Simulation Modeling and Statistical Computing

Richard E. Nance Editor

Binomial Random Variate Generation

VORATAS KACHITVICHYANUKUL and BRUCE W. SCHMEISER

ABSTRACT: Existing binomial random-variate generators are surveyed, and a new generator designed for moderate and large means is developed. The new algorithm, BTPE, has fixed memory requirements and is faster than other such algorithms, both when single, or when many variates are needed.

1. INTRODUCTION

There are many methods for generating random variates from the binomial probability mass function

$$f_B(x) = \binom{n}{x} p^x(1-p)^{n-x} \qquad x = 0, 1, 2, \ldots, n$$
$$= 0 \qquad \text{elsewhere}$$

where n and p are parameters of the distribution satisfying $n \in \{1, 2, \ldots\}$ and $0 \le p \le 1$. A review of existing algorithms is given in Section 2. A new exact, fast algorithm *BTPE* is developed in Section 3. The algorithm *BTPE* is compared with existing algorithms and the computational results are given in Section 4.

2. LITERATURE REVIEW

Before reviewing the state-of-the-art of binomial variate generation, three terms which are used throughout the paper are defined as follows:

a) Set-up time: time required to calculate algorithm parameters as functions of n and p;
b) Marginal execution time: time required to generate one variate without performing the set-up;
c) Uniformly fast algorithm: an algorithm with finite execution time over the range of parameter values considered.

This research was supported by Office of Naval Research Contract N00014-79-C-0832 to Purdue University. The second author received time and resource support from the Department of Operations Research and the C^3 I Program of the Naval Postgraduate School during the 1985–1986 academic year.

The following review of existing algorithms is structured around the four fundamental approaches to variate generation: inverse transformation, special properties, composition, and acceptance/rejection (see Schmeiser [23], [24]). All four approaches have been used as the basis for existing binomial algorithms.

Inverse Transformation.

The inverse transformation of the binomial distribution function can be implemented by using the recursive formula

$$f_B(0) = (1-p)^n$$
$$f_B(x) = f_B(x-1)\,\frac{n-x+1}{x}\,\frac{p}{1-p}$$

for $x = 1, 2, \ldots, n$ as follows:

Algorithm BINV

```
1. Set $q \leftarrow 1 - p$, $s \leftarrow p/q$, $a \leftarrow (n+1)s$, $r \leftarrow q^n$.
2. Generate $u \sim U(0, 1)$, and set $x \leftarrow 0$.
3. If $u \le r$, then return $x$.
4. Set $u \leftarrow u - r$, $x \leftarrow x + 1$, $r \leftarrow ((a/x) - s)r$, go to 3.
```

Here and throughout, $U(0, 1)$ denotes the uniform distribution over the unit interval.

Algorithm *BINV* executes with speed proportional to np and can be improved by exploiting the property that if x is binomial with parameters n and p, then $n - x$ is binomial with parameters n and $1 - p$. Specifically, replace p by $\min(p, 1-p)$ and return x if p is less than ½ and return $n - x$ otherwise. The execution speed is then proportional to n times $\min(p, 1-p)$. In both cases, the setup requires the relatively time-consuming calculation of q^n. Hence, the efficiency can be further improved by saving all the constants in Step 1, and the probabilities r in Step 3 whenever more than one variate is to be generated for a fixed set of n and p.

Two potential problems may arise due to the finite word size of the computer. Firstly, the calculation of q^n may cause underflow when n is only moderately large.

Secondly, the recursive expression for r is a potential source of roundoff error which accumulates, and can become serious as n increases. Fishman [17] implemented this algorithm using double precision for s, a, and r to safeguard against this problem.

The index table approach proposed by Chen and Asau [8], and recently analyzed by Fishman and Moore [18], is a general method that can be used to further improve the marginal execution time of *BINV* by starting the search of the cumulative probabilities near the value to be returned. The cost is the additional memory required for the index table and time to set up index values whenever the parameter values change.

Special Properties.

Special properties have been the basis for three binomial algorithms. All three algorithms are based on the property that the binomial random variable is the sum of n Bernoulli trials, with the probability of success in each trial being p. The most basic Bernoulli method generates n independent $U(0, 1)$ random variates, and returns the numbers that are less than or equal to p.

Algorithm BU

```
1. Set $x \leftarrow 0$, $k \leftarrow 0$.
2. Repeat
       Generate $u \sim U(0, 1)$, $k \leftarrow k + 1$
       If $u \le p$, then set $x \leftarrow x + 1$
   Until $k = n$.
3. Return.
```

The algorithm executes with speed proportional to n and relies on the speed of the $U(0, 1)$ variate generator. Essentially no setup is required.

The geometric method by Devroye [12] avoids the generation of n independent $U(0, 1)$ variates by using the result: If Y_i is geometrically distributed with parameter p, then the smallest integer x for which $\sum_{i=1}^{x+1} y_i > n$ is binomially distributed with parameters n and p.

Algorithm BG

```
1. Set $y \leftarrow 0$, $x \leftarrow 0$, $c \leftarrow \ln(1 - p)$.
2. If $c = 0$, return $x$.
3. Generate $u \sim U(0, 1)$.
4. $y \leftarrow y + \lfloor \ln(u)/c \rfloor + 1$.
5. If $y < n$, set $x \leftarrow x + 1$, and go to 3.
6. Return $x$.
```

where $\lfloor s \rfloor$ denotes the integer portion of s. Note that $\ln(u)$ in Step 4 can be replaced by the negative of an exponential variate which is faster in some cases. The execution time is proportional to np.

Relles [22] proposed algorithm *RBINOM* which avoids direct generation of n independent $U(0, 1)$ random variates by noting that each order statistic of $U(0, 1)$ random variables has a beta distribution. Algorithm *RBINOM* generates the median of $U(0, 1)$ samples of size $\lfloor n/2^i \rfloor$, $i = 0, 1, 2, \ldots, \log_2(n)$, from the symmetric beta $(\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 1)$ when n is odd. The number of $U(0, 1)$ variates less than or equal to p in a sample of size n is counted by generating the median, then recursively generating the number of points between the median and p, which is conditionally binomial with n only half as large. The process is repeated until the remaining n is small and algorithm *BINV* or *BU* can be used. Although Relles used an approximation to generate the symmetric beta variates, *RBINOM* is exact if an exact beta generator, such as *BS* by Ahrens and Dieter [2] is used.

Ahrens and Dieter [2] refined Relles' algorithm to use nonsymmetric beta variates for order statistics other than the median. In particular, they proposed generating the kth order statistics as a beta $(k, n - k + 1)$ random variable but only gave an algorithm which is equivalent to Relles' algorithm by setting $k = n - k + 1$ and use *BS* to generate the symmetric beta variate. A reasonable choice for k is the integer close to the mean np. This choice is implemented in algorithm *RBNM2* in the computational results of Section 4. Both *RBINOM* and *RBNM2* execute with speed proportional to $\ln(n)$ if a uniformly fast beta variate generator is used. Two uniformly fast beta variate generators can be found in Schmeiser and Babu [25], and Cheng [9]. Cheng's algorithm is more applicable here since usually more than one beta variate is needed for each binomial variate.

Composition

The alias method [21, 28], a composition approach, is a source of efficient algorithms for generating random variates from discrete distributions with a finite number of mass points. While no binomial algorithm based on this method has been published, the method can be easily implemented for the binomial distribution as follows:

Algorithm BALIAS

```
Set-up.
1. Calculate the binomial probabilities and store in
   an array of size $n + 1$.
2. Set-up the alias table $A(\cdot)$ and the table of its cut-
   off values $F(\cdot)$, each also stored in an array of size
   $n + 1$ (see Kronmal and Peterson [21]).
To generate one binomial random variate
3. Generate $u \sim U(0, 1)$, set $u \leftarrow u(n + 1)$, $i \leftarrow \lfloor u + 1 \rfloor$
4. If $u > F(i)$, set $x \leftarrow A(i) - 1$, and return $x$.
5. Set $x \leftarrow i - 1$, and return $x$.
```

The marginal execution time of *BALIAS* is uniformly fast, but the setup time and memory requirements are proportional to n. *BALIAS* is not suitable for very large n, or when the parameters change often.

Acceptance/Rejection.

Fishman [16] describes a method for sampling from the binomial distribution by using the acceptance/rejection technique with a uniform majorizing function, and later improves the idea by changing to a Poisson majorizing function [17]. Algorithm *BP*, implemented with the Poisson generator *PIF* [15], executes with speed proportional to np. However, *BP* is uniformly fast if a uniformly fast Poisson algorithm is used. In view of recent advancement in uniformly fast algorithms for Poisson variates by Atkinson [5, 6], Devroye [10], Ahrens and Dieter [4], and Schmeiser and Kachitvichyanukul [26], which all use acceptance/rejection from other density functions, relying on a Poisson majorizing function seems an unnecessary intermediate step.

Ahrens and Dieter [3] developed algorithm *KBP* for sampling from the binomial distribution with uniformly fast computation times for $n \min(p, 1-p)$ greater than 16. *KBP* uses acceptance/rejection with a double exponential majorizing function. The execution time is relatively stable, but no direct comparison with other existing algorithms is given.

Algorithm *LB* by Devroye [11] generates binomial random variates with $p = ½$ and n greater than 1 by using direct rejection from the normal density. Devroye and Naderisamani [13] developed an algorithm *NA*, which is designed for applications in which n and p remain fixed for a considerable number of consecutive calls. The execution times are uniformly fast. Both *LB* and *NA* require a normal variate generator.

3. ALGORITHM *BTPE*

As discussed in section 2, there are several effective algorithms for $n \min(p, 1-p) < 10$. In this section, the binomial random variate generation algorithm *BTPE*, valid for $n \min(p, 1-p) \geq 10$, is developed. *BTPE* (Binomial, Triangle, Parallelogram, Exponential) has structure similar to the Poisson algorithm *PTPE* by Schmeiser and Kachitvichyanukul [26]. The algorithm generates binomial variates via acceptance/rejection based on the function $f(x)$

$$f(x) = \frac{f_B(y)}{f_B(M)} = \frac{M!(n-M)!}{y!(n-y)!}\left(\frac{p}{1-p}\right)^{y-M} \qquad -0.5 \leq x \leq n + 0.5 \tag{1}$$
$$= 0 \qquad \text{elsewhere}$$

where $y = \lfloor x + .5 \rfloor$ and $M = \lfloor np + p \rfloor$. The function $f(x)$ is proportional to the binomial probability function and satisfies $f(M) = 1$ where M is the mode of the distribution. Advantages of using $f(x)$ rather than $f_B(x)$ are discussed in [26].

As shown in Figure 1, the majorizing function used in *BTPE* is

$$\begin{aligned} t(x) &= c \exp[-\lambda_L(x_L - x - .5)] && \text{if } -\infty < x \leq x_L - .5 \\ &= (1 + c) - |M - x + .5|/p_1 && \text{if } x_L - .5 < x \leq x_R - .5 \\ &= c \exp[-\lambda_R(x + .5 - x_R)] && \text{if } x_R - .5 < x < \infty \end{aligned} \tag{2}$$

and the minorizing function is

$$\begin{aligned} b(x) &= 1 - |M - x + .5|/p_1 && \text{if } x_L - .5 \leq x \leq x_R - .5 \\ &= 0 && \text{elsewhere} \end{aligned} \tag{3}$$

The constants c, λ_R, λ_L, and p_1 are functions of n and p, as given in step 0 below. That $b(x) \leq f(x) \leq t(x)$ for all x is established in [19].

As is apparent from Figure 1, composition based on four regions 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 two parallelograms that can be generated as uniform variates. Regions 3 and 4 are negative exponential. The probability of selecting each region is proportional to its area and is a function of n and p.

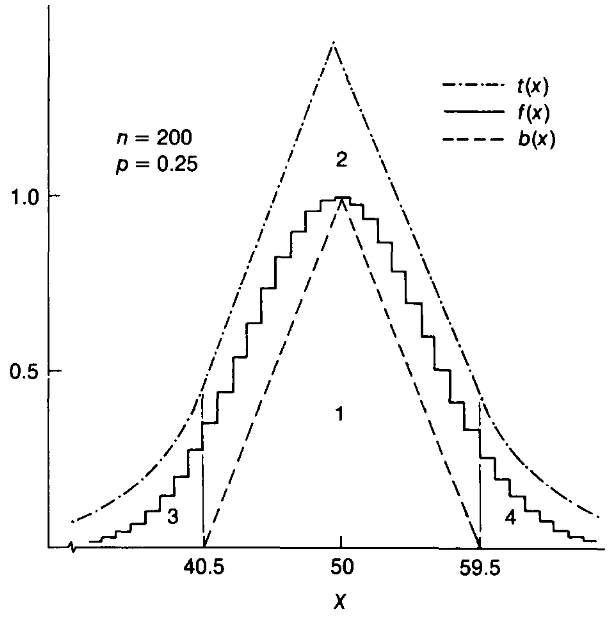


FIGURE 1

The algorithm for $n \min(p, 1-p) \geq 10$ can be implemented as follows:

Algorithm BTPE

```
Step 0. Set-up constants as functions of n and p. Execute
        whenever the value of n or p changes.
  Set
    $r \leftarrow \min(p, 1-p)$, $q \leftarrow 1 - r$, $f_M \leftarrow nr + r$,
    $M \leftarrow \lfloor f_M \rfloor$, $p_1 \leftarrow \lfloor 2.195\sqrt{nrq} - 4.6q \rfloor + 0.5$,
    $x_M \leftarrow M + 0.5$, $x_L \leftarrow x_M - p_1$, $x_R \leftarrow x_M + p_1$,
    $c \leftarrow 0.134 + 20.5/(15.3 + M)$,
    $a \leftarrow (f_M - x_L)/(f_M - x_L r)$, $\lambda_L \leftarrow a(1 + a/2)$,
    $a \leftarrow (x_R - f_M)/(x_R q)$, $\lambda_R \leftarrow a(1 + a/2)$,
    $p_2 \leftarrow p_1(1 + 2c)$,
    $p_3 \leftarrow p_2 + c/\lambda_L$,
    $p_4 \leftarrow p_3 + c/\lambda_R$.
Step 1. Generate $u \sim U(0, p_4)$ for selecting the region. If
        region 1 is selected, generate a triangularly dis-
        tributed variate.
  Generate $u \sim U(0, p_4)$, $v \sim U(0, 1)$.
  If $u > p_1$, go to 2.
  Otherwise
    set $y \leftarrow \lfloor x_M - p_1 v + u \rfloor$, go to 6.
```

```
Step 2.  Region 2, parallelograms. Check if region 2 is
         used. If so, generate $y \sim U(x_L - 0.5, x_R - 0.5)$.
  If $u > p_2$, go to 3.
  Otherwise
    set $x \leftarrow x_L + (u - p_1)/c$,
      $v \leftarrow vc + 1. - |M - x + .5|/p_1$.
      If $v > 1$, go to 1.
      Otherwise
        set $y \leftarrow \lfloor x \rfloor$, go to 5.

Step 3.  Region 3, left exponential tail.
  If $u > p_3$, go to 4.
  Otherwise
    set $y \leftarrow \lfloor x_L + \ln(v)/\lambda_L \rfloor$.
      If $y < 0$, go to 1.
      Otherwise
        set $v \leftarrow v(u - p_2)\lambda_L$, go to 5.

Step 4.  Region 4, right exponential tail.
  Set $y \leftarrow \lfloor x_R - \ln(v)/\lambda_R \rfloor$.
  If $y > n$, go to 1.
  Otherwise set $v \leftarrow v(u - p_3)\lambda_R$.

Step 5.  Acceptance/Rejection Comparison.
  5.0  Test for appropriate method of evaluating $f(y)$.
    Set $k \leftarrow |y - M|$.
    If $k > 20$ and $k < (nrq)/2 - 1$, go to 5.2

  5.1  Evaluate $f(y)$ via the recursive relationship $f(y) =$
       $f(y - 1)(a/x - s)$. Start the search from the mode.
    Set $s \leftarrow r/q$, $a \leftarrow s(n + 1)$, $F \leftarrow 1.0$.
    If $m < y$,
      then set $i \leftarrow M$ and
        Repeat
          $i \leftarrow i + 1$
          $F \leftarrow F(a/i - s)$
        until $i = y$.
      Otherwise
        If $M > y$,
          then set $i \leftarrow y$ and
            Repeat
              $i \leftarrow i + 1$
              $F \leftarrow F/(a/i - s)$
            until $i = M$
        End if
    End if
    If $v > F$, go to 1.
    Otherwise, go to 6.

  5.2  Squeezing. Check the value of $\ln(v)$ against upper
       and lower bound of $\ln f(y)$.
    $\rho \leftarrow (k/(nrq))((k(k/3 + 0.625) + 1/6)/(nrq) + 0.5)$
    $t \leftarrow -k^2/(2(nrq))$
    $A \leftarrow \ln(v)$
    If $A < t - \rho$, go to 6
    If $A > t + \rho$, go to 1.

  5.3  Final Acceptance/Rejection Test.
    $x_1 \leftarrow y + 1$, $f_1 \leftarrow M + 1$, $z \leftarrow n + 1 - M$,
      $w \leftarrow n - y + 1$,
    $x_2 \leftarrow x_1^2$, $f_2 \leftarrow f_1^2$, $z_2 \leftarrow z^2$, $w_2 \leftarrow w^2$.
    If $A > x_M \ln(f_1/x_1) + (n - M + .5)\ln(z/w)$
        $+ (y - M)\ln(wr/x_1 q)$
        $+ (13860. - (462. - (132. - (99.$
              $- 140./f_2)/f_2)/f_2)/f_2)/f_1/166320.$
        $+ (13860. - (462. - (132. - (99.$
              $- 140./z_2)/z_2)/z_2)/z_2)/z/166320.$
        $+ (13860. - (462. - (132. - (99.$
              $- 140./x_2)/x_2)/x_2)/x_2)/x_1/166320.$
        $+ (13860. - (462. - (132. - (99.$
              $- 140./w_2)/w_2)/w_2)/w_2)/w/166320.$
        go to 1.

Step 6.  Check if original probability p is greater than 1/2.
         If so, return $n - y$.
  If $p > 1/2$, set $y \leftarrow n - y$.
  Return.
```

Remark 1. The expected number of iterations for *BTPE* is

$$\int_{-\infty}^{\infty} t(x)\,dx \Big/ \int_{-\infty}^{\infty} f(x)\,dx$$
$$= p_4 \Big/ \int_{-\infty}^{\infty} \frac{f_B(\lfloor x + .5 \rfloor)}{f_B(M)}\,dx$$
$$= p_4 f_B(M) \Big/ \int_{-\infty}^{\infty} f_B(\lfloor x + .5 \rfloor)\,dx$$
$$= p_4 \binom{n}{M} r^M (1 - r)^{n-M}$$

where $p_4 = \int_{-\infty}^{\infty} t(x)\,dx$, M and r are defined in Step 0.

Remark 2. The expected number of $U(0, 1)$ variates required to generate a binomial variate via *BTPE* is $2p_4\binom{n}{M}r^M(1 - r)^{n-M}$, which is obtained by multiplying the expected number of iterations by the two $U(0, 1)$ variates required per iteration.

Remark 3. A major contributing factor to the speed of *BTPE* is the squeeze used in Step 5.2. It is based on the DeMoivre-Laplace limit theorem (see Feller [14]) with tighter bounds being derived in [19]. The numerical constant 20, used to determine whether the exact explicit evaluation or the bounds are used, is somewhat arbitrary, and could be tuned to the computer used, although large changes in execution time will not be obtained.

Remark 4. The expression used in Step 5.3 of the algorithm is obtained by using Stirling's approximation to $\ln f(y)$ [1]. The accuracy of the expression as implemented is beyond the machine accuracy of the CDC 6500 computer.

4. COMPUTATIONAL EXPERIENCE

Algorithm *BTPE*, developed in Section 3, is compared in this section with the following algorithms:

1. Algorithm *KBP* by Ahrens and Dieter [3], with the logic for generating the Poisson variate deleted and generating exponential variates using natural logarithms.

2. Algorithm *NA* by Devroye and Naderisamani [13], implemented with the normal generator *KR* by Kinderman and Ramage [20].
3. Algorithm *RBINOM* by Relles [22], implemented with symmetric beta generator *BS* by Ahrens and Dieter [2].
4. Algorithm *RBNM2*, which is the refined version of *RBINOM* proposed by Ahrens and Dieter [2], implemented with $k = \lfloor np \rfloor$ and the uniformly fast beta generator *BB* by Cheng [9].
5. Algorithm *BP* by Fishman [17], implemented with the uniformly fast Poisson generator *PTPE* by Schmeiser and Kachitvichyanukul [26].
6. Algorithm *BINV* based on the inverse transformation method.
7. Algorithm *BALIAS* based on the alias method as described by Kronmal and Peterson [21].

All the algorithms compared are implemented in FORTRAN using the MNFFTN compiler on Purdue University's CDC 6500 computer with the intrinsic uniform (0, 1) generator *RANF*. The comparison is made for $n \geq 20$ since algorithms *BINV* and *BU* clearly dominate the uniformly fast algorithms for small n.

For each combination of n = 20, 50, 100, 1000, 10000, and 10000000, p = .5, .35, .2, .1, and .000001 and algorithm, four replications of 3000 variates were generated and timed. The execution times shown in Tables I and II are the averages of the replication averages, and are accurate within one or two units of the last digit shown.

The marginal execution times, as shown in Table Ia and IIa, favor *BALIAS*. Among algorithms that require constant memory, the marginal execution times favor *BTPE* by more than a 2 to 1 margin.

The execution times for setting up the algorithm and generating one variate are given in Tables Ib and IIb. The times were obtained by incrementing p by 1×10^{-10} with each call to the algorithm. Incrementing the

TABLE I. Comparison of Uniformly Fast Algorithms

	a. Marginal Execution Times					b. Incremental Times				
n	p	*BTPE*	*KBP*	*NAB*	*BP*	p	*BTPE*	*KBP*	*NAB*	*BP*
20	.5	.54	.65[a]	1.31	1.20	.5	.62	.77[a]	2.70	1.58
	.35	.40[a]	.52[a]	1.29	.96	.35	.51[a]	.63[a]	1.56	1.48
	.2	.29[a]	.37[a]	.86[b]	.77	.2	.40[a]	.49[a]	1.13[b]	1.27
	.1	.22[a]	.27[a]	.58[b]	.67	.1	.33[a]	.38[a]	.84[b]	1.18
	.000001	.14[a]	.18[a]	.29[b]	.56	.000001	.25[a]	.29[a]	.55[b]	1.08
50	.5	.48	1.21	1.08	1.05	.5	.77	1.36	2.46	1.45
	.35	.52	1.26	1.44	.96	.35	.81	1.39	2.69	1.58
	.2	.58	.66[a]	1.35	1.00	.2	.88	.78[a]	2.74	1.52
	.1	.32[a]	.42[a]	1.01[b]	.78	.1	.45[a]	.54[a]	1.28[b]	1.29
	.000001	.14[a]	.18[a]	.30[b]	.56	.000001	.27[a]	.30[a]	.55[b]	1.08
100	.5	.43	1.18	.98	1.05	.5	.73	1.31	2.35	1.44
	.35	.46	1.18	1.06	.94	.35	.75	1.31	2.43	1.56
	.2	.52	1.20	1.20	.87	.2	.79	1.34	2.58	1.52
	.1	.60	.67[a]	1.46	.96	.1	.91	.79[a]	2.83	1.46
	.000001	.14[a]	.18[a]	.30[b]	.57	.000001	.28[a]	.31[a]	.54[b]	1.07
1000	.5	.37	1.12	.81	.96	.5	.66	1.24	2.18	1.35
	.35	.37	1.11	.83	.85	.35	.66	1.24	2.19	1.48
	.2	.39	1.13	.88	.79	.2	.68	1.25	2.26	1.42
	.1	.43	1.14	.96	.77	.1	.70	1.26	2.31	1.40
	.000001	.14[a]	.18[a]	.30[b]	.58	.000001	.32[a]	.36[a]	.55[b]	1.09
10000	.5	.30	1.11	.75	.94	.5	.59	1.22	2.12	1.32
	.35	.30	1.09	.77	.84	.35	.59	1.22	2.13	1.46
	.2	.31	1.09	.78	.77	.2	.60	1.22	2.15	1.42
	.1	.33	1.10	.80	.73	.1	.61	1.23	2.17	1.36
	.000001	.14[a]	.18[a]	.30[b]	.58	.000001	.37[a]	.40[a]	.55[a]	1.07
10000000	.5	.27	1.00	.73	.93	.5	.57	1.11	2.16	1.45
	.35	.27	.99	.73	.84	.35	.56	1.12	2.16	1.38
	.2	.27	1.01	.73	.77	.2	.56	1.12	2.17	1.37
	.1	.27	1.01	.74	.73	.1	.57	1.14	2.16	1.33
	.000001	.62	.67[a]	1.47	.92	.000001	.68	1.23[a]	2.44	1.39
Memory in Words		417	380	411	128					
Lines of Code		89	83	86	22					

[a] Generated via the inverse transformation method.
[b] Generated via the geometric method.

TABLE II. Comparison of Non-uniformly Fast Algorithms

	a. Marginal Execution Times					b. Incremental Times				
n	p	*BINV*	*RBNM2*	*RBINOM*	*BALIAS*	p	*BINV*	*RBNM2*	*RBINOM*	*BALIAS*
20	.5	.47	.58[a]	.58[a]	.11	.5	.58	.58[a]	.58[a]	2.45
	.35	.37	.47[a]	.47[a]	.11	.35	.46	.47[a]	.47[a]	2.46
	.2	.26	.36[a]	.36[a]	.11	.2	.35	.36[a]	.36[a]	2.45
	.1	.19	.29[a]	.29[a]	.11	.1	.29	.29[a]	.29[a]	2.42
	.000001	.11	.21[a]	.21[a]	.11	.000001	.21	.22[a]	.22[a]	2.38
50	.5	1.01	.95	1.10	.11	.5	1.14	.95	1.11	5.64
	.35	.75	.93	1.26	.11	.35	.86	.93	1.27	5.65
	.2	.47	.59[a]	.59[a]	.11	.2	.59	.60[a]	.59[a]	5.63
	.1	.30	.42[a]	.41[a]	.11	.1	.40	.41[a]	.41[a]	5.54
	.000001	.12	.23[a]	.23[a]	.11	.000001	.23	.23[a]	.23[a]	5.50
100	.5	1.93	1.02	1.54	.11	.5	2.05	1.01	1.53	10.87
	.35	1.39	.99	1.47	.11	.35	1.50	.98	1.46	10.85
	.2	.84	.96	1.40	.11	.2	.96	.95	1.39	10.83
	.1	.48	.60[a]	.60[a]	.11	.1	.59	.60[a]	.59[a]	10.72
	.000001	.12	.24[a]	.24[a]	.11	.000001	.24	.24[a]	.24[a]	10.25
Memory in Words		84	112	111	$146 + 3n$					
Lines in Code		22	31	33	64					

[a] Generated via the inverse transformation method.

parameter value to force the setup with each call to the algorithm was easy, since the code invokes the setup automatically whenever the parameter values differ from those of the last call. In addition, such an approach allows the times to be based on exactly the same code as the marginal times. For n min(p, $1 - p$) $>$ 10, *BTPE* dominates all existing algorithms. For smaller values of np, *BINV* and *BU* are more efficient.

As implemented, the inverse transformation method is used in *BTPE* for n min(p, $1 - p$) < 10, and in *KBP* for n min(p, $1 - p$) < 16, whereas algorithm *NA* uses the geometric method when $np \leq 7$. The slight discrepancies in the execution times of the inverse transformation method, used in *BTPE* and *KBP*, are due to the overhead associated with performing preliminary checking.

One comment concerns *BP*. The implementation given by Fishman [17] requires n positions of memory to store the values of ln x!, which prevents *BP* from being implemented for large n. One possible remedy is to use Stirling's formula to evaluate ln x! and make the algorithm more general. Nevertheless, the algorithm is dominated by *BTPE*.

The memory required and the number of lines of code are given in the last two lines of Tables Ia and IIa. Algorithms *BINV*, *RBINOM*, *RBNM2*, *BP*, *KBP*, *NAB*, *BTPE*, and *BALIAS* require 84, 111, 112, 128, 380, 411, 417, and $146 + 3n$ words of memory, respectively. The number of lines of FORTRAN code for *BINV*, *BP*, *RBNM2*, *RBINOM*, *BALIAS*, *KBP*, *NAB*, and *BTPE* are 22, 22, 31, 33, 64, 83, 86, and 89, respectively. All figures shown do not include the memory and lines of code required for the supporting routines such as the normal generator for *NA*, the Poisson generator for *BP*, and the beta generators for *RBINOM* and *RBNM2*.

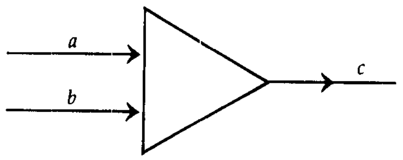
5. SUMMARY

Algorithm *BTPE* for generating binomial random variates when n min(p, $1 - p$) ≥ 10 has been developed and compared to existing algorithms. Among those algorithms where the memory requirements are constant, algorithm *BTPE* dominates all existing algorithms in terms of execution times. For the algorithms compared, *BALIAS* has the best marginal execution time, but has memory requirements and setup time which are functions of n. For small values of the mean, *BINV* is the most efficient. The execution time of *NA* is sensitive to the source of normal random variates, and *BP* is sensitive to the source of Poisson random variates. In the implementation, the inverse transformation algorithm is used for the cases where n min(p, $1 - p$) ≤ 10. The computer program in standard FORTRAN is available from the first author.

BTPE is numerically stable through quite large values of the mean. For a particular user with a particular application, such stability is unnecessary when a normal approximation is acceptable. But an extensive range of parameter values is convenient, in that then the issue of whether the approximation is adequate disappears. Two situations for which the wide range is particularly convenient is first, for algorithms implemented in software libraries where the eventual user and application are unknown and second, when the algorithm is to be called with (possibly random) arguments computed during execution.

For applications where the primary concern is not speed but correlation induction, inversion method implemented with better index search technique is always preferred over algorithms based on other techniques. A FORTRAN implementation of the bucket binary search and the index search can be

It is also possible to use triangular-shaped operator boxes that show the input and outputs:



Finally, users may wish to adapt some symbols from systems theory and represent the network as shown in Figure 4 (e.g., for eq. (5)). Summation is usually shown with the inputs represented as arrows into a circle enclosing a capital sigma. However, the reader should be aware this notation is traditionally intended to represent linear operations, and z often represents a delay-in-time operation. In general, fairly detailed comparisons between networks are useful and can be achieved by a variety of notations, but sometimes at a cost of the loss of an intuitive feeling for the structures. Differences between structures may obscure the similarities. The approach used here is one possible way of simply summarizing topologies.

operations. For certain initial input values, the signal at a particular test node explodes. Insight into the intricacy of these nonlinear systems may be gained from experimentation on the computer. The goal is to describe the behavior of points of the form $z = x + iy$ under recursion of the network. A few simple examples are provided, and more complicated network examples are suggested for future experimentation. Wiggly lines refer to the process of squaring for all the following figures except Figure 5b and c.

The Network-Explorer Graphics System

The user console of the graphics system consists of a vector graphics display (Tektronix 614) and a standard cathode ray tube terminal (IBM 3277 GA). The support software is written in PL/1. A joystick-driven cursor enables the user to magnify and display screen regions interactively. Several system parameters may be entered at the terminal keyboard: picture boundaries (regions of the complex input at a node to be studied), various threshold values for convergence testing, and the number of iterations, n. Any coordinate values of interest on the complex plane may be queried and

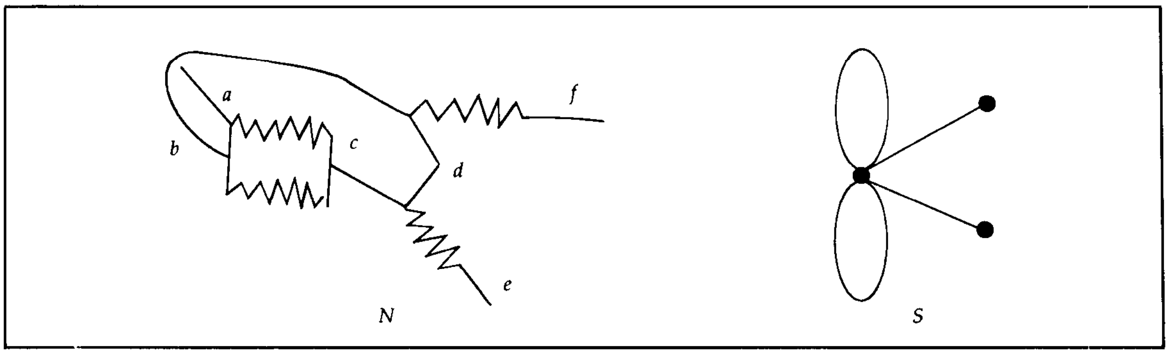


FIGURE 3. Network and Skeleton for $c = (a + b)^2$; $b = c^2$; $e = (c + d)^2$; $f = (b + d)^2$

STABILITY PLOTS FOR SPECIFIC EXAMPLES

Convergence maps, which distinguish regions of stable behavior from divergent, exploding behavior, are often extremely intricate and exhibit fractal characteristics [6, 14, 28, 32]. The propagation of complex input through networks that contain feedback cycles (drawn as rings in S), such as those described earlier, leads to chaotic (nonlinear and irregular) behavior for specific

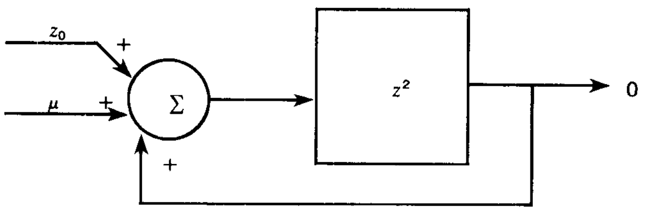


FIGURE 4. Network Representation Adapted from System Theory Symbols

printed simply by pointing on the screen with the joystick-driven cursor. Some of the high-resolution plots were created on a photocomposer with a resolution of 240 dots/in.

Two-Dimensional Stability Plots

Figures 5–8 show graphics for a few simple networks. White regions in the monochrome figures represent those complex points that do not lead to exploding values at the variable node where the network is probed (o). It is sometimes conceptually useful to think of these as regions of the circuit (i.e., a node) that would stay cool, and the energy is stable. The convergence test checks whether the absolute value of the real or imaginary part goes above a certain threshold after n iterations [14, 32]. The use of this test efficiently shows wakes left behind by successive iterations. Figure 9 is a sample gallery of the wide variety of colorful stability plots made possible with the network approach described in this article.

found in Bratley, Fox, and Schrage [7]. When both speed and correlation induction are important, minor modifications to *BTPE* similar to those suggested in Schmeiser and Kachitvichyanukul [27] provide fast algorithm with reasonable correlation induction capability.

REFERENCES

1. Abramowitz, M., and Stegun, I.A. *Handbook of Mathematical Functions*, National Bureau of Standards, Applied Mathematics Series 55, June 1964.
2. Ahrens, J.H., and Dieter, U. Computer methods for sampling from gamma, beta, poisson and binomial distributions, *Computing 12*, (1974), 223–246.
3. Ahrens, J.H., and Dieter, U. Sampling from binomial and poisson distributions: A method with bounded computation times, *Computing 25*, 1980, 193–208.
4. Ahrens, J.H., and Dieter, U. Computer generation of poisson deviates from modified normal distributions, *ACM Transactions of Mathematical Software 8*, 1980, 163–179.
5. Atkinson, A.C. The computer generation of poisson random variables, *Applied Statistics 28*, 1, 1979, 29–35.
6. Atkinson, A.C. Recent developments in the computer generation of poisson random variables, *Applied Statistics 28*, 3, 1979, 260–263.
7. Bratley, P., Fox, B.L., and Schrage, L.E. *A Guide to Simulation*. Springer-Verlag, New York, 1983.
8. Chen, H.C., and Asau, Y. On generating random variates from an empirical distribution, *AIIE Transactions 6*, 1974, 163–166.
9. Cheng, R.C.H. Generating beta variates with non-integral shape parameters, *Communications of the ACM 21*, 4, (April 1978), 317–322.
10. Devroye, L. The computer generation of poisson random variables, *Computing 26*, 1981, 197–207.
11. Devroye, L. "The Computer Generation of Binomial Random Variables," Technical Report, McGill University, Montreal, Quebec, Canada, 1980.
12. Devroye, L. Generating the maximum of independent identically distributed random variables, *Computers and Mathematics with Applications 6*, 1980, 305–315.
13. Devroye, L., and Naderisamani, A. "Binomial Random Variate Generator," Technical Report, McGill University, Montreal, Quebec, Canada, 1980.
14. Feller, W. *An Introduction to Probability Theory and Its Applications*, Volume 1, Wiley, New York, 1968.
15. Fishman, G.S. Sampling from the poisson distribution on a computer, *Computing 17*, 1976, 147–156.
16. Fishman, G.S. *Principles of Discrete Event Simulation*, Wiley, New York, 1978.
17. Fishman, G.S. Sampling from the binomial distribution on a computer, *Journal of the American Statistical Association 74*, 366, 1979, 418–423.
18. Fishman, G.S., and Moore, L.R. Sampling from a discrete distribution while preserving monotonicity, *The American Statistician 38*, 3 1984, 219–223.
19. Kachitvichyanukul, V., and Schmeiser, B.W. "Binomial Random Variate Generation," Technical Report 83-9, Industrial and Management Engineering, The University of Iowa, 1983.
20. Kinderman, A.J., and Ramage, J.G. Computer generation of normal random variables, *Journal of the American Statistical Association 71*, 356, 1976, 893–896.
21. Kronmal, R.A., and Peterson, A.V., Jr. On the alias method for generating random varaibles from a discrete distribution, *American Statistician 33*, 1979, 214–218.
22. Relles, D.A. A simple algorithm for generating binomial random variables when *N* is large, *Journal of the American Statistical Association 67*, 1972, 612–613.
23. Schmeiser, B.W. "Random Variate Generation: A Survey." In *Simulation with Discrete Models: A State-of-the-Art View*, T.I. Oren, C.M. Shub, and P.F. Roth (eds.), In *Proceedings of the 1980 Winter Simulation Conference*, IEEE, 1980, 79–104.
24. Schmeiser, B.W. "Random Variate Generation." In *Proceedings of the 1981 Winter Simulation Conference*, T.I. Oren, C.M. Delfosse, C.S. Shub (eds.). IEEE, 1981, 227–242.
25. Schmeiser, B.W., and Babu, A.J.G. Beta variate generation via exponential majorizing functions, *Operations Research 28*, 4, 1980, 917–926.
26. Schmeiser, B.W., and Kachitvichyanukul, V. "Poisson Random Variate Generation," Research Memorandum 81-4, Purdue University, 1981.
27. Schmeiser, B.W., and Kachitvichyanukul, V. "Correlation Induction withou the Inverse Transformation," In *Proceedings of the 1986 Winter Simulation Conference*, IEEE, 1986, 266–274.
28. Walker, A.J. An efficient method for generating discrete random variables with general distributions, *ACM Transactions on Mathematical Software 3*, 1977, 252–256.

CR Categories and Subject Descriptors: G.3 [**Mathematics of Computing**]: Probability and Statistics—*random number generation, statistical computing*
General Terms: Algorithms
Additional Key Words and Phrases: acceptance–rejection method, binomial distribution, composition, inverse transformation, random numbers, simulation

Received 3/86; revised 12/86; accepted 6/87

Authors' Addresses: Voratas Kachitvichyanukul, Department of Industrial and Management Engineering, The University of Iowa, Iowa City, Iowa 52242; Bruce W. Schmeiser, School of Industrial Engineering, Purdue University, West Lafayette, Indiana 47907.