Computing 12, 223—246 (1974)
© by Springer-Verlag 1974

Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions*

By

J. H. Ahrens, Halifax, and **U. Dieter**, Graz

Received September 10, 1973

Abstract — Zusammenfassung

Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions. Accurate computer methods are evaluated which transform uniformly distributed random numbers into quantities that follow gamma, beta, Poisson, binomial and negative-binomial distributions. All algorithms are designed for variable parameters. The known convenient methods are slow when the parameters are large. Therefore new procedures are introduced which can cope efficiently with parameters of all sizes. Some algorithms require sampling from the normal distribution as an intermediate step. In the reported computer experiments the normal deviates were obtained from a recent method which is also described.
Key words and phrases: Random numbers, pseudorandom, normal distribution, gamma distribution, beta distribution, Poisson distribution, binomial distribution, simulation, numerical analysis.

Computermethoden zur Erzeugung Gamma-, Beta-, Poisson- und Binomialverteilter Zufallszahlen. Zur Erzeugung nicht-gleichverteilter Zufallszahlen braucht man Methoden, die gleichverteilte Zufallszahlen in Größen der gegebenen Verteilung transformieren. Es werden Transformationen untersucht, die Gamma-, Beta-, Poisson- oder Binomial-verteilte Zufallszahlen produzieren. Approximative Verfahren werden nicht behandelt. Die bisher bekannten Algorithmen sind langsam, wenn die Parameter der Verteilungen groß sind. Daher werden neue Methoden eingeführt, die diesen Nachteil weitgehend vermeiden. In allen Verfahren dürfen die Parameter beliebig und jedesmal neu gewählt werden. Für manche Transformationen werden normalverteilte Zufallszahlen als Zwischenschritt benötigt; die hierfür verwendete Methode ist ebenfalls angegeben.

1. Introduction

This article presents research on generating random samples from gamma, beta, Poisson and binomial distributions. Sampling methods are not selected for their mathematical appeal but for convenience and efficiency as proved by numerous experiments on a large computer. Other published or unpublished material is suppressed along with approximate methods which are considered dangerous. All listed algorithms are as accurate as can be expected from the floating point arithmetic of the computer in hand.

* This research was supported by the National Research Council of Canada and by Deutsche Forschungsgemeinschaft.

Standardized gamma distributions depend on one parameter a (the mean) whereas beta distributions have two parameters a and b. If these quantities are fixed then fast general methods may be applied. For instance, polynomial sampling as explained in Ahrens-Dieter [1] is suitable, and the decision tree algorithms of Lewis and Payne [21] could also be considered. However, such methods require large tables and fairly involved preparations which can be justified only if samples from a specific gamma (a) or beta (a, b) distribution are needed in very large quantities. The same is true for Marsaglia's table methods (compare Marsaglia [19] or Ahrens-Dieter [1]) which are applicable especially to the discrete Poisson distributions with fixed parameter μ (mean) and binomial distributions with fixed parameters n and p.

All algorithms in this article are constructed for *variable* parameters, and they are specific methods for the given types of distributions. No adjustments are necessary even if the parameters shift continually between samples.

The formal statements of the algorithms are in the style of Knuth [16] with the following conventions:

1. "*Generate* u" (or $\bar{u}$, u_0, u^*, etc.) signifies taking a sample u from the uniform distribution in the open interval between 0 and 1. If several such quantities are generated the samples are assumed to be independent.
2. "*Deliver* x" always refers to the final sample x from the target distribution. An immediate exit from the algorithm is implied after delivery.

All methods were checked out on a CDC-6400 computer. The uniformly distributed variables $u_i \in (0, 1)$ were generated by means of the multiplicative congruential method.

$$y_0 = 1; \; y_{i+1} \equiv 43490275647445\, y_i \pmod{2^{48}}; \; u_i = y_i/2^{48} \tag{1.1}$$

The factor is of the form $8k+5$ and as close to $p(5^{\frac{1}{2}}-1)/2$ where p is the period 2^{46} of the sequence (1.1). For a motivation of this choice the reader is referred to Ahrens, Dieter and Grube [3] and Dieter [7]. An assembler routine expressing (1.1) is 4 words long and requires 23 μsec per sample on the CDC-6400 computer.

Some methods in this paper require efficient sampling from the standard normal distribution. For this a new algorithm by Forsythe, Ahrens and Dieter was preferred to all procedures in Ahrens-Dieter [1]. The statement of this new method for the normal distribution in Section 2 may be considered as a necessary supplement to the survey paper Ahrens-Dieter [1].

Section 3 starts with the simple "multiplication" method for gamma distributions of integer parameters. This is then extended to non-integer values of a but remains unsatisfactory for large a. Therefore three new efficient methods are added which are based on comparisons with Cauchy and normal distributions.

Every method for gamma distributions can also be applied to beta distributions. However, in Section 4 direct comparisons with the normal distribution yield alternative specific algorithms for beta distributions.

Section 5 starts with the well-known multiplication method for the Poisson distribution which is again well behaved only for small means μ. Of the two new algorithms the "center triangle" method is elementary and very efficient but somewhat long. The second one is an elegant "compound" method which is however dependent on the existence of a good routine for sampling from gamma distributions.

For the binomial distributions in Section 6 and the negative-binomial distributions in Section 7 the pattern repeats itself: the "obvious" methods are convenient but slow for large main parameters. But as in all other cases, this paper provides efficient alternatives whose computation times remain reasonable.

2. The Normal Distribution

The new algorithm for sampling from the standard normal distribution is related to J. von Neumann's classical method [23] for the exponential distribution (Algorithm NE in Ahrens-Dieter [1]). Its generalisation to arbitrary probability density functions of the form $\exp(-G(x))$ is due to G. E. Forsythe [13] and the authors [10]. The special algorithm for the normal distribution was extended in Ahrens-Dieter [2] to a sequence of methods which range from short medium-speed algorithms to longer and very fast procedures. Some of the methods in Sections 3—7 require normal deviates, and in these cases the new Algorithm FL below was always used. FL is the special case FL_5 in Ahrens-Dieter [2] and depends on the split of the positive part of the standard normal distribution into 31 "center" intervals of equal area 1/32 and into "tail" intervals of areas 1/64, 1/128, 1/256, ... More precisely, let

$$\Phi(x) = \int_x^\infty (2/\pi)^{1/2} e^{-z^2/2}\, dz \tag{2.1}$$

Then the following quantities have to be tabulated:

Center: Table C

(1) *The boundaries* $a_i = \Phi^{-1}((33-i)/32)$ *of the* 31 *center intervals for* $i = 1, 2, \ldots, 32$.

(2) *The quantities* $t_i = ((a_{i+1} - a_i)/2 + a_i)(a_{i+1} - a_i)$ *for* $i = 1, 2, \ldots, 31$.

(3) *The quantities* $h_i = (a_{i+1} - a_i)/(1 - t_i)$ *for* $i = 1, 2, \ldots, 31$.

Tail: Table T

(1) *The lengths of the tail intervals* $d_i = \Phi^{-1}(2^{-i}) - \Phi^{-1}(2^{-i+1})$ *for* $i = 6, 7, \ldots, B-1$ *where B is the number of bits in the mantissa of the floating-point numbers for the computer in hand.* ($B-1=47$ *in the case of the very accurate CDC* 6400 *machine.*) (*The first* 5 d_i *in the Table T below are not needed.*)

With the quantities in Tables C and T the complete Algorithm FL can now be spelt out. For the derivation of all the technical details the reader is referred to Ahrens-Dieter [2].

Algorithm FL *(Forsythe, Ahrens-Dieter)*

```
1. Generate $u$. Store the first bit of $u$ as a sign $s$. Left-shift $u$ by 1 bit.
2. Shift $u$ to the left by 5 bits ($u \leftarrow 32u$) and equate $i$ to the integer part of
   $u$ ($i \leftarrow [u]$). If $i=0$ go to 9.
3. (Center). Set $u^* \leftarrow u-i$ and $a \leftarrow a_i$.
4. If $u^* > t_i$ set $w \leftarrow (u^*-t_i)\,h_i$ and go to 17.
5. Generate $u$. Set $w \leftarrow u\,(a_{i+1}-a)$ and calculate $t \leftarrow (w/2+a)\,w$.
6. If $u^* > t$ go to 17.
7. Generate $u$. If $u^* < u$ generate $u^*$ and go to 4.
8. Set $t \leftarrow u$, generate $u^*$ and go to 6.
9. (Tail). Set $i \leftarrow 6$ and $a \leftarrow a_{32}$.
10. Left-shift $u$ by 1 bit ($u \leftarrow 2u$). If $u \geq 1$ go to 12.
11. Set $a \leftarrow a+d_i$, $i \leftarrow i+1$ and go to 10.
12. Set $u \leftarrow u-1$.
13. Set $w \leftarrow u\,d_i$ and $t \leftarrow (w/2+a)\,w$.
14. Generate $u^*$. If $u^* > t$ go to 17.
15. Generate $u$. If $u^* < u$ generate $u$ and go to 13.
16. Set $t \leftarrow u$ and go to 14.
17. Set $y \leftarrow a+w$. If $s=0$ deliver $x \leftarrow y$, if $s=1$ deliver $x \leftarrow -y$.
```

The CDC-6400 assembler code program for FL was 28 words (at 60 bits) long plus 136 words for the constants in Tables C and T. It took an average of 47 μsec to produce one standard-normally distributed sample which is only twice the time required for sampling from the (0, 1)-uniform distribution. The expected number N of uniformly distributed quantities needed (in Steps 1, 5, 7, 8, 14 and 15) is 1.23156 which means that in the majority of cases the only generation (of u) is in Step 1.

In Ahrens-Dieter [2] FL is compared with the best algorithms in the survey paper Ahrens-Dieter [1]. It appears that the FL is faster than all other methods except Marsaglia's [18] Rectangle-Wedge-Tail algorithm RT. Since FL is more elegant, easier to program and (a little) more accurate than RT, it is preferred by the authors as the most reasonable compromise between speed, accuracy and convenience.

Table C. *32 Center Intervals*

i	a_i	t_i	h_i
1	0.00000000000000	0.00076738283767	0.03920617164634
2	0.03917608550309	0.00230687039764	0.03932704963665
3	0.07841241273311	0.00386061844387	0.03950999486086
4	0.11776987457909	0.00543845406707	0.03975702679515
5	0.15731068461017	0.00705069876857	0.04007092772490
6	0.19709908429430	0.00870839582019	0.04045532602655

i	a_i	t_i	h_i
7	0.23720210932878	0.01042356984914	0.04091480886081
8	0.27769043982157	0.01220953194966	0.04145507115859
9	0.31863936396437	0.01408124734637	0.04208311051344
10	0.36012989178957	0.01605578804548	0.04280748137995
11	0.40225006532172	0.01815290075142	0.04363862733472
12	0.44509652498551	0.02039573175398	0.04458931789605
13	0.48877641111466	0.02281176732513	0.04567522779560
14	0.53340970624127	0.02543407332319	0.04691571371696
15	0.57913216225555	0.02830295595118	0.04833486978119
16	0.62609901234641	0.03146822492920	0.04996298427702
17	0.67448975019607	0.03499233438388	0.05183858644724
18	0.72451438349236	0.03895482964836	0.05401138183398
19	0.77642176114792	0.04345878381672	0.05654656186515
20	0.83051087820539	0.04864034918076	0.05953130423884
21	0.88714655901887	0.05468333844273	0.06308488965373
22	0.94678175630104	0.06184222395816	0.06737503494905
23	1.00999016924958	0.07047982761667	0.07264543556657
24	1.07751556704027	0.08113194985866	0.07926471414968
25	1.15034938037600	0.09462443534514	0.08781922325338
26	1.22985875921658	0.11230007889456	0.09930398323927
27	1.31801089730353	0.13649799954975	0.11555994154118
28	1.41779713799625	0.17168856004707	0.14043438342816
29	1.53412054435253	0.22762405488269	0.18361418337460
30	1.67593972277344	0.33049802776911	0.27900163464163
31	1.86273186742164	0.58470309390507	0.70104742502766
32	2.15387469406144	—	—

Table T. *Lengths of Tail Intervals*

i	d_i	i	d_i	i	d_i
1	0.67448975019607	17	0.15040938382813	33	0.10597677198479
2	0.47585963017993	18	0.14590257684509	34	0.10433484129317
3	0.38377116397654	19	0.14177003276856	35	0.10276601206127
4	0.32861132306910	20	0.13796317369537	36	0.10126505151402
5	0.29114282663980	21	0.13444176150074	37	0.09982723448906
6	0.26368432217502	22	0.13117215026483	38	0.09844828202068
7	0.24250845238097	23	0.12812596512583	39	0.09712430874765
8	0.22556744380930	24	0.12527909006226	40	0.09585177768776
9	0.21163416577204	25	0.12261088288608	41	0.09462746119186
10	0.19992426749317	26	0.12010355965651	42	0.09344840710526
11	0.18991075842246	27	0.11774170701949	43	0.09231190933664
12	0.18122518100691	28	0.11551189226063	44	0.09121548217294
13	0.17360140038056	29	0.11340234879117	45	0.09015683778986
14	0.16684190866667	30	0.11140272044119	46	0.08913386650005
15	0.16079672918053	31	0.10950385201710	47	0.08814461935364
16	0.15534971747692	32	0.10769761656476		

3. Gamma Distributions

The standard gamma (a) distributions are given by their probability densities

$$f(x)=e^{-x}x^{a-1}/\Gamma(a),\ x\geq 0;\ a>0. \tag{3.1}$$

Non-standard gamma distributions are obtained by a linear transformation of x. The following well-known fact will be used extensively.

Lemma 3.1: *If x and y are independent gamma-distributed random variables with parameters a and b respectively then $x+y$ is gamma-distributed with parameter $a+b$.*

In the case $a=1$ (3.1) becomes $f(x)=e^{-x}$: standard gamma (1) variables are exponential and may be generated as negative logarithms $-\ln u$ of uniform variables (method LG in Ahrens-Dieter [1]). If a is a positive integer n, Lemma 3.1 can be applied ($n-1$ times) giving $x\ \ -\ln u_1-\ln u_2-\ldots-\ln u_n$ as a valid transformation. In other words, one has

Algorithm GM (*Multiplication Method, $a=n$ integer*)

```
1. Initialize $i \leftarrow p \leftarrow 1$.
2. Generate $u$ and form $p \leftarrow pu$.
3. If $i=n$ deliver $x \leftarrow -\ln p$, otherwise set $i \leftarrow i+1$ and go to 2.
```

For *fractional* parameters a rejection technique (compare General Method AR in Ahrens-Dieter [1]) is now presented based on the majorizing function

$$g(x)=x^{a-1}/\Gamma(a)\ \text{if}\ 0<x\leq 1;\ g(x)=e^{-x}/\Gamma(a)\ \text{if}\ 1\leq x. \tag{3.2}$$

Since $e^{-x}\leq 1$ if $0<x$ and $x^{a-1}\leq 1$ if $a\leq 1$ and $1\leq x$ the inequality $e^{-x}x^{a-1}/\Gamma(a)=$ $=f(x)\leq g(x)$ is valid for all $x>0$. The function

$$h(x)=x^{a-1}\,ea/(e+a)\ \text{if}\ 0<x\leq 1;\ h(x)=e^{-x}\,ea/(e+a)\ \text{if}\ 1\leq x$$

is a probability density that is proportional to $g(x)$. Sampling from $h(x)$ is no problem since both parts have easily invertible integrals: with a probability of $e/(e+a)$ an x below 1 (first part of $h(x)$) is sampled, otherwise the second part of $h(x)$ is used. The rejection test is based on $f(x)/g(x)$ which is e^{-x} or x^{a-1}. After some streamlining one obtains the

Algorithm GS (*Ahrens*, $0<a\leq 1$)

```
1. Generate $u$. Set $b \leftarrow (e+a)/e$ and $p \leftarrow bu$. If $p>1$ go to 3.
2. (Case $x\leq 1$). Set $x \leftarrow p^{1/a}$. Generate $u^*$. If $u^*>e^{-x}$ go to 1 (rejection), otherwise
   deliver $x$.
3. (Case $x>1$). Set $x \leftarrow -\ln((b-p)/a)$. Generate $u^*$. If $u^*>x^{a-1}$ go to 1 (rejection),
   otherwise deliver $x$.
```

The expected number of trials per sample is $\alpha=(e+a)/(ea\,\Gamma(a))$, a quantity which is maximally 1.39 (at $a\ \ 0.8$) and on average 1.27 (if all a between zero and one are considered equally likely). Therefore GS is a reasonably efficient supplement for GM, and the next algorithm uses Lemma 3.1 to combine GM and GS into a method for *all* parameters $a>0$.

Algorithm GT (*Two-Part-Method*)

```
1. Split a into the integer part $m \leftarrow [a]$ and the fractional part $f \leftarrow a-m$.
2. If $m=0$ set $y \leftarrow 0$ and go to 3. Otherwise take a sample $y$ from the gamma $(m)$
      distribution using GM.
3. If $f=0$ set $z \leftarrow 0$ and go to 4. Otherwise take a sample $z$ from the gamma $(f)$
   distribution using GS.
4. Deliver $x \leftarrow y+z$.
```

It is clear — and confirmed in the performance Table G — that the computation times of GM and GT must grow linearly with the size of a. The resulting unsatisfactory behaviour in the case of large a is avoided in the next method which is based on the inequality

$$g(x)=\frac{1}{\Gamma(a)}\,\frac{e^{-(a-1)}(a-1)^{a-1}}{1+(x-(a-1))^2/(2a-1)}\geq\frac{e^{-x}x^{a-1}}{\Gamma(a)}=f(x)\text{ if }a>1. \tag{3.3}$$

Proof: In order to show that the constant $2a-1$ is optimal one may even prove that

$$\varphi(x)=e^{-x}x^{a-1}\left(1+c^{-1}(x-(a-1))^2\right)\leq e^{-(a-1)}(a-1)^{a-1}$$

is true for all $c\geq 2a-1$ and all $x\geq 0$. Obviously for $x=a-1$ one has equality. The first derivative of $\varphi(x)$ works out to

$$\varphi'(x)=-c^{-1}e^{-x}x^{a-2}(x-(a-1))\left((x-a)^2+c-(2a-1)\right).$$

Since $c\geq 2a-1>0$, $\varphi'(x)$ changes sign only at $x=a-1$. Therefore $\varphi(x)$ has its only maximum at $x=a-1$ and cannot exceed $e^{-(a-1)}(a-1)^{a-1}$ anywhere.

The function $g(x)$ is proportional to a Cauchy probability density whose integral is an arctan. Inverting this distribution function yields the sampling method

$$x\leftarrow a-1+(2a-1)^{\frac{1}{2}}\tan(\pi(u-0.5))$$

which is embodied in Step 2 of the "*Cauchy-Method*" below.

Algorithm GC (*Ahrens*, $a>1$)

```
1. Set $b \leftarrow a-1$, $A \leftarrow a+b$ and $s \leftarrow A^{\frac{1}{2}}$.
2. Generate $u$. Set $t \leftarrow s \tan(\pi(u-0.5))$ and $x \leftarrow b+t$.
3. If $x<0$ go to 2.
4. Generate $u^*$. If $u^*>\exp(b\,ln(x/b)-t+ln(1+t^2/A))$ go to 2 (rejection). Other-
   wise deliver $x$.
```

It can be shown that the expected number α of trials per complete sample is $\alpha=\pi(2a-1)^{\frac{1}{2}}e^{-(a-1)}(a-1)^{a-1}/\Gamma(a)$. This quantity decreases from $\alpha=\pi=3.1416$

for $a=1+\varepsilon$ to $\alpha=2.0018$ for $a=2$ to $\alpha=1.9014$ for $a=3$, and approaches $\alpha=\pi^{\frac{1}{2}}=1.7725$ as $a\to\infty$.

Therefore GC is an easily programmable method whose computation time quickly becomes practically constant as a is increased (compare Table G). The fact that α is eventually $\pi^{\frac{1}{2}}$ and not 1 is the price paid for convenience: it is easy to tuck the entire gamma density under the very wide tails of the Cauchy distribution.

In order to decrease the number of rejections one should really compare the given gamma (a) distribution with a suitable normal distribution, since this is after all the limiting case for $a\to\infty$. Unfortunately, it is *impossible* to encase any gamma distribution completely under a Gaussian curve; no matter how the parameters are chosen some part of the tail will stick out. Yet a highly efficient algorithm is now devised by combining a normal envelope $g(x)$ around the bulk of the gamma distribution with an exponential cover $h(x)$ above the tail. The method is prepared in the next lemma in which $\varphi(x)$ is proportional to the gamma density $f(x)$ in (3.1), and $g(x)$ and $h(x)$ are proportional to normal and exponential probability densities.

Lemma 3.2: *(Dieter) Let*
$\varphi(x)=(x/(a-1))^{a-1}\,e^{-(x-(a-1))},\quad x\in[0,\infty);$
$g(x)=\exp(-(x-(a-1))^2/(2(a+ca^{\frac{1}{2}}))),\quad x\in[0,b];$
$h(x)=(b/(a-1))^{a-1}\exp(-(1-(a-1)/b)\,x),\quad x\in(b,\infty);$
where $c=(8/3)^{\frac{1}{2}}$ *and* $b=a-1+(3/2)\,c\,(a+ca^{\frac{1}{2}})^{\frac{1}{2}}$.
Then $\varphi(x)\le g(x)$ *for* $0\le x\le b$ *and* $\varphi(x)\le h(x)$ *for* $b\le x<\infty$.

The reader is referred to Dieter-Ahrens [11] for the lengthy proof of $\varphi(x)\le g(x)$ (the other inequality is trivial). It is not difficult to amass sufficient numerical evidence, but only the exact proof shows that the above special values of b and c are really well motivated.

The areas under $g(x)$ and $h(x)$ are

$$G=\int_{-\infty}^{+\infty} g(x)\,dx=(2\pi(a+ca^{\frac{1}{2}}))^{\frac{1}{2}};$$
$$H=\int_{b}^{\infty} h(x)\,dx=(b/(a-1))^{a-1}\,(b/(b-(a-1)))\,e^{-(b-(a-1))} \qquad (3.4)$$

In the following preliminary statement of the "normal-exponential" method the majorizing function is $g(x)$ in $(-\infty,b]$ and $h(x)+g(x)$ in $[b,\infty)$. However, all x from $g(x)$ are discarded if they lie outside the interval $[0,b]$.

Preliminary Normal-Exponential Method

1. Generate u. If $u\le H/(G+H)$ go to 3.
2. Take a sample s from the standard normal distribution and set x $a-1+s\sigma$ where $\sigma=(a+ca^{\frac{1}{2}})^{\frac{1}{2}}$. If $x<0$ or $x>b$ go to 1. Otherwise carry out a rejection test: generate u and deliver x if $u\le\varphi(x)/g(x)$, otherwise go to 1.

3. Take a sample s from the standard exponential distribution and set $x \leftarrow b + bs/(b-(a-1))$. Generate u and deliver x if $u \leq \varphi(x)/h(x)$, otherwise go to 1.

The trouble with this preliminary method is the lengthy calculation of $H/(G+H)$ in Step 1. But this can be avoided if the largest possible value of $H(G+H)$ is taken in all cases. It turns out that the maximum

$$\beta = \max\,(H/(G+H)) = 0.009572265238289 \tag{3.5}$$

occurs at $a = 5.949512$. Now let

$$\chi(x) = h(x)\,(\beta/(1-\beta))\,(2\pi(a+ca^{\frac{1}{2}}))^{\frac{1}{2}}\,((a-1)/b)^{a-1}\,((b-(a-1))/b)\,e^{b-(a-1)} \tag{3.6}$$

It follows from (3.4) and (3.5) that $\chi(x) \geq h(x)$ for all $a > 0$ and that

$$\int_b^\infty \chi(x)\,dx/(G + \int_b^\infty \chi(x)\,dx) = \beta \tag{3.7}$$

irrespective of a. Hence if the slightly wider tail envelope $\chi(x)$ is taken instead of $h(x)$ the following version of the normal-exponential method emerges.

Algorithm GN (*Dieter-Ahrens*, $a > 1$)

1. *Set* $\mu \leftarrow a-1$, $\sigma \leftarrow (a + 1.63299316185545\, a^{\frac{1}{2}})^{\frac{1}{2}}$, $d \leftarrow 2.44948974278318\,\sigma$ and $b \leftarrow \mu + d$. (*The constants are* $(8/3)^{\frac{1}{2}}$ *and* $6^{\frac{1}{2}}$.)
2. *Generate u. If* $u \leq 0.009572265238289$ *go to* 5. (*The constant is* β.)
3. *Take a sample s from the standard normal distribution and set* $x \leftarrow \mu + \sigma s$. *If* $x < 0$ *or* $x > b$ *go to* 2.
4. *Generate u. If* $\ln u > \mu(1 + \ln(x/\mu)) - x + s^2/2$ *go to* 2, *otherwise deliver x.*
5. *Take a sample s from the standard exponential distribution and set* $x \leftarrow b(1 + s/d)$.
6. *Generate u. If* $\ln u > \mu(2 + \ln(x/\mu) - x/b) + 3.7203284924588 - b - \ln(\sigma d/b)$ *go to* 2, *otherwise deliver x.* (*The constant is* $-\ln((2\pi)^{\frac{1}{2}}\beta/(1-\beta))$).

In the preliminary method the most difficult calculation occurred in Step 1. Using the constant β the present algorithm transfers the tedious job to Step 6 which is carried out in less than 1% of all cases.

The performance of GN was tested by a Fortran function which used FL (in assembler code) for the standard normal distribution (47 μ sec per sample) and the efficient (assembler) algorithm SA in Ahrens-Dieter [1] (43 μ sec per sample) for the exponential distribution. However, if simple inversion $s \leftarrow -\ln u$ were used for the first part of Step 5 the speed would not suffer appreciably, since this concerns only the rare tail case. On the other hand, only a good routine for the normal distribution (such as FL) will improve the performance of GN over GC by about a factor two (Table G).

The expected number α of trials per sample can be worked out numerically: it is $\alpha = 4.1067$ for $a = 1 + \varepsilon$; $\alpha = 1.9328$ for $a = 2$; $\alpha = 1.2985$ for $a = 10$; $\alpha = 1.0936$ for $a = 100$; and $\alpha = 1.0358$ for $a = 1000$. So should one not simply substitute normal samples for gamma (a) samples as soon as a is „large enough"? The authors have enough numerical evidence to believe that this *cannot* be recommended if the occasional samples from the tail (the "accidents") are of special significance as they often are: for large x the two probability densities show characteristic differences even in the case of sizeable parameters a.

The algorithm GN has one major shortcoming: the calculation of the two logarithms in Step 4 which occurs in over 99% of all cases. It will now be shown how this can be avoided most of the time. The right hand side of the test in Step 4 may be written as $\ln Q(x)$ where

$$Q(x)=e^{\mu}(x/\mu)^{\mu}e^{-x}e^{s^2/2}=(1+s\sigma/\mu)^{\mu}e^{-s\sigma}e^{s^2/2}=q(s). \tag{3.8}$$

The following inequalities hold for this function.

Lemma 3.3:
$1-(s^2/2)(\sigma^2/\mu-1)+s^3\sigma^2/(\mu a^{\frac{1}{2}})\le q(s)$ *if* $s\le 0$ *and* $a>2.5327805161251$
$1-(s^2/2)(\sigma^2/\mu-1)\le q(s)$ *if* $s\ge 0$ *and* $a>1$ *(Dieter-Ahrens).*

Again the reader is referred to Dieter-Ahrens [11] for the somewhat tedious proof of this lemma.

The improvement on Algorithm GN is now achieved by replacing Step 4 with tests based on Lemma 3.3. In the optimized algorithm GO below the old comparison (now Step 7) occurs only if the simpler tests in Steps 5 and 6 fail.

Algorithm GO (*Dieter-Ahrens*, $a>2.5327805161251$)

```
1. Set $\mu\leftarrow a-1$, $V\leftarrow a^{\frac{1}{2}}$, $\sigma^2\leftarrow a+1.632993161855\,V$, $\sigma\leftarrow(\sigma^2)^{\frac{1}{2}}$, $W\leftarrow\sigma^2/\mu$,
   $d\leftarrow 2.44948974278318\,\sigma$ and $b\leftarrow\mu+d$. (The constants are $(8/3)^{\frac{1}{2}}$ and $6^{\frac{1}{2}}$).
2. Generate u. If $u\le 0.009572265238289$ go to 8. (The constant is $\beta$.)
3. Take a sample s from the standard normal distribution and set $x\leftarrow\mu+\sigma s$. If
   $x<0$ or $x>b$ go to 2.
4. Generate u and set $S\leftarrow s^2/2$. If $s\ge 0$ go to 6.
5. If $u\le 1-S\left((1-2s/V)\,W-1\right)$ deliver x, otherwise go to 7.
6. If $u\le 1-S\,(W-1)$ deliver x.
7. If $\ln u>\mu\left(1+\ln(x/\mu)\right)-x+S$ go to 2, otherwise deliver x.
8. Take a sample s from the standard exponential distribution and set $x\leftarrow b\,(1+s/d)$.
9. Generate u. If $\ln u>\mu\left(2+\ln(x/\mu)-x/b\right)+3.7203284924588-b-\ln(\sigma d/b)$
   go to 2, otherwise deliver x. (The constant is $-\ln\left((2\pi)^{\frac{1}{2}}\beta/(1-\beta)\right)$.)
```

GO was again tested by a Fortran function which used assembler programs of FL and SA for sampling from the standard normal and exponential distributions. The improvement over GN was marked and was further enhanced by re-writing the entire method in assembler code. The core load of this final program had 94 words for GO proper (including in-line square root), 37 words for a fast logarithm, 164 words for FL and 31 words for SA — a total of 326 words (at 60 bits).

In its final form GO the normal-exponential method is a superior algorithm for gamma distributions, but a complete application program will still need GT to cover small parameters (e.g. GT for $a \leq 3$ and GO for $a > 3$).

The computation times in Table G (and subsequent performance statistics) are based on experiments of 1000 samples each.

Table G. *Gamma Distributions — Computation Times* [μsec]

Parameter $a=$	0.5	1	2	5	10	20	50	100	1000	10000
GM Fortran	—	175	213	315	479	815	1799	3460	—	—
GT Fortran[1]	511	636	686	766	961	1281	2268	3893	—	—
GC Fortran	—	1167[2]	896	919	938	938	962	957	—	—
GN Fortran[4]	—	997[2]	629	584	553	531	507	496	480	474
GO Fortran[4]	—	—	520[3]	498	451	420	390	372	344	333
GO Assembler	—	—	416[3]	400	361	331	302	289	261	253

[1] The parameters are 0.5, 1.5, 2.5, ..., 100.5 instead of 0.5, 1, 2, ..., 100.
[2] Instead of $a=1$ read $a=1+\varepsilon$ (ε very small).
[3] Read $a=3$ instead of $a=2$. (Algorithm invalid below $a=2.53$.)
[4] With Assembler subprograms for sampling from the normal and exponential distributions.

4. Beta Distributions

Beta (a, b) distributions are defined by their probability density functions

$$f(x)=x^{a-1}(1-x)^{b-1}/B(a,b);\ 0 \leq x \leq 1;\ a,b>0; \qquad B(a,b)=\Gamma(a)\,\Gamma(b)/\Gamma(a+b). \tag{4.1}$$

Sampling from any beta distribution may always be reduced to sampling from two gamma distributions according to the following well-known fact.

Lemma 4.1: *If x and y are two independent deviates which are (standard) gamma-distributed with parameters a and b respectively, then $x/(x+y)$ is beta (a,b) distributed.*

Therefore all gamma methods in Section 3 reappear in Table B which lists sampling speeds for beta distributions.

There are, however, specific methods for beta distributions. First of all, if a and b are positive integers a well-known statistical theorem allows the following

Algorithm BR (*Jöhnk, a, b integers*)

1. *Generate $u_1, u_2, \ldots, u_n$ where $n=a+b-1$.*
2. *Deliver the a-th smallest of the u_i as the sample x from the beta (a,b) distribution.*

In order to find the one u_i that has rank a one could first order the set $\{u_i\}$. But even efficient sorting routines will lead to computation times which grow proportionally to $n \ln n$. However, there are other algorithms which do not sort the entire set and produce the a-th smallest u_i in kn steps. One such method (based on "splitting") was programmed efficiently in assembler code, but the performance proved disappointing: it is shown in Table B that BR is not even better than the simpler multiplication method GM. Since there is no way of making the computation times grow less than linearly with $n=a+b-1$ the Algorithm BR seems to be useless in spite of its apparent simplicity and appeal.

It will be more useful to reconsider enveloping normal distributions. Since the beta distributions are confined to the interval $[0, 1]$ there is no trouble with tails, and total encasing becomes possible. An optimal way for arbitrary parameters a, b is stated in the following inequality

$$(x/A)^A\,((1-x)/B)^B\,C^C \le \exp\left(-2\,C\,(x-A/C)^2\right) \tag{4.2}$$

where $A=a-1$, $B=b-1$, $C=A+B=a+b-2$, $a>1$, $b>1$ and $x\in[0, 1]$.

The left hand side of (4.2) is proportional to the beta (a, b) density. The right hand side is proportional to the density of a normal distribution with mean $\mu=A/C$ and standard deviation $\sigma=1/(2\,C^{\frac{1}{2}})$.

For the proof of (4.2) consider the quotient function

$$f_\omega(x)=(x/A)^A\,((1-x)/B)^B\,C^C\,\exp\left((\omega/2)\,C\,(x-A/C)^2\right).$$

It is easily seen that $f_\omega(A/C)=1$ for all ω. Proving (4.2) means showing that $f_4(x)\le 1$ for all x. The derivative of $f_\omega(x)$ works out to

$$f'_\omega(x)=f_\omega(x)\,C\,(x-A/C)\,\left(\omega-1/(x\,(1-x))\right).$$

One has $1/(x\,(1-x))\ge 4$ for $x\in[0, 1]$. Consequently if $\omega\le 4$ then $f'_\omega(x)$ is positive if $x<A/C$ and negative if $x>A/C$. This proves (4.2) and shows that the choice $\omega=4$ yields the tightest envelopes.

The resulting rejection algorithm reads as follows:

Algorithm BN (*Ahrens-Dieter*, a, $b>1$)

1. *Set* $A\leftarrow a-1$, $B\leftarrow b-1$, $C\leftarrow A+B$, $L\leftarrow C\ln C$, $\mu\leftarrow A/C$ *and* $\sigma\leftarrow 0.5/C^{\frac{1}{2}}$.
2. *Take a sample s from the standard normal distribution and form* $x\leftarrow s\,\sigma+\mu$.
3. *If* $x<0$ *or* $x>1$ *go to* 2 (*rejection*).
4. *Generate u. If* $\ln u>A\ln(x/A)+B\ln((1-x)/B)+L+0.5\,s^2$ *go to* 2 (*rejection*). *Otherwise deliver x.*

The expected number α of trials (Steps 2) per completed sample works out to

$$\alpha=(\tfrac{1}{2}\pi/C)^{\frac{1}{2}}\,A^A\,B^B\,\Gamma(a+b)/(C^C\,\Gamma(a)\,\Gamma(b))\approx\tfrac{1}{2}(A+B)/(AB)^{\frac{1}{2}} \tag{4.3}$$

The approximation is based on Stirling's formula and is good except for small a and b. It follows that the method quickly attains maximum efficiency in the symmetrical case $a=b$. The computation times in Table B indicate that BN is better than the previous gamma methods except in very skew cases.

For symmetrical beta distributions an improvement of BN was explored which is similar in its approach to the gamma routine GO. If $a=b$ then $A=B$, $C=2A$, and Step 4 tests whether $u>(4x(1-x))^A e^{s^2/2}$ is true. From $x=s\sigma+\mu$ it follows that the condition may be written as $u>q(s)$ where

$$q(s)=(1-\tfrac{1}{2}s^2/A)^A e^{s^2/2} \tag{4.4}$$

The following inequalities hold for this function.

Lemma 4.2:
$1-s^4/(8a-12)\leq q(s)\leq 1-s^4/(8a-8)+\frac{1}{2}(s^4/(8a-8))^2$ *for all* $a>3/2$ *(Dieter-Ahrens).*

The proof is again found in Dieter-Ahrens [11], and the improvement on Algorithm BN reads:

Algorithm BS (*Ahrens-Dieter*, $a=b>1.5$)

```
1. Set $A \leftarrow a-1$ and $t \leftarrow (A+A)^{\frac{1}{2}}$.
2. Take a sample $s$ from the standard normal distribution and form $x \leftarrow \frac{1}{2}(1+s/t)$.
3. If $x<0$ or $x>1$ go to 2.
4. Generate $u$. If $u \leq 1-s^4/(8a-12)$ deliver $x$.
5. If $u \geq 1-s^4/(8a-8)+\frac{1}{2}(s^4/(8a-8))^2$ go to 2.
6. If $\ln u > A \ln(4x(1-x))+s^2/2$ go to 2.
7. Deliver $x$.
```

Table B. *Beta Distributions — Computation Times* [μsec]

First Parameter	$a=$	1	2	5	10	20	50	100
GM Fortran	$b=a$	383	452	650	981	1644	3613	6921
GT Fortran[1]	$b=a$	1314	1383	1567	1915	2560	4530	7799
GC Fortran	$b=a$	2389[2]	1814	1837	1879	1919	1942	1939
GN Fortran[4]	$b=a$	2030[2]	1280	1180	1120	1080	1030	1010
GO Fortran[4]	$b=a$	—	1060[3]	1010	920	860	800	760
GO Assembler	$b=a$	—	850[3]	810	740	680	620	590
BR Fortran[5]	$b=a$	136	212	465	911	1805	4547	9063
BN Fortran[4]	$b=1$	—	752	939	1197	1581	2319	3180
,,	$b=2$	757	617	633	693	809	1108	1391
,,	$b=5$	929	619	597	612	677	860	1024
,,	$b=10$	1182	695	610	576	600	710	859
,,	$b=20$	1549	822	667	600	568	612	739
,,	$b=50$	2321	1083	866	712	612	596	614
,,	$b=100$	3165	1417	1049	869	732	618	593
BS Assembler	$b=a$	—	189	165	157	153	151	151

[1] The parameters are 1.5, 2.5, ..., 100.5 instead of 1, 2, ..., 100.
[2] Instead of $a=1$ read $a=1+\varepsilon$ (ε very small).
[3] Read $a=3$ instead of $a=2$ (Algorithm invalid below $a=2.53$).
[4] With assembler subprogram for sampling from the normal distribution.
[5] With assembler subprogram for finding an a-th smallest number.

The algorithm was programmed in Assembler Code using method FL for sampling from the standard normal distribution in Step 2. As shown in Table B the computation times stabilized quickly at 150 μ sec. The additional effort for the special case $a=b$ was considered worthwhile, because of the important sampling method BB for the binomial distribution in Section 6 which requires samples from symmetrical beta distributions of medium and large parameters.

The computation times for the methods GM, GT, GC, GN and GO are essentially composed of the times for sampling from gamma (a) and gamma (b) distributions. In Table B it is therefore sufficient to list the symmetrical case $a=b$. Similarly the speed of BR obviously depends only on $a+b-1$. Since BS is restricted to $a=b$ the algorithm BN is the only method that required experiments with skew ($a \neq b$) distributions. Here random fluctuations in the 1000 samples caused only slight deviations from an otherwise expected symmetrical performance table.

5. Poisson Distributions

The Poisson distribution of mean μ is defined by its probability function

$$p_i = e^{-\mu} \mu^i / i! \quad (i=0, 1, 2, \ldots). \tag{5.1}$$

A simple sampling method is described in Knuth [16] (Algorithm Q on page 117):

Algorithm PM (*Multiplication Method*)

```
1. Set $L \leftarrow e^{-\mu}$ and initialize $k \leftarrow 0$ and $p \leftarrow 1$.
2. Generate $u$ and replace $p \leftarrow pu$.
3. If $p \geq L$ increase $k \leftarrow k+1$ and go to 2.
4. Deliver $k$ as the required Poisson ($\mu$) sample.
```

This algorithm leads to very short computer programs (8 words in CDC 6400 assembler) and performs reasonably well as long as the mean μ is small. When μ is increased the computation times grow linearly as shown in Table P. There are a number of alternative algorithms which have the same unsatisfactory behaviour for large means, one is the classical method of searching through a table of cumulative probabilities which have to be calculated at the same time.

Efficient sampling in the case of large means remains the real problem. Again, simply replacing the Poisson distribution with a normal approximation is a poor idea whenever samples from the tail are important. A normal envelope (in the style of GN in Section 3) is a possibility which the authors discarded after some experiments. The fact that the Poisson distribution is discrete makes accurate methods based on continuous approximations rather messy.

The first successful and accurate algorithm works with an isosceles triangle which is inscribed under the histogram of the given Poisson (μ) distribution. Let m be the integer part $[\mu]$ of the mean and let

$$g(x)=b/s-(b/s^2)\,|x-m| \text{ if } x\in[m-s,m+s] \;(g(x)=0 \text{ elsewhere}),\quad s=cm^{\frac{1}{2}}-0.78125 \text{ and } c=2.2160358671665 \tag{5.2}$$

If the constant b in (5.2) is small enough then the inequalities

$$t_i=\int_i^{i+1} g(x)\,dx\le p_i \quad (i=0,1,2,\ldots) \tag{5.3}$$

may hold for *all* i. The maximum b for which this is true was determined numerically for all $4\le m\le 200$. Some values are

m	4	8	9	20	50	100	200	∞
$b_{\max}$	.8264	.8427	.8450	.8580	.8677	.8726	.8760	.8841

The particular choice of c in (5.2) is well motivated: the vertices of the maximum isosceles triangle that can be placed wholly inside the standard normal curve are at $(0,1/(2\pi)^{\frac{1}{2}})$ and $(\pm c,0)$. As m goes to infinity $b_{\max}$ approaches the area of this triangle which is 0.88407040229876. The correction $-0.78125=-25/32$ in (5.2) improves the value of $b_{\max}$ for small m.

The "*Center-Triangle*" algorithm now proceeds as follows. After splitting μ into $m=[\mu]$ and $f=\mu-m$ sampling from the Poisson distribution with integer mean m is carried out first. If $m\le 8$ the multiplication method PM is applied. If $m\ge 9$ the probability function t_i/b is used in $84.375\%=27/32$ of all cases. Its fast and well-known sampling method is contained in Step 9. Much less often (probability 5/32) the remaining probabilities p_i-t_i (5.3) have to be considered. Since $b_{\max}\ge 0.8450>27/32$ for $m\ge 9$ these p_i-t_i are positive.

The "center" probability $p_m=e^{-m}m^m/m!$ is either looked up (as Q_m if $m\le L$) or calculated by means of a Stirling series (Step 10). Sampling then proceeds by comparing a uniform deviate u in (0, 1) with cumulative probabilities that are calculated in a zig-zag pattern:

Set $b\leftarrow 0.84375$. If $u\le b$ then sample from the triangle (Step 5).
Otherwise set $b\leftarrow b+p_m-t_m$ and if $u\le b$ deliver m.
Otherwise set $b\leftarrow b+p_{m-1}-t_{m-1}$ and if $u\le b$ deliver $m-1$.
Otherwise set $b\leftarrow b+p_{m+1}-t_{m+1}$ and if $u\le b$ deliver $m+1$.
Otherwise set $b\leftarrow b+p_{m-2}-t_{m-2}$ and if $u\le b$ deliver $m-2$. ....

The $p_{m-1}, p_{m+1}, p_{m-2}, p_{m+2}, p_{m-3}, \ldots$ are calculated successively in Steps 18 and 19. The t_i start at $t_m\leftarrow t_{m-1}\leftarrow b/s-b/(2s^2)$ in Step 11 and are updated to $t_{i\pm 1}\leftarrow t_i-b/s^2$ in Step 19. For the intervals in which the basis of the triangle $g(x)$ ends the correction in Step 13 applies. Thereafter all t_i are zero, and the updating of b is finally transferred from Steps 14 and 15 to the simpler Steps 16 and 17.

To the resulting sample k from Steps 9, 14, 15, 16 or 17 from the Poisson (m) distribution a Poisson (f) sample has to be added. This is helped by the table of cumulative probabilities P_i for the Poisson (1) distribution: K uniform deviates u

are generated in Step 22, K being determined in Step 21 by $P_{K-1} < u^* \leq P_K$ where u^* is uniform in (0, 1). Each time the sample count k is increased by 1 only with a probability f (Step 22). It is easy to prove that this is a correct way of sampling from Poisson (f) distributions for all $f \leq 1$.

In validating the program assurance should be obtained that b will never grow beyond 1 on account of rounding errors. This can be checked by temporarily setting $u \leftarrow 2$ in Step 8 and printing the final values of b as the procedure leaves the emergency exit in Step 16. This exit takes care of the very rare cases in which rounding errors prevent b from ever exceeding a u which is extremely close to 1. On the CDC-6400 computer all final values of b for $9 \leq m \leq 300$ were between 0.9999999999995 and 1.

Algorithm PT (*Center-Triangle Method, Ahrens*)

PREPARATIONS. Incorporate four tables:

1. *The numbers* $E_m = e^{-m}$ *for* $m = 1, 2, \ldots, 8$.
2. *The center probabilities* $Q_m = e^{-m} m^m/m!$ *for* $m = 9, 10, \ldots, L$. (*L must be large enough so that the Stirling approximation in Step* 10 *is accurate for* $m > L$; $L = 100$ *was chosen for the CDC* 6400).
3. *The square roots* $R_m = m^{\frac{1}{2}}$ *for* $i = 9, 10, \ldots, L$.
4. *The probabilities* $P_i = e^{-1} \sum_{j=0}^{i} (1/j!)$ *for* $i = 0, 1, \ldots, M$. (*M is determined by the condition that* P_N *must be* 1 *within the accuracy of the computer*; $M = 17$ *on the CDC* 6400).

PROCEDURE.

```
1. Split the mean $\mu$ into $m \leftarrow [\mu]$ and $f \leftarrow \mu - m$.
2. If $m > 8$ go to 6. If $m = 0$ set $k \leftarrow 0$ and go to 20.
3. Look up $E \leftarrow E_m$, initialize $k \leftarrow 0$, generate $u$ and set $p \leftarrow u$.
4. If $p \leq E$ go to 20.
5. Increase $k \leftarrow k + 1$. Generate $u$ and set $p \leftarrow pu$. Go to 4.
6. If $m \leq L$ look up $R \leftarrow R_m$, otherwise calculate $R \leftarrow m^{\frac{1}{2}}$.
7. Form $s = 2.2160358671665\, R - 0.78125$.
8. Generate $u$. If $u > 0.84375$ go to 10.
9. (Triangle). Generate $u_1$ and $u_2$. Set $k \leftarrow m + [s(u_1 + u_2 - 1)]$ and go to 20.
10. If $m \leq L$ look up $Q \leftarrow Q_m$, otherwise calculate
    $Q \leftarrow A/(1 + 1/(12\, m) + 1/(288\, m^2) - 139/(51840\, m^3) - 571/(2488320\, m^4))/R$
    where $A = 1/(2\pi)^{\frac{1}{2}} = 0.398942280401433$.
11. Set $b \leftarrow 0.84375$, $r \leftarrow b/s$, $g \leftarrow r/s$, $h \leftarrow g/2$, and $t \leftarrow r - h$.
12. Set $q \leftarrow p$, $i \leftarrow m$, $j \leftarrow m - 1$.
13. If $t \leq -h$ go to 16.
    If $t < h$ set $t \leftarrow (t^2/h + 2t + h)/4$.
14. Update $b \leftarrow b + p - t$.
    If $u \leq b$ set $k \leftarrow i$ and go to 20.
```

15. *Update* $b \leftarrow b+q-t$.
    If $u \leq b$ *set* $k \leftarrow j$ *and go to* 20, *otherwise go to* 18.
16. *If* $p<\varepsilon$ *go to* 1 (ε *depends on the accuracy of the computer* -2^{-49} *for a CDC*-6400 *with* 48 *bits in the mantissa*). *Update* $b \leftarrow b+p$. *If* $u \leq b$ *set* $k \leftarrow i$ *and go to* 20.
17. *If* $j<0$ *go to* 19. *Update* $b \leftarrow b+q$. *If* $u \leq b$ *set* $k \leftarrow j$ *and go to* 20.
18. *Set* $q \leftarrow qj/m$ *and then* $j \leftarrow j-1$.
19. *Set* $i \leftarrow i+1$ *and then* $p \leftarrow pm/i$. *Update* $t \leftarrow t-g$ *and go to* 13.
20. *Initialize* $i \leftarrow 0$ *and generate* u^*.
21. *If* $u^* \leq P_i$ *deliver* k.
22. *Generate* $\tilde{u}$, *and if* $\tilde{u}<f$ *then increase* $k \leftarrow k+1$.
23. *Increase* $i \leftarrow i+1$ *and go to* 21.

The CDC-6400 assembler program for CT had 102 words plus 209 words for the four tables. The computation times in Table P show that in assembler the center-triangle method is always faster than PM, and for large μ the times grow only like $c_1+c_2\,\mu^{\frac{1}{2}}$.

Another efficient algorithm is a so-called *compound method* which depends on a good sampling method for gamma distributions.

Lemma 5.1: *The following procedure samples correctly from a Poisson* (μ) *distribution.*

A. Select a positive integer n (*typically* n *is a little smaller than* μ).
B. Take a sample x *from a gamma* (n) *distribution.*
C. If $x>\mu$ *return a sample* k *from the binomial distribution with parameters* $n-1$ *and* μ/x.
D. If $x \leq \mu$ *take a sample* j *from the Poisson distribution of mean* $\mu-x$ *and return* $k \leftarrow n+j$.

Proof: In the case C one has $\mu<x<\infty$, and

$$p_k=\int_{\mu}^{\infty} \binom{n-1}{k} (\mu/x)^k (1-\mu/x)^{n-1-k}\, e^{-x}\, x^{n-1} \left(1/\Gamma(n)\right) dx$$

is the probability of obtaining k from the binomial distribution $(n-1,\ \mu/x)$. With $\Gamma(n)=(n-1)!$ this transforms easily into

$$p_k=\left(\mu^k/(k!\,(n-1-k)!)\right) \int_{\mu}^{\infty} (x-\mu)^{n-1-k} e^{-x}\, dx=\mu^k\, e^{-\mu}/k!$$

(After substituting $t=x-\mu$ the integral becomes $\Gamma(n-k)\,e^{-\mu}=(n-1-k)!\;e^{-\mu}$).

In the Case D one has $0 \leq x \leq \mu$, and

$$p_k=\int_{0}^{\mu} \left((\mu-x)^{k-n}\, e^{-(\mu-x)}/(k-n)!\right) \left(e^{-x}\, x^{n-1}/\Gamma(n)\right) dx$$

is the probability of sampling $j=k-n$ from the Poisson distribution of mean $\mu-x$.

Substituting $t=x/\mu$ yields

$$p_k=(e^{-\mu}\,\mu^{n-1}\,\mu^{k-n}\,\mu)/((n-1)!\,(k-n)!)\int_0^1 t^{n-1}\,(1-t)^{k-n}\,dt$$

The integral is the beta function $B(n,k-n+1)=(n-1)!\,(k-n)!/k!$. Hence the required expression (5.1) is obtained again.

In the explicit algorithm n is equated to $[0.875\,\mu]$. This avoids Case C almost completely if μ is large so that a simple method for sampling from the binomial distribution becomes sufficient: Algorithm BU from the next Section 6 is built into Steps 8—10 below. The algorithm for sampling from the Poisson distribution in Case D is the equally simple multiplication method PM in Steps 3—5. However, if $\mu-x$ is still larger than the "cut-off point" c then the procedure is iterated with a new auxiliary sample from a gamma distribution with parameter $n'=[0.875(\mu-x)]$ etc.

Algorithm PG (*Gamma Method, Ahrens-Dieter*)

```
1. Initialize $k \leftarrow 0$ and $w \leftarrow \mu$.
2. If $w \geq c$ go to 6.
3. (Case D). Set $p \leftarrow 1$ and calculate $b \leftarrow e^{-w}$.
4. Generate u and set $p \leftarrow pu$. If $p < b$ deliver k.
5. Increase $k \leftarrow k+1$ and go to 4.
6. Set $n \leftarrow [dw]$ where $d = 7/8$. Take a sample x from the gamma (n) distribution
   (method GO for efficiency). If $x > w$ go to 8.
7. Set $k \leftarrow k+n$, $w \leftarrow w-x$ and go to 2.
8. (Case C). Set $p \leftarrow w/x$.
9. Generate u. If $n \leq p$ increase $k \leftarrow k+1$.
10. Set $n \leftarrow n-1$. If $n > 1$ go to 9.
11. Deliver k.
```

Table P. *Poisson Distributions — Computation Times* [μsec]

Mean $\mu=$	1	2	5	10	20	50	100	200	500
PM Fortran	174	214	337	530	930	2118	4087	8025[1]	19839[1]
PM Assembler	130	145	192	265	415	858	1599	3081[1]	7527[1]
PT Fortran	267	302	412	382	404	451	504	644	789
PT Assembler	85	98	142	153	165	194	224	314	399
PG Fortran[2]	—	270[3]	347	540	780	990	1175	1285	1375
PG Assembler	—	197[3]	229	310	471	651	750	867	970

[1] Extrapolated estimates.
[2] Using the assembler routine for GO for gamma samples.
[3] Read $a=3$ instead of $a=2$ (since GO is used).

The performance of the gamma method depends on the cut-off point c. In Fortran $c=16$ proved reasonable whereas the assembler program worked best on $c=24$. The convenient value $d=7/8$ in Step 6 was also determined experimentally as a compromise.

The assembler program was only 41 words long but the main work is done inside the gamma routine GO (326 words including FL). The computation times will eventually grow only like $c_1+c_2 \, ln \, \mu$ which means that PG will ultimately become faster than PT. However in the practically important range of μ the simple minded but somewhat messy center-triangle method PT performs better than the theoretically more satisfying gamma method PG.

6. Binomial Distributions

The (ordinary) binomial (n, p) distribution

$$p_i=\binom{n}{i} p^i (1-p)^{n-i}; \; 0\leq p\leq 1; \; (i=0, 1, \ldots, n) \tag{6.1}$$

describes the probabilities p_i of having i successes in n independent trials each of which has a probability p of succeeding. This is directly translated into

Algorithm BU (*Bernoulli Process*)

```
1. Initialize $m \leftarrow k \leftarrow 0$.
2. Generate $u$. If $u \leq p$ score a success: $k \leftarrow k+1$.
3. Increase $m \leftarrow m+1$. If $m<n$ go to 2, otherwise deliver $k$.
```

The Algorithm is short (10 words in CDC 6400 assembler) but its computation times grow again linearly with the main parameter n (compare Table BI).

Much better behaved are compound Bernoulli methods based on

Lemma 6.1: *The following procedure samples correctly from a binomial (n, p) distribution.*

A. Select any two positive integers a and b for which $a+b-1=n$.
B. Take a sample s from the beta (a, b) distribution.
C. If $s\leq p$ take a sample k from the binomial distribution with parameters $(b-1, (p-s)/(1-s))$ and return $k \leftarrow a+k$.
D. If $s>p$ take a sample k from the binomial distribution with parameters $(a-1, p/s)$ and return this k.

Proof: If $s\leq p$ (case C) the probability P of returning $k+a$ *is*

$$P=\{(a+b-1)!/((a-1)!\,(b-1)!)\} \int_0^p s^{a-1} (1-s)^{b-1} \binom{b-1}{k} ((p-s)/(1-s))^k ((1-p)/(1-s))^{b-1-k} \, ds$$

$$P=\{(a+b-1)!/((a-1)!\,(b-1-k)!\,k!)\} (1-p)^{b-1-k} p^{a+k} \int_0^1 z^{a-1} (1-z)^k \, dz$$

where $z=s/p$.

The integral is $B(a, k+1)=(a-1)!\,k!/(a+k)!$. Hence

$$P=\binom{a+b-1}{a+k}\,p^{a+k}(1-p)^{b-1-k}.$$

If $s>p$ (case D) the probability of returning k is

$$P=\{(a+b-1)!/((a-1)!\,(b-1)!)\}\int_p^1 s^{a-1}(1-s)^{b-1}\binom{a-1}{k}(p/s)^k((s-p)/s)^{a-1-k}\,ds$$

$$P=\{(a+b-1)!/((b-1)!\,(a-1-k)!\,k!)\}\,p^k(1-p)^{a+b-1-k}\int_0^1 z^{b-1}(1-z)^{a-1-k}\,dz$$

where $z=(1-s)/(1-p)$.

The integral is $B(a-k, b)=(a-1-k)!\,(b-1)!/(a+b-1-k)!$ Hence

$$P=\binom{a+b-1}{k}\,p^k(1-p)^{a+b-1-k}\ \text{as required.}$$

There is a freedom of choice in Step A. If a and b are selected such that $a/(a+b)\approx p$ then $s-p$ will usually be small. Therefore one would have a small $(p-s)/(1-s)$ in Step C or a large p/s (close to 1) in Step D. This is advantageous because special fast sampling methods may be constructed for binomial distributions with small or large second parameters.

Another possibility is to choose $a=b$ and use the very efficient Algorithms BS for the symmetrical beta distribution. Preliminary studies suggested that this would be faster in the interesting ranges of n and p. Of course the condition $a+b-1=n$ under A permits $a=b$ only for odd n. In the following iterative algorithm the first parameter is split repeatedly ($a\leftarrow(m+1)/2$ in Step 4, and $m\leftarrow a-1$ in Step 6), but if it becomes even an ordinary counting step (as in BU) is inserted in Step 3. Splitting ceases as soon as m drops below the cut-off point N (Step 2) in which case BU handles the remaining binomial distribution.

Algorithm BB (*Beta Method, Ahrens-Dieter*)

```
1. Initialize $m\leftarrow n$, $q\leftarrow p$, $k\leftarrow 0$, $y\leftarrow 0$, $h\leftarrow 1$.
2. If $m\le N$ (the optimal value of $N$ in CDC-6400 assembler was $N=38$) then
   take a sample $j$ from the binomial distribution with parameters $m, q$ using the
   Bernoulli process BU. Deliver $k\leftarrow k+j$.
3. If $m$ is even decrease $m\leftarrow m-1$ and generate $u^*$. If $u^*\le q$ increase $k\leftarrow k+1$.
4. Set $a\leftarrow(m+1)/2$ and take a sample $s$ from the symmetrical beta distribution
   with parameters $a, a$ using Algorithm BS. Set $g\leftarrow hs$ and $z\leftarrow y+g$.
5. If $z\le p$ update $y\leftarrow z$, $h\leftarrow h-q$, $q\leftarrow(p-z)/h$ and $k\leftarrow k+a$. Otherwise update
   $h\leftarrow g$ and $q\leftarrow(p-y)\,h$.
6. Set $m\leftarrow a-1$ and go to 2.
```

The above version BB of the compound Bernoulli method is practically equal to the algorithm of Relles [22], except that BB is theoretically accurate whereas Relles uses an uncorrected normal approximation to the beta (a, a) distribution and studies the error.

The computation times for BB in Table BI grow only like $c\ln n$: each doubling of n adds close to 200 μ sec.

A special obvious sampling method exists for binomial distributions with parameters n and $p=1/2$. Because, if $b_1\, b_2 \dots b_n$ is a string of random bits then the number of bits that are 1 follows that distribution. In the next algorithm this approach is extended to the case of arbitrary p.

Algorithm BC (*Count-Ones, Ahrens*)

1. *Initialize $m \leftarrow n$, $i \leftarrow k \leftarrow 0$, and let $p = 0.\, p_1\, p_2\, p_3 \dots$ be the binary representation of the second parameter.*
2. *Determine the number c of 1-bits in a random bit string of length m.*
3. *Increase $i \leftarrow i+1$, and if $p_i = 1$ set $k \leftarrow k+c$. Then decrease $m \leftarrow m-c$.*
4. *If $m=0$ deliver k. Otherwise go to 2.*

The validity of the method follows from a comparison with BU. For $i=1$ the sample c in Step 2 determines the number of uniform deviates in $[0, 1/2]$ if $p_1=1$ or in $[1/2, 1]$ if $p_1=0$. In the first case these are counted as c successes and in the second case as c failures. It is obvious that the remaining $n-c$ uniform deviates are binomially distributed with parameters $n-c$ and $0.\, p_2\, p_3 \dots$ etc.

The Algorithm BC is not as completely stated as all the other methods in this article. In fact a more precise description of Step 2 must take into account internal characteristics of the computer in hand. The efficiency of BC depends on the number of bits that may be produced at the same time. On the CDC-6400 machine the first 32 bits of 48-bit normalized random mantissas were used to put together the string in Step 2. The last 16 bits were discarded since congruential generators such as (1.1) yield tail-end bit patterns that are not irregular enough.

Counting the number c of 1-bits is also machine-dependent. In a byte-oriented computer a table of c_j could be provided where $j=0, 1, \dots, 255$ and c_j is the number of one-bits in the 8-bit representation of j. On the CDC-6400 a table-free fast method was available, since the machine code contains an instruction which counts the number of 1-bits in a word. It can be seen in Table BI that the CDC computer "liked" the count-ones method, it was clearly better than BB in the range $20 \le n \le 200$. For large n BC was six to seven times faster than BU. However, since the computation times are still ultimately linear the Beta Method BB will always be superior for very large values of n.

Table BI. *Binomial Distribution* — ($p=0.5$) — *Computation Times* [μsec]

First Parameter $n=$	1	2	5	10	20	50	100	200	500	1000[1]
BU Fortran	110	140	231	384	690	1605	3133	6186	15350	30621
BU Assembler	37	47	79	132	238	555	1083	2143	5318	10629
BB Assembler	48	60	91	144	250	501	693	889	1155	1352
BC Assembler[2]	84	101	129	149	174	232	322	475	897	1611

[1] None of the three algorithms depends appreciably on the size of p. For $p=0.25$ instead of $p=0.5$ the last column ($n=1000$) reads 30020, 10701, 1351, 1625.

[2] Cut-off point $N=38$.

7. Negative Binomial Distributions

In negative binomial (a, p) distributions

$$p_i = \binom{a-1+i}{i} p^a (1-p)^i; \; a>0, \; 0 \le p \le 1; \; i=0,1,2,\ldots \tag{7.1}$$

the first parameter a need not be an integer. If it is then p_i is the probability of observing i failures *before* the a-th success in a sequence of n independent trials each of which has a probability p of succeeding:

Algorithm NU (*a integer*)

```
1. Initialize $m \leftarrow k \leftarrow 0$.
2. Generate $u$.
3. If $u > p$ set $k \leftarrow k+1$ and go to 2.
4. Set $m \leftarrow m+1$. If $m < a$, go to 2, otherwise deliver $k$.
```

If a is not an integer, and if the mean $\mu = a(1-p)/p$ of the negative binomial distribution is not large, ordinary linear search may be satisfactory. For this a uniform deviate u is compared with the cumulative probabilities $\sum p_i$ which are updated in the variable p below.

Algorithm NS

```
1. Initialize $k \leftarrow 0$, $p \leftarrow p^a$, $P \leftarrow p$, $q \leftarrow 1-p$, $A \leftarrow (a-1)q$. Generate $u$.
2. If $u \le P$ deliver $k$.
3. Increase $k \leftarrow k+1$ and set $p \leftarrow p(q+A/k)$ and $P \leftarrow P+p$. Go to 2.
```

The computation times of NS in Table NB grow only as fast as the ones of NU, but they start at a higher level. This means that for $p=0.5$ NU should be used if a is an integer. However, the formulae in the last column suggests that NS becomes faster than NU if $p>0.5$ provided that a is not too small.

Again better methods for large parameters a are desirable. In the case of integer a an algorithm was studied which uses sampling from the ordinary binomial distribution as an intermediate step. Slightly more successful proved the following compound method which works for all $a>0$.

Algorithm NG (*Léger* [17])

```
1. Take a sample $x$ from the standard gamma distribution with parameter $a$.
2. Deliver a sample $k$ from the Poisson distribution of mean $\mu \leftarrow x(1-p)/p$.
```

This is known as the compound Poisson process (compare Fisz [12]). The proof is quite short:

Let $b = p/(1-p)$. Since μb is gamma (a) distributed one has

$$p_i = \int_0^\infty (e^{-\mu} \mu^i / i!) \, (e^{-b\mu} (b\mu)^{a-1} / \Gamma(a)) \, d(b\mu).$$

substituting $s \leftarrow (b+1)\,\mu$ yields

$$p_i = b^a / \left(i!\,\Gamma(a)\,(b+1)^{a+i}\right) \int_0^\infty s^{a+i-1}\, e^{-s}\, ds.$$

The integral is $\Gamma(a+i)$ and with $p = b/(b+1)$ the expression (7.1) for p_i is easily derived.

The computation times of NG can always be predicted from the times for gamma (a) and Poisson (μ) samples where the μ in Step 2 may be replaced with the expected value $a(1-p)/p$. The observed times in Table NB refer to the submethods GO and CT. They indicate that these very efficient algorithms make the compound method worthwhile if $a > 20$. .

Table NB. *Negative Binomial Distribution* — $(p = 0.5)$ — *Computation Times* [μsec]

First parameter $a =$	1	2	5	10	20	50	100	1000	Formula[1]
NU Assembler	48	70	138	247	463	1117	2207	21807[2]	$27 + 10.9\,a/p$
NS Assembler	293	315	380	489	708	1363	2452	22072[2]	$272 + 21.8\,a(1-p)/p$
NS Fortran[4]	—	557[3]	574	573	544	528	565	771	—

[1] The formulae permit estimates for $p \neq 0.5$.
[2] Extrapolated estimates.
[3] Read $a = 3$ instead of $a = 2$.
[4] Using assembler routines GO and CT for gamma and Poisson deviates.

References

[1] Ahrens, J. H., and U. Dieter: Computer methods for sampling from the exponential and normal distributions. Comm. ACM **15**, 873—882 (1972).
[2] Ahrens, J. H., and U. Dieter: Extensions of Forsythe's method for random sampling from the normal distribution. Math. Comput. **27**, 927—937 (1973).
[3] Ahrens, J. H., U. Dieter, and A. Grube: Pseudo-random numbers: A new proposal for the choice of multiplicators. Computing **6**, 121—138 (1970).
[4] Bankövi, G.: A note on the generation of beta-distributed and gamma-distributed random variables. Publ. Math. Inst. Hung. Acad. Scie. **9**, 555—563 (1964).
[5] Békéssy, A.: Remarks on beta-distributed random numbers. Publ. Math. Inst. Hung. Acad. Scie. **9**, 565—571 (1964).
[6] Dieter, U.: Autokorrelation multiplikativ-erzeugter Pseudo-Zufallszahlen. Operations Res. Verfahren **6**, 69—85 (1969).
[7] Dieter, U.: Pseudo-random numbers: The exact distribution of pairs. Math. Comput. **25**, 855—883 (1971).
[8] Dieter, U.: Pseudo-random numbers: Permutations of triplets. J. Res. Nat. Bureau Standards (to appear).
[9] Dieter, U., and J. H. Ahrens: An exact determination of serial correlations of pseudo-random numbers. Numer. Math. **17**, 101—123 (1971).
[10] Dieter, U., and J. H. Ahrens: A combinatorial method for the generation of normally distributed random numbers. Computing **11**, 137—146 (1973).
[11] Dieter, U., and J. H. Ahrens: Pseudo-random numbers. Preliminary version in preprint, 1972. (340 pages.)
[12] Fisz, M.: Probability Theory and Mathematical Statistics. New York: J. Wiley. 1962.
[13] Forsythe, G. E.: Von Neumann's comparison method for random sampling from the normal and other distributions. Math. Comput. **26**, 817—826 (1972).

[14] Jöhnk, M. D.: Erzeugen von betaverteilten und gammaverteilten Zufallszahlen. Metrika **8**, 5—15 (1964).

[15] Jöhnk, M. D.: Erzeugen und Tasten von Zufallszahlen. (Berichte aus dem Institut für Statistik und aus dem Institut für Angewandte Statistik der FU Berlin, Heft 6.) Würzburg: Physica-Verlag. 1969.

[16] Knuth, D. E.: The art of computer programming, Vol. II: Seminumerical algorithms. Reading, Mass.: Addison Wesley. 1969.

[17] Léger, R.: On sampling from the negative binomial and Weibull distributions. Master's thesis, Dalhousie University, Halifax, N.S., 1973.

[18] MacLaren, M. D., G. Marsaglia, and T. A. Bray: A fast procedure for generating normal random variables. Comm. ACM **7**, 4—10 (1964).

[19] Marsaglia, G.: Expressing a random variable in terms of uniform random variables. Ann. Math. Stat. **32**, 894—899 (1961).

[20] Marsaglia, G.: Random variables and computers. Trans. Third Prague Conf. Information Theory, Statistics and Decision Functions. Prague 1964, 499—512.

[21] Payne, W. H., and T. G. Lewis: Conditional bit sampling: Accuracy and speed. Math. Software (Rice, J. R., ed.). Academic Press. 1971.

[22] Relles, D. A.: A simple algorithm for generating binomial random variables when N is large. J. Am. Stat. Ass. **67**, 612—613 (1972).

[23] von Neumann, J.: Various techniques in connection with random digits. Monte Carlo Methods. Nat. Bureau Standards, AMS **12**, 36—38 (1951).

Dr. J. H. Ahrens
Nova Scotia Technical College
Department of Applied Mathematics
Halifax, N.S., B3J 2X4
Canada

Dr. U. Dieter
Lehrkanzel und Institut für
Mathematische Statistik
Hamerlinggasse 6, VI
A-8010 Graz
Austria