUP  (EXECUTE ONLY WHEN XMU CHANGES)
C
      YMU=XMU
      M=YMU
      FM=M
      P1=INT(2.195*SQRT(FM)-2.2)+0.5
      C=.133+8.56/(6.83+YMU)
      XM=M+0.5
      XL=XM-P1
      XR=XM+P1
      AL=(YMU-XL)/YMU
      XLL=AL*(1.+.5*AL)
      AL=(XR-YMU)/XR
      XLR=AL*(1.+.5*AL)
      P2=P1*(1.+C+C)
      P3=P2+(0.109+8.25/(10.86+YMU))/XLL
      P4=P3+C/XLR
C
C*****GENERATE VARIATE
C
    2 U=RANF(ISEED)*P4
      V=RANF(ISEED)
C
C     TRIANGULAR REGION
C
      IF(U.GT.P1)  GO TO 3
      IX=XM-P1*V+U
      GO TO 14
C
C     PARALLELOGRAM REGION
C
    3 IF(U.GT.P2)  GO TO 4
      X=XL+(U-P1)/C
      V=V*C+1.-ABS(FM-X+0.5)/P1
      IF(V.GT.1.)  GO TO 2
      IX=X
      GO TO 6
C
C     LEFT TAIL
C
    4 IF(U.GT.P3)  GO TO 5
      IX=XL+ALOG(V)/XLL
```

```
      IF(IX.LT.0)  GO TO 2
      V=V*(U-P2)*XLL
      GO TO 6
C
C     RIGHT TAIL
C
    5 IX=XR-ALOG(V)/XLR
      V=V*(U-P3)*XLR
C
C*****ACCEPTANCE-REJECTION TEST.  COMPARE  V TO
C     THE SCALED POISSON MASS FUNCTION
C
    6 IF(M.GE.100.AND.IX.GT.50)  GO TO 12
C
C     EXPLICIT EVALUATION
C
      F=1.0
      IF(M-IX) 7,11,9
    7 MP=M+1
      DO 8 I=MP,IX
    8 F=F*YMU/I
      GO TO 11
    9 IX1=IX+1
      DO 10 I=IX1,M
   10 F=F*I/YMU
   11 IF(V-F)  14,14,2
C
C     SQUEEZING USING  UPPER AND LOWER BOUNDS ON ALOG(F(X))
C
   12 X=IX
      Q=(YMU-X)/X
      UB=X-YMU+(X+.5)*Q*(1.+Q*(-.5+.33333333333*Q))+.00084
      ALV=ALOG(V)
      IF(ALV.GT.UB)  GO TO 2
      D=(X+0.5)*.25*(Q*Q)**2
      IF(Q.LT.0.)  D=D/(1.+Q)
      IF(ALV.LT.UB-D-.004)  GO TO 14
C
C     STIRLING'S FORMULA TO MACHINE ACCURACY FOR
C     THE FINAL ACCEPTANCE/REJECTION TEST
C
      IF(ALV.GT.(FM+.5)*ALOG(FM/YMU)+(X+.5)*ALOG(YMU/X)-FM+X
     1     +(1./FM-1./X)/12.+.0027777777778/(X*X*X)
     2    -.00277777778/(FM*FM*FM))  GO TO 2
   14 JX=IX
      RETURN
      END
```

```
      SUBROUTINE KPOISS(A,IR,KPOIS)
C
C     J. H. AHRENS AND U. DIETER
C
C     COMPUTER GENERATION OF POISSON DEVIATES FROM
C     MODIFIED NORMAL DISTRIBUTIONS
C
      DIMENSION FACT(10),PP(35)
      DATA AA,AAA,A0,A1,A2,A3,A4,A5,A6,A7 /0.,0.,-.5,.33333333,
     1   -.2500068,.2000118,-.1661269,.1421878,-.1384794,.125006/
      DATA FACT /1.,1.,2.,6.,24.,120.,720.,5040.,40320.,362880./
      IF(A.EQ.AA)  GO TO 1
      IF(A.LT.10.0)  GO TO 12
      AA=A
      S=SQRT(A)
      D=6.0*A*A
      L=INT(A-1.1484)
    1 CALL NORMAL(IR,TT)
      G=A+S*TT
      IF(G.LT.0.0)  GO TO 2
      KPOIS=INT(G)
      IF(KPOIS.GE.L)  RETURN
      FK=FLOAT(KPOIS)
      AK=A-FK
      U=RANF(IR)
      IF(D*U.GE.AK*AK*AK)  RETURN
    2 IF(A.EQ.AAA)  GO TO 3
      AAA=A
      OMEGA=.3989423/S
      B1=.4166667E-1/A
      B2=.3*B1*B1
      C3=.1428571*B1*B2
      C2=B2-15.*C3
      C1=B1-6.*B2+45.*C3
      C0=1.-B1+3.*B2-15.*C3
      C=.1069/A
    3 IF(G.LT.0.0)  GO TO 5
      KFLAG=0
      GO TO 7
    4 IF(FY-U*FY.LE.PY*EXP(PX-FX))  RETURN
    5 E=-ALOG(RANF(IR))
      U=RANF(IR)
      U=U+U-1.0
      T=1.8+SIGN(E,U)
      IF(T.LE.-0.6744)   GO TO 5
      KPOIS=INT(A+S*T)
      FK=FLOAT(KPOIS)
      AK=A-FK
      KFLAG=1
      GO TO 7
    6 IF(C*ABS(U).GT.PY*EXP(PX+E)-FY*EXP(FX+E))  GO TO 5
      RETURN
    7 IF(KPOIS.GE.10)  GO TO 8
      PX=-A
      PY=A**KPOIS/FACT(KPOIS+1)
      GO TO 11
    8 DEL=.8333333E-1/FK
      DEL=DEL-4.8*DEL*DEL*DEL
      V=AK/FK
```

```
      IF(ABS(U).LE.0.25)   GO TO 9
      PX=FK*ALOG(1.0+U)-AK-DEL
      GO TO 10
    9 PX=FK*U*U*(((((((A7*U+A6)*U+A5)*U+A4)*U+A3)*U+A2)*U+A1)*U+A0)-DEL
   10 PY=.3989423/SQRT(FK)
   11 X=(0.5-AK)/S
      XX=X*X
      FX=-0.5*XX
      FY=OMEGA*(((C3*XX+C2)*XX+C1)*XX+C0)
      IF(KFLAG)  4,4,6
   12 AA=0.0
      IF(A.EQ.AAA)  GO TO 13
      AAA=A
      M=MAX0(1,INT(A))
      L=0
      P=EXP(-A)
      Q=P
      P0=P
   13 U=RANF(IR)
      KPOIS=0
      IF(U.LE.P0)  RETURN
      IF(L.EQ.0)  GO TO 15
      J=1
      IF(U.GT.0.458)  J=MIN0(L,M)
      DO 14 KPOIS=J,L
      IF( U.LE.PP(KPOIS))   RETURN
   14 CONTINUE
      IF(L.EQ.35)   GO TO 13
   15 L=L+1
      DO 16 KPOIS=L,35
      P=P*A/FLOAT(KPOIS)
      Q=Q+P
      PP(KPOIS)=Q
      IF(U.LE.Q)  GO TO 17
   16 CONTINUE
      L=35
      GO TO 13
   17 L=KPOIS
      RETURN
      END
```

```
      SUBROUTINE IP(SL,ISEED,IX)
C
C     THE SUBPROGRAM IP PRODUCES RANDOM POISSON VARIATES WITH
C     PARAMETER SL>0. A REJECTION METHOD WITH SQUEEZING IS USED.
C
C     BY LUC DEVROYE, MCGILL UNIVERSITY, CANADA 1980
C
C     AUXILIARY SUBPROGRAMS REQUIRED :
C
C      UNIFORM (0,1) RANDOM NUMBER GENERATOR
C      STANDARD NORMAL RANDOM NUMBER GENERATOR
C      EXPONENTIAL RANDOM NUMBER GENERATOR
C
      DATA L/0/,SLM,D,D2,D3,STDEV,PTAIL,PBODY,CON,RL,RI,TWO,EL/12*0./
C
C     SETUP
C
      IF(SL.EQ.SLM)  GO TO 10
      L=INT(SL)
      RL=FLOAT(L)
      RI=1./RL
      EL=EXP(RL-SL)
      TWO=RL+RL
      D=SQRT(RL*ALOG(1.+10.18593*RL))
      D2=D+TWO
      D3=D2/D
      STDEV=SQRT(0.5*D2)
      PTAIL=D3*EXP(-(D+1.)/D3)
      CON=0.25/TWO
      PBODY=EXP(CON)*SQRT(3.141593*D2)
      SUM=PTAIL+PBODY+1.0
      PBODY=1./SUM
      PTAIL=PBODY+PTAIL/SUM
      SLM=SL
C
   10 IF(L.EQ.0)  GO TO 99
    1 A=0.
      IX=0
      U=RANF(0)
      IF(U.LT.PTAIL)  GO TO 50
      CALL NORMAL(ISEED,R)
      X=R*STDEV-0.5
      IF(X.GT.D.OR.X.LT.-RL)  GO TO 1
      IF(X.GT.0)  GO TO 18
      A=1.0
      X=X-2.
   18 IX=INT(X+1.)
      Y=FLOAT(IX)
      V=ALOG(RANF(ISEED))-0.5*R**2+CON
   19 T=Y*(Y+1.)/TWO
      IF(V.LT.-T.AND.A.EQ.0.)  GO TO 100
      QR=T*(-1.+(Y+Y+1.)*0.1666667*RI)
      QA=QR-T**2*0.3333333/(RL+(Y+1.)*A)
      IF(V.LT.QA)  GO TO 100
      IF(V.GT.QR)  GO TO 1
      RM=RI*(1.-2*A)
      K=-IX-1
      IF(IX.GT.0) K=IX
      PD=1.0
```

```
      S=0.
      DO 20 J=1,K
      S=S+RM
   20 PD=PD*(1.+S)
      IF(U.LT.(2*A-1.)*ALOG(PD))  GO TO 100
      GO TO 1
   50 IF(U.LT.PBODY)  GO TO 100
      X=D-ALOG(RANF(ISEED))*D3
      IX=INT(X+1.)
      Y=FLOAT(IX)
      U=ALOG(RANF(ISEED))-(X+1.)/D3
      IF(U.GT.-Y*(Y+1.)/(TWO+Y)) GO TO 1
      GO TO 19
   99 IX=0
  100 IX=IX+L
      PD=RANF(ISEED)
  110 IF(PD.LT.EL)  RETURN
      IX=IX+1
      PD=PD*RANF(ISEED)
      GO TO 110
      END


      SUBROUTINE PA(XMU,ISEED,IX)
C
C     GENERATE THE POISSON RANDOM VARIATE IX WITH MEAN XMU
C     USING ATKINSON'S ALGORITHM (XMU.GE.10)
C     APPLIED STATISTICS, 28, 1(1979), 29-39
C     ALOG(IX FACTORIAL) VIA STIRLING'S APPROXIMATION
C
      DATA SAVE/-1./
      IF(XMU.EQ.SAVE)  GO TO 100
      SAVE=XMU
      B=1.8137993642/SQRT(XMU)
      A=B*XMU
      C=ALOG(0.767-3.36/XMU)-XMU-ALOG(B)
      AM=ALOG(XMU)
  100 U=RANF(ISEED)
      X=(A-ALOG((1.-U)/U))/B
      IF(X.LE.-0.5)  GO TO 100
      IX=X+0.5
      U=RANF(ISEED)
      D=1.+EXP(A-B*X)
      IF(A-B*X+ALOG(U/(D*D)).LE.C+IX*AM-XLFAC(IX))  RETURN
      GO TO 100
      END


      FUNCTION XLFAC(I)
C
C     FUNCTION TO EVALUATE LOG OF THE FACTORIAL I
C        I .GT. 7 USE STERLING'S APPROXIMATION
C        I .LE. 7 USE TABLE LOOKUP
C
      DIMENSION AL(8)
      DATA AL/0.,0.,0.6931471806,1.791759469,3.178053830,
     1        4.787491743,6.579251212,8.525161361/
      IF(I.GT.7)   GO TO 100
      XLFAC=AL(I+1)
      RETURN
  100 XLFAC=(I+0.5)*ALOG(FLOAT(I))-I+0.08333333333333/I
     1      -0.00277777777777/(I*I*I) + 0.9189385332
      RETURN
      END
```

```
      SUBROUTINE NORMAL(ISEED,X)
C
C        GENERATION OF ONE NORMAL(0,1) VARIATE USING
C        THE ALGORITHM GIVEN BY KINDERMAN AND RAMAGE
C        IN THE JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION 12/76
C
C        CODED BY PETER BONNER AND MODIFIED BY BRUCE SCHMEISER
C        MARCH 1977 AND JUNE 1977 RESPECTIVELY
C
C
      DATA TAIL/2.216035867166471/
      UU=RANF(ISEED)
      IF(UU.GE..884070402298758) GO TO 2
C
C        RETURN TRIANGULAR VARIATE 88 PERCENT OF THE TIME
C
      Y=RANF(ISEED)
      X=TAIL*(1.131131635444180*UU+Y-1.0)
      RETURN
    2 IF(UU.LT..973310954173898) GO TO 4
C
C        TAIL COMPUTATION
C
    3 V=RANF(ISEED)
      W=RANF(ISEED)
      T1=TAIL*TAIL/2.0
      T=T1-ALOG(W)
      IF(V*V*T.GT.T1) GO TO 3
      X=SQRT(2.0*T)
      IF(UU.GE..986655477086949) X=-X
      RETURN
    4 IF(UU.LT..958720824790463) GO TO 6
C
C        FIRST NEARLY LINEAR DENSITY
C
    5 V=RANF(ISEED)
      W=RANF(ISEED)
      Z=V-W
C     LET V= MAX(V,W) AND LET W=MIN(V,W)
      IF(V.GT.W) GO TO 100
      TEMP=V
      V=W
      W=TEMP
  100 T=TAIL-.630834801921960*W
      IF(V.LE..755591531667601) GO TO 9
      DIFF=EXP(-T*T*.5)/2.50662827463100-.180025191068563*
     *     (2.216035867166471-ABS(T))
      IF(ABS(Z)*.034240503750111.LE.DIFF)  GO TO 9
      GO TO 5
    6 IF(UU.LT..911312780288703) GO TO 8
C
C        SECOND NEARLY LINEAR DENSITY
C
    7 V=RANF(ISEED)
      W=RANF(ISEED)
      Z=V-W
C     LET V= MAX(V,W) AND LET W=MIN(V,W)
      IF(V.GT.W) GO TO 101
      TEMP=V
```

```
      V=W
      W=TEMP
  101 T=.479727404222441+1.105473661022070*W
      IF(V.LE..872834976671790) GO TO 9
      DIFF=EXP(-T*T*.5)/2.50662827463100-.180025191068563*
     *      (2.216035867166471-ABS(T))
      IF(ABS(Z)*.049264496373128.LE.DIFF)  GO TO 9
      GO TO 7
C
C        THIRD NEARLY LINEAR DENSITY
C
    8 V=RANF(ISEED)
      W=RANF(ISEED)
      Z=V-W
C     LET V= MAX(V,W) AND LET W=MIN(V,W)
      IF(V.GT.W) GO TO 102
      TEMP=V
      V=W
      W=TEMP
  102 T=.479727404222441-.595507138015940*W
      IF(V.LE..805577924423817) GO TO 9
      DIFF=EXP(-T*T*.5)/2.50662827463100-.180025191068563*
     *      (2.216035867166471-ABS(T))
      IF(ABS(Z)*.053377549506886.LE.DIFF)  GO TO 9
      GO TO 8
    9 X=T
      IF(Z.GE.0.0) X=-X
      RETURN
      END
```

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A111883
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Poisson Random Variate Generation	5. TYPE OF REPORT & PERIOD COVERED Research Memorandum
	6. PERFORMING ORG. REPORT NUMBER 81-4
7. AUTHOR(s) Bruce Schmeiser, Voratas Kachitvichyanukul	8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0832
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Industrial Engineering, Purdue University, West Lafayette, IN 47907	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 042-477
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Code 411, 800 North Quincy Street, Arlington, VA 22217	12. REPORT DATE December 1981
	13. NUMBER OF PAGES 40
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation	Process Generation
Monte Carlo	Poisson
Sampling	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Approximate algorithms have long been the only available methods for generating Poisson random variates when the mean is large. A new algorithm is developed that is exact, has execution time insensitive to the value of the mean, and is valid whenever the mean is greater than ten. This algorithm is compared to the three other algorithms which have been developed recently for generating Poisson variates when the mean is large. Criteria used are set-up time, marginal execution time, memory requirements, and lines of code. New simple tight bounds on Poisson probabilities contribute to the speed of the algorithm, but are useful in a general context. In addition, Poisson variate generation is surveyed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-014-6601

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ILMED

8