A Simple Method for Generating Gamma Variables

GEORGE MARSAGLIA
The Florida State University
and
WAI WAN TSANG
The University of Hong Kong

We offer a procedure for generating a gamma variate as the cube of a suitably scaled normal variate. It is fast and simple, assuming one has a fast way to generate normal variables. In brief: generate a normal variate x and a uniform variate U until $\ln(U) < 0.5x^2 + d - dv + d\ln(v)$, then return dv. Here, the gamma parameter is $\alpha \geq 1$, and $v = (1 + x/\sqrt{9d})^3$, with $d = \alpha - 1/3$. The efficiency is high, exceeding 0.951, 0.981, 0.992, 0.996 at $\alpha = 1, 2, 4, 8$. The procedure can be made to run faster by means of a simple squeeze that avoids the two logarithms most of the time: return dv if $U < 1 - 0.0331x^4$. We give a short C program for any $\alpha \geq 1$, and show how to boost an $\alpha < 1$ into an $\alpha > 1$. The gamma procedure is particularly fast for C implementation if the normal variate is generated in-line, via the #define feature. We include such an inline version, based on our ziggurat method. With it, and an inline uniform generator, gamma variates can be produced in 400MHz CPUs at better than 1.3 million per second, with the parameter α changing from call to call.

Categories and Subject Descriptors: G.4 [**Mathematics of Computing**]: Mathematical Software; I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Random number generation, gamma distribution, ziggurat method

1. INTRODUCTION

Along with the normal and exponential, the gamma is one of the most important distributions in probability and statistics. The standard gamma

George Marsaglia's research was supported by the National Science Foundation.
Authors' addresses: G. Marsaglia, Department of Statistics, The Florida State University, Tallahassee, FL 32306; email: geo@stat.fsu.edu; W. W. Tsang, Department of Computer Science and Information Systems, The University of Hong Kong, Pokfulam Road, Hong Kong; email: tsang@csis.hku.hk.

variate has density $y^{\alpha-1}e^{-y}/\Gamma(\alpha)$, $y > 0$. Because simple functions of such gamma variates can easily provide many of the important variates for simulations—chi-square, t, F, beta, Dirichlet—a very fast and simple method for generating gamma variates would be a desirable element in the toolbox of Monte Carlo researchers.

We offer such a one here, based on the assumption that one already has another essential element in that toolbox—a fast procedure for generating standard normal variates x. A gamma variate can then be simply and quickly generated as dv, with $d = \alpha - 1/3$ and $v = (1 + x/\sqrt{9d})^3$:

generate normal x and uniform U

until $\ln(U) < 0.5x^2 + d - dv + d\ln(v)$,

then return dv.

The efficiency of the rejection method exceeds 0.951, 0.973, 0.982, 0.989 for $\alpha = 1, 1.5, 2, 3$, and 0.9971 for $\alpha = 10$ and 0.9997 for $\alpha = 100$. (The method is not feasible for $\alpha < 1$, but it is easy to boost the parameter from α to $\alpha+1$; see the note below.) The two logarithms in the rejection test are fast enough on modern machines that, as it stands, the method is very fast. But with a simple squeeze we can avoid the logarithms most of the time, and make it faster.

Devroye [1986] summarizes many of the methods for generating gamma and other random variables. Of the gamma methods published before or after Devroye's book, we compared those of Ahrens and Dieter [1982], Cheng and Feast [1979], Minh [1988], and Schmeiser and Lal [1980]. We also compared with one based on applying the Monty Python method to transformed gamma densities [Marsaglia and Tsang 1998a; 1998b]. The Monty Python method has nearly the speed of the method we give here, but is more complicated. The Ahrens-Dieter [1982] method is also quite fast when using our inline RNOR, but the algorithm is also very complicated. Exact comparisons depend on the platform, language, compiler, uniform and/or normal generator, etc. And the claim for simplicity is based, of course, on the assumption that one has that desirable element of any Monte Carloist's toolbox: a quick and easy normal generator. Some time comparisons are in Section 7.

2. THE NEW METHOD

We want to generate random variables with the gamma density $y^{\alpha-1}e^{-y}/\Gamma(\alpha)$, with parameter $\alpha \geq 1$. Let

$$h(x) = d(1 + cx)^3$$

for

$$-1/c < x < \infty,$$

with

$$d = \alpha - 1/3,\ c = 1/\sqrt{9d}.$$

If we generate a random variable X with density $h(x)^{\alpha-1}e^{-h(x)}h'(x)/\Gamma(\alpha)$, then $Y = h(X)$ will have the density $y^{\alpha-1}e^{-y}/\Gamma(\alpha)$. This is Marsaglia's exact-approximation method [Marsaglia 1984], which we use here to provide a gamma variate via the third power of a normal variate.

If a very fast normal generator is available, we should get a gamma variate without much more difficulty than that for the normal variate itself. (Every serious Monte Carloist should have a fast normal generator. We provide leads to fast and very fast normal generators in Section 6.)

Since we will be dealing with a simple rejection method, that nuisance constant $\Gamma(\alpha)$ will cancel in comparing functional values, and we need not bother with it from this point on.

After simplifying $h(x)^{\alpha-1}e^{-h(x)}h'(x)$, ignoring constant factors, and putting everything into exponential form, we want to generate a random variable whose density is some normalizing constant times $e^{g(x)}$, where

$$g(x) = d\ln((1 + cx)^3) - d(1 + cx)^3 + d,$$

$$d = \alpha - 1/3,$$

$$c = 1/\sqrt{9d},\ -1/c < x < \infty.$$

As with $\Gamma(\alpha)$, the normalizing constant cancels in our application of the rejection method, and need not be mentioned again. The extra d is put into g to make $e^{g(0)} = 1$.

2.1 Justification

Our method is based on this property of (and the reason for) our choice for $g(x)$: with $\alpha \geq 1$,

$$e^{g(x)} \leq e^{-0.5x^2},\quad -1/c < x < \infty.$$

That is, $e^{g(x)}$ can be put under the (unscaled) normal density $e^{-0.5x^2}$, and furthermore, is quite close to it, occupying some 95.2% of the area at $\alpha = 1$ to 99.7% at $\alpha = 10$. A plot of $e^{g(x)}$ and the dominating $e^{-0.5x^2}$ is in Figure 1, for $\alpha = 1, 2, 4$.

It is evident that the efficiency of a rejection procedure will be quite high. An exact measure of the efficiency is $\int_{-1/c}^{\infty} e^{g(x)}dx / \int_{-\infty}^{\infty} e^{-0.5x^2}dx$, which has (rounded) values 0.95167, 0.98166, 0.99203, 0.99628 for $\alpha = 1, 2, 4, 8$,

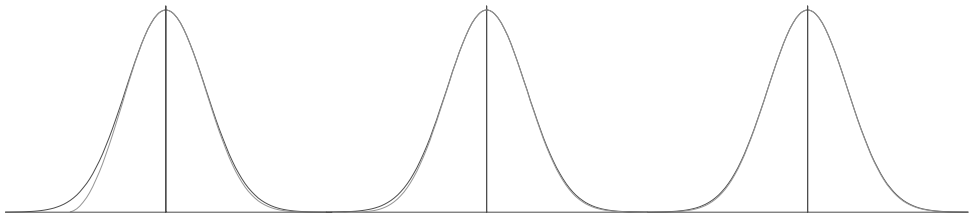

Fig. 1. The densities $e^{g(x)}$ with dominating $e^{-0.5x^2}$ for $\alpha = 1, 2, 4$.

easily obtained to that accuracy by numerical integration, or by expanding the integrand of the numerator and integrating term by term.

Another feature of our choice of g and the dominating normal curve is that we have $Ue^{-0.5x^2} < e^{g(x)}$ if, and only if, $\ln(U) - 0.5x^2 < g(x)$. This avoids the exponential function by using $\ln(U)$ in the rejection method; logarithms are faster than calls to exp(). Furthermore, a very fast exponential variate generator might be used, rather than $\ln(U)$, to make the acceptance test. But the squeeze test developed below virtually eliminates the use of logarithms. Since it requires U, the exponential option is not worth exploring.

To prove that $e^{-0.5x^2} \geq e^{g(x)}$ for $-1/c < x < \infty$, it suffices to prove that $w(x) = -0.5x^2 - g(x) \geq 0$. A little algebra will verify that $(1 + cx)w'(x) = c^2x^3/3$ and $(1 + cx)^2w''(x) = c^2x^2 + 2c^3x^3/3$. Thus $w(x)$ is U-shaped, with minimum zero at $x = 0$.

2.2 Motivation

This paragraph explains the choice of $h(x)$. We begin with a question: what choice of d, c, and k in $h(x) = d(1 + cx)^k$ will make $h(x)^{\alpha-1}h'(x)e^{-h(x)}/\Gamma(\alpha)$ nearly a normal density? With normalizing constants ignored, an answer is provided by the following: choose d, c, and k so that

$$f(x) = e^{g(x)} = e^{(k\alpha-1)\ln(1+cx)-d(1+cx)^k+d} = e^{-\frac{1}{2}x^2+a_3x^3+a_4x^4+\cdots},$$

with a_3 and a_4 small. The choices $d = \alpha - 1/k$, $c = 1/\sqrt{k^2\alpha - k}$ will do it. With those choices, f becomes

$$f(x) = e^{-\frac{1}{2}x^2-\frac{(k-3)c}{6}x^3-\frac{(k^2-6k+11)c^2}{24}x^4+\cdots}.$$

Evidently $k = 3$, $d = \alpha - 1/3$, and $c = 1/\sqrt{9\alpha - 3}$ are good choices, and the ones we use. But $k = 4$ or $k = 8$ are also interesting choices that still make computing $(1 + cx)^k$ easy. The g in the algorithm and used in Figure 1 is based on the choices $k = 3$, $d = \alpha - 1/3$, and $c = 1/\sqrt{9\alpha - 3}$.

3. THE ALGORITHM

Even without squeezing $e^{g(x)}$ from below, we have a simple and fast method for generating a gamma variate with parameter $\alpha \geq 1$:

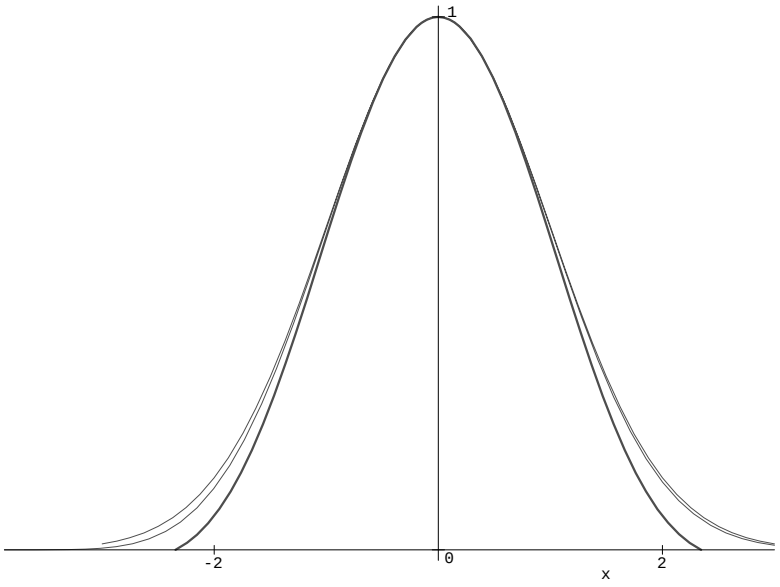

Fig. 2. The squeeze for $\alpha = 2$, $s(x) \le e^{g(x)} \le e^{-0.5x^2}$.

(1) `Setup: d=a-1/3, c=1/sqrt(9d).`

(2) `Generate: v=(1+c*x)^3, with x standard normal.`

(3) `if v>0 and log(UNI) < 0.5*x^2+d-d*v+d*log(v) return d*v.`

(4) `go back to step 2.`

3.1 Squeezes

While the above method is simple and fast, it is worth considering how we might avoid the two logarithms, as they account for a significant part of the average running time. The direct rejection procedure chooses a uniform point $(x, Ue^{-0.5x^2})$ under the curve $e^{-0.5x^2}$, then keeps x if $Ue^{-0.5x^2} < e^{g(x)}$. We can improve on this with the squeeze method of Marsaglia [1977]: find an easy-to-test squeeze function $s(x)$ such that

$$s(x) \le e^{g(x)} \le e^{-0.5x^2}$$

and then accept x if $Ue^{-0.5x^2} \le s(x)$.

We have found a promising squeeze function:

$$s(x) = (1 - 0.0331x^4)e^{-0.5x^2}.$$

In that form $s(x)$ is not very easy to evaluate; however, the exponential parts cancel in the comparison, and only the $(1 - 0.0331x^4)$ part is needed.

Figure 2 shows $s(x) \le e^{g(x)} \le e^{-0.5x^2}$ for $\alpha = 2$. The three curves $s(x)$, $e^{g(x)}$, $e^{-0.5x^2}$ are very close over regions for which a normal x appears most of the time. Note that $s(x)$ is negative for $|x| > r = 0.0331^{-1/4} = 2.344\ldots$; so the squeeze is surely wasted there, but normal x's seldom fall there. The squeeze ratio (area s/area $e^{g(x)}$) for the choice 0.0331 in s is about 0.9638 at $\alpha = 1$, then decreasing with larger alpha—for example, about 0.9199 at $\alpha = 10$, 0.9185 at $\alpha = 20$, and 0.91748 at $\alpha = 100$. The tighter squeeze for low α partly compensates for the lower efficiency there, causing the final algorithm to have nearly constant average time for all $\alpha \ge 1$.

A proof that s is a proper squeeze may be provided by a sequence of steps as follows:

(1) For fixed $\alpha \ge 1$ and for each $x > 0$, $e^{g(-x)} \le e^{g(x)}$. Thus, since $s(x)$ is symmetric, it suffices to show that $s(x) \le e^{g(-x)}$ for $1 \le \alpha$ and $0 < x < r$. (With $w = [(1 - cx)/(1 + cx)]^3$, $e^{g(-x)-g(x)}$ becomes $e^{d[\ln(w)-w]}$, and the exponent is negative for $w < 1$.)

(2) For fixed x in $0 < x < r$, $e^{g(-x)}$ is an increasing function of α for $1 \le \alpha < \infty$. That is, the domination of $e^{g(x)}$ over any purported $s(x)$ gets better as α increases, so the worse case is at $\alpha = 1$. (Substitute $z = x/\sqrt{9\alpha - 3}$ in $e^{g(-x)}$ to get $e^{x^2 h(z)/9}$, with $h(z) = [\ln((1 - z)^3) - (1 - z)^3 + 1]/z^2$, and then note that $h(z)$ is decreasing for $0 < z < 1$.)

(3) Now let $\alpha = 1$ and $D(t) = e^{2t+t^2+2t^3/3+2\ln(1-t)-\ln(1-1.1916t^4)}$, obtained from $e^{g(x)+x^2/2}/(1 - 0.0331x^4)$ by substituting $x = -t/c$ then simplifying to a single exponential form. We may verify that s is a proper squeeze by showing that half the exponent in $D(t)$

$$E(t) = t + t^2/2 + t^3/3 + \ln(1 - t) - \ln(1 - 1.1916t^4)/2,$$

is positive for $0 < t < t_0 = 1.1916^{-0.25} = 0.97512\cdots$. A plot of $E(t)$ for $0 < t < t_0$ is in Figure 3.

To prove that $E(t)$ is positive in $(0, t_0)$, we note that $E(0) = 0$ and look at the derivative, $E'(t)$, which can be represented as a ratio of polynomials with numerator $t^3(1.3832 - 2.3832t + 1.1916t^4)$. The quartic has two real roots 0.7014303105 and 0.8793253979 and two complex roots $-0.7903778542 \pm 1.121296558i$ (to 10 places). At the real roots, $E(t)$ is positive, and thus positive for $0 < t < t_0$. As the figure indicates, E has a relative maximum at the first root and a relative minimum at the second.

4. THE FINAL ALGORITHM

Here is an expanded version of the above outline, with the squeeze-motivated test used to bypass the two logarithms. (For lack of Greek letters in most computer languages, α is represented by a).

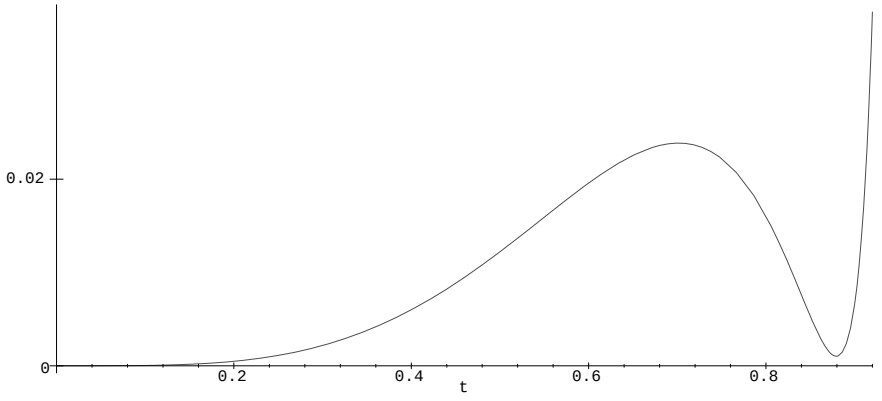

Fig. 3. $E(t)$: half the exponent in $e^{g(x)+x^2/2-\ln(1-0.0331x^4)}$ after setting $x = -t/c$.

(1) Setup: d=a-1/3, c=1/sqrt(9d).

(2) Generate v=(1+cx)^3 with x normal. Repeat if v<=0 [rare; requires x< -sqrt(9a-3)].

(3) Generate uniform U.

(4) If U < 1-0.0331*x^4 return d*v.

(5) If log(U) <0.5*x^2+d*(1-v+log(v)) return d*v.

(6) Go to step 2.

Note that the setup is only required if α is changed from call to call. For repeated calls with α fixed, this algorithm is very fast. Changing α's impose the added cost of a square root and a division.

5. IMPLEMENTATION IN C

The procedure listed below uses RNOR, which provides a standard normal where it occurs in an expression, and UNI, which provides a uniform (0,1) variate. Both are inline functions based on the #define feature of C, and are described in detail below. Assuming the availability of RNOR and UNI, this little procedure will provide gamma variates with parameter a:

```
float rgama(float a) {
float d,c,x,v,u;
d=a-1./3.; c=1./sqrt(9.*d);
for(;;) do {x=RNOR; v=1.+c*x;} while(v<=0.);
     v=v*v*v; u=UNI;
     if( u<1.-.0331*(x*x)*(x*x) ) return (d*v);
     if( log(u)<0.5*x*x+d*(1.-v+log(v)) ) return (d*v); }
          }
```

6. PROVIDING THE NECESSARY UNIFORM AND NORMAL VARIATES

Generation of the necessary uniform and normal variates in the gamma routine itself avoids the necessary overhead costs of seperate routines. The

time needed for a normal variate is the major determinant of speed for this algorithm. The fastest normal generators we know of are based on the ziggurat method of Marsaglia and Tsang [1984; 2000]. Another fast normal generator is based on our Monty Python Method [Marsaglia and Tsang 1998a]. And the Ahrens-Dieter [1989] method is also a fast normal generator that might be used. In terms of both speed and size, we have chosen to use a new implementation of our ziggurat method [Marsaglia and Tsang 2000], providing the required RNOR most of the time in-line, in C. The C programming language allows this to be done via its **define** feature.

For our uniform generator, we use an inline UNI, based on SHR3, [Marsaglia 1996]), a very fast integer generator that adds two successive terms of the 3-shift shift register generator with left shift 13, right shift 17, left shift 5. We have found it to be one of the fastest that seem to pass all the tests in Marsaglia [1996] and not need tables. Complete listings for the inline generators UNI and RNOR follow. The variables are "float," assumed single-precision 32-bit IEEE arithmetic, which seems adequate for most Monte Carlo work. But the ziggurat tables are developed using double precision; for more precision, refer to their development in Marsaglia and Tsang [2000].

```
#define    SHR3    (jz=jsr,    jsr^=(jsr<<13),    jsr^=(jsr>>17),\
jsr^=(jsr<<5), jz+jsr)
#define UNI (SHR3*0.2328306e-9)
#define RNOR (hz=SHR3, iz=hz&127, (abs(hz)[iz])? hz*wn[iz]: nfix())
static unsigned long iz,jz,jsr=349576937,kn[128]; static long hz;
static float wn[128],fn[128];

float nfix(void) /* nfix() provides RNOR if inline cannot */ {
       const float r = 3.713086f;
       static float x,y;
       for(;;) x=hz*wn[iz];
     if(iz==0) { if( UNI* 0.002669629 < exp(-0.5*x*x) ) return x;
     do{x=-log(UNI)*0.2693178; y=-log(UNI); } while(y+y<x*x);
       return ( (hz>0)? r+x: -r-x ); }
       if( fn[iz]+UNI*(fn[iz-1]-fn[iz]) < exp(-0.5*x*x) )
            return x;
         /* start all over */
       hz=SHR3; iz=hz&127; if(abs(hz)<kn[iz]) return (hz*wn[iz]);
         } }

/*--------This procedure sets the seed and creates the tables------*/
void zigset(unsigned long jsrseed)
       const double m1 = 2147483648.0;
     double dn=3.442619855899,tn=dn,vn=9.91256303526217e-3, q;
       int i; jsr=jsrseed;
       q=vn/exp(-0.5*dn*dn);
       kn[0]=(dn/q)*m1; kn[1]=0;
       wn[0]=q/m1; wn[127]=dn/m1;
       fn[0]=1.; fn[127]=exp(-0.5*dn*dn);
       for(i=126;i>=1;i--) {
         dn=sqrt(-2.*log(vn/dn+exp(-0.5*dn*dn)));
         kn[i+1]=(dn/tn)*m1; tn=dn;
         fn[i]=exp(-0.5*dn*dn); wn[i]=dn/m1; } }
```

Table I.

Method	Time per Call, Setup Installed					Time per Call, Setup Each Time				
	α=1	α=2	α=4	α=8	α=16	α=1	α=2	α=4	α=8	α=16
The above rgama	43	42	41	41	41	60	60	59	59	59
Monty Python [Marsaglia et al. 1998b]	52	55	53	51	51	84	86	83	81	81
Ahrens-Dieter [1989]	102	98	67	67	61	134	125	86	80	77
Cheng-Feast [1979]	175	151	187	187	189	253	214	246	246	250
Schmeiser-Lal [1988]	143	130	116	114	114	301	472	464	463	463
Minh [Marsaglia et al. 1999]	176	121	115	111	109	374	460	467	467	467

Note. For the rare cases when a γ_α variate is required for $\alpha < 1$, one can use $\gamma_\alpha = \gamma_{1+\alpha} U^{1/\alpha}$ with U uniform (0,1), and still have a method faster than any we know of. To prove the gamma assertion: the product of the characteristic function of $\ln(\gamma_{\alpha+1})$ and the characteristic function of $\ln(U)/\alpha$ is the characteristic function of $\ln(\gamma\alpha)$.

7. TIMING RESULTS

We conclude with a table showing average times for calls to the above algorithm and to those of Ahrens-Dieter [1982], Cheng-Feast [1979], Schmeiser and Lal [1980], and Minh [1988], as well as for our Monty Python gamma method [Marsaglia and Tsang 1998b].

There are two times for each α of 1, 2, 4, 8, and 16 (actually, $\alpha + 0.0001$ because some methods will not work for $\alpha = 1$ and because some compilers provide $\sqrt{\alpha}$ much faster if α is exactly 1, 4, or 16—perfect squares?). For given α, the faster time assumes the parameters have been assigned before repeated calls.

Times are in nanoseconds, coming from 20 million calls in a 488MHz Pentium III desktop using gcc in MS-DOS. But we also timed the routines on other machines and found the relative speeds much the same. Note that in order that comparisons be fairly made all the methods used the inline uniform generator UNI, based on the fast SHR3 and, when required, the fast inline normal generator RNOR described above.

REFERENCES

AHRENS, J. H. AND DIETER, U. 1982. Generating gamma variates by a modified rejection technique. *Commun. ACM 25*, 1 (Jan.), 47–54.

AHRENS, J. H. AND DIETER, U. 1989. An alias method for sampling from the normal distribution. *Computing 42*, 2-3 (Sep.), 159–170.

CHENG, R. C. H. AND FEAST, G. M. 1979. Some simple gamma variate generators. *Appl. Stat. 28*, 3, 290–295.

DEVROYE, L. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY.

MARSAGLIA, G. 1977. The squeeze method for generating gamma variates. *Comput. Math. Appl. 3*, 321–325.

MARSAGLIA, G. 1984. The exact-approximation method for generating random variables in a computer. *J. Am. Stat. Assoc. 79*, 385 (Mar.), 218–221.

MARSAGLIA, G. 1996. The Marsaglia Random Number CDROM, with the Diehard Battery of Tests of Randomness. Produced under a grant from the National Science at Florida State University. Available at http://stat.fsu.edu/pub/diehard/cdrom.

MARSAGLIA, G. AND TSANG, W. W. 1984. A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions. *SIAM J. Sci. Stat. Comput. 5*, 2 (June), 349–359.

MARSAGLIA, G. AND TSANG, W. W. 1998a. The Monty Python method for generating random variables. *ACM Trans. Math. Softw. 24*, 3 (Sept.), 341–350.

MARSAGLIA, G. AND TSANG, W. W. 1998b. The Monty Python method for generating gamma variables. *J. Stat. Softw. 3*, 1–8.

MARSAGLIA, G. AND TSANG, W. W. 2000. The ziggurat method for generating random variables. *J. Stat. Softw. 5*. Preprint available: geo@stat.fsu.edu or tsang@csis.hku.hk.

MINH, D. L. 1988. Generating gamma variates. *ACM Trans. Math. Softw. 14*, 3 (Sept.), 261–266.

SCHMEISER, B. W. AND LAL, R. 1980. Squeeze methods for generating gamma variates. *J. Am. Stat. Assoc. 75*, 371, 675–682.

Received: March 1999; revised: November 1999 and January 2000; accepted: January 2000