Ann. Inst. Statist. Math.
30 (1978), Part A, 271-279

ON A GENERATION OF NORMAL PSEUDO-RANDOM NUMBERS

HIROTAKA SAKASEGAWA

(Received May 24, 1978; revised July 13, 1978)

1. Introduction

It goes without saying that uniform random numbers play an important role in statistical simulations. In many situations, however, they are used after being transformed into variables with given distribution rather than uniform distribution. There are several works on generating normal random numbers. Among others are Box and Muller [5], Marsaglia [9], Marsaglia and Bray [11], Marsaglia, MacLaren and Bray [12], Ahrens [1], Ahrens and Dieter [2], [3], Fosythe [7], Brent [6], Kinderman and Ramage [8], Shimizu [15] and so on.

Recently, Atkinson and Pearce [4] discussed the comparative merits of some of these generators. It seems that they and the debaters of that paper (see also [4]) were not satisfied with existing algorithms of the normal random number generation, and we are still in need of faster algorithm with a reasonable memory occupation. In this paper, we propose two new algorithms of generating normal pseudo-random numbers, the one is exact in principle and the other approximate by nature. These two algorithms are recommended in that they require less generation time and in that they are simple in construction. Algorithms of an approximate normal random number generation have not necessarily been appraised high (for example, see [4] cited above), but if an algorithm of this kind produces well approximated sequence quickly, then it would deserve consideration for most simulation experiments. Our algorithms and their theoretical background are developed in Section 2. Brief survey of many existing algorithms is given in Section 3. Results of timing tests are discussed in Section 4.

2. Algorithms of our new generators

2.1. *Algorithm for generating exact sequences*

Our first algorithm is based on the following three lemmas. These lemmas are easy to prove and proofs are omitted.

LEMMA 1. *Let* X_i, $i=1, 2$ *be independent random variables* (*r.v.*)

distributed uniformly on $[a_i, b_i]$ *and let* c_i, $i=1, 2$ *be positive constants such that* $c_1(b_1-a_1) \geqq c_2(b_2-a_2)$. *Then the density* $f(\cdot)$ *of* $c_1X_1+c_2X_2$ *is given by*

$$f(x)=\begin{cases} 0 & \text{if } x \leqq c_1a_1+c_2a_2 \text{ or } x \geqq c_1b_1+c_2b_2 , \\ \dfrac{x-c_1a_1-c_2a_2}{c_1c_2(b_1-a_1)(b_2-a_2)} & \text{if } c_1a_1+c_2a_2 < x < c_1a_1+c_2b_2 , \\ \dfrac{1}{c_1(b_1-a_1)} & \text{if } c_1a_1+c_2b_2 \leqq x < c_1b_1+c_2a_2 , \\ \dfrac{-x+c_1b_1+c_2b_2}{c_1c_2(b_1-a_1)(b_2-a_2)} & \text{if } c_1b_1+c_2a_2 \leqq x < c_1b_1+c_2b_2 . \end{cases}$$

LEMMA 2. *Let* $F(\cdot)$ *be a distribution function which has the bounded p.d.f.* $f(\cdot)$ *vanishing outside* $[a, b]$. *Let* c *be the supremum of* $f(x)$ *and let* X *and* Y *be r.v.'s distributed uniformly on* $[a, b]$ *and* $[0, c]$, *respectively. Then*

$$\Pr(X<x \mid Y<f(X))=F(x) .$$

LEMMA 3 ([10]). *Let* $\Phi(\cdot)$ *be the distribution function of the standard normal distribution and let* X *and* Y *be r.v.'s distributed uniformly on* $[0, 1]$. *Then for any* $x>a>0$,

$$\begin{aligned} &\Pr(a^2-2\log(X)<x^2 \mid Y^2(a^2-2\log(X))<a^2) \\ &\quad=(\Phi(x)-\Phi(a))/(1-\Phi(a)) . \end{aligned}$$

Now, let $f_i(x)$, $i=1, \cdots, k$ be p.d.f.'s with the shape like a symmetric trapezoid which is constant on $[-x_i, x_i]$ and is zero outside $[-x_{i+1}, x_{i+1}]$ for $0(=x_0)<x_1<x_2<\cdots<x_{k+1}<\infty(=x_{k+2})$. For given k, determine p_i, $i=1, \cdots, k$ and x_i, $i=1, \cdots, k+1$ in such a way that $\sum_{i=1}^{k} p_i \cdot f_i(x)$ approximates the standard normal density $\varphi(x)$ from below as close as possible. Later, we shall determine p_i's and x_i's heuristically, which is sufficient for our purpose. Next, let $g_i(x)$, $i=1, \cdots, k+2$ be defined as follows:

$$g_i(x)=\begin{cases} h(x)/2\int_{x_{i-1}}^{x_i} h(x)dx & \text{if } x_{i-1}<|x|<x_i , \\ 0 & \text{otherwise} , \end{cases} \tag{2.1}$$

where $h(x)=\varphi(x)-\sum_{i=1}^{k} p_if_i(x)$. Then, we can represent $\varphi(x)$ as a mixture of $2k+2$ densities as follows:

$$\varphi(x)=\sum_{i=1}^{k} p_i f_i(x)+\sum_{i=1}^{k+1} p_{k+i} g_i(x)+p_{2k+2} g_{k+2}(x) \tag{2.2}$$

where $p_{k+i}=2\int_{x_{i-1}}^{x_i} h(x)dx$ $(i=1,\cdots,k+2)$. A random variate with the density $f_i(x)$, $i=1,\cdots,k$ can be generated from two uniform random numbers according to Lemma 1. A random variate with the density $g_i(x)$, $i=1,\cdots,k+1$ can be generated by the rejection method based on Lemma 2. Finally, a random variate with the density $g_{k+2}(x)$ can be generated by the tail method based on Lemma 3. For the later use, write

$$P=p_1+\cdots+p_k \quad \text{and} \quad F(x)=\sum_{i=1}^{k}(p_i/P)f_i(x)\ . \tag{2.3}$$

Then the first term of the RHS of (2.2) becomes $P\cdot F(x)$, which we call the principal part because the value of P has a great influence on generation speed. The formal description of our algorithm is given below. In the list, u_i denotes a $(0, 1)$-uniform random number and Q_j denotes the sum of p_i's from 1 to j.

Algorithm I.

1. If $Q_{j-1}<u_1\leqq Q_j$ then deliver $y=a_{j1}u_1+a_{j2}u_2+a_{j3}$ $(j=1,\cdots,k)$.
2. If $u_1>Q_{2k+1}$ then go to 5.
3. If $Q_{k+j-1}<u_1\leqq Q_{k+j}$ then go to 4(j) $(j=1,\cdots,k+1)$.

4(j). If $h(b_{j1}|u_2-0.5|+b_{j2})<b_{j3}u_3$ then repeat 4(j) using another u_2 and u_3, else deliver $y=\text{sign}\,(u_2-0.5)(b_{j1}|u_2-0.5|+b_{j2})$ $(j=1,\cdots,k+1)$.

5. If $(u_2-0.5)^2(x_{k+1}^2/2-\log(u_3))>x_{k+1}^2/8$ then repeat testing with new u_2 and u_3, else deliver $y=\text{sign}\,(u_2-0.5)\sqrt{x_{k+1}^2-2\log(u_3)}$, where

$$a_{j1}=(x_{j+1}-x_j)/p_j\ ,\quad a_{j2}=x_j+x_{j+1}\ ,\quad a_{j3}=(Q_jx_j-Q_{j+1}x_{j+1})/p_j \qquad (j=1,\cdots,k)\ ,$$

$$b_{j1}=2(x_j-x_{j-1})\ ,\quad b_{j2}=x_{j-1} \quad \text{and} \quad b_{j3}=\sup_{x_{j-1}<x<x_j} h(x) \qquad (j=1,\cdots,k+1)\ .$$

Using our algorithm, average number of uniform random numbers required to produce one normal random number is given by

$$1+Q_k+4\sum_{i=1}^{k+1} b_{i3}(x_i-x_{i-1})+\sqrt{\frac{8}{\pi}}\frac{1}{x_{k+1}}\exp\left(-\frac{x_{k+1}^2}{2}\right)\ .$$

Now, we give some numerical examples. For $k=5$, we use the values 0.1726, 0.5410, 1.5085, 1.9499, 2.4520 and 3.1650 for x_j's and 0.0345,

0.4530, 0.2361, 0.1755 and 0.0868 for p_j's. These values were determined so as to make b_{j3}'s as close to each other as possible. In this case, P in (2.3) becomes 0.9860, that is, only two uniform random numbers are needed in 98.60 percent of all cases and it requires 2.046 uniform random numbers on an average for single normal random number generation. If $k=4$ or 6, P becomes, respectively, 0.9806 or 0.9889 using the similar procedure to determine x_j's and p_j's. Generally speaking, the expected number of uniform random numbers required can be reduced by letting k large, but on the other hand the larger k becomes, the more steps for comparison are needed and this might result in the increase of generation time. The optimal value for k is not known as yet. But our numerical experiments in Section 4 shows that $k=5$ or 6 is a reasonable choice. A copy of complete Fortran subprogram will be delivered on request.

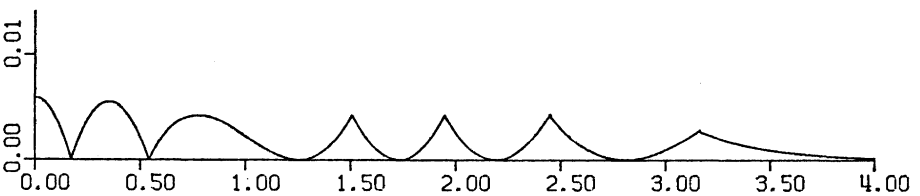

Fig. 2.1. The value of $\varphi(x)-P\cdot F(x)$ for $k=5$.

2.2. *Algorithm for generating approximate sequences*

There are many algorithms to generate approximate normal random numbers (see Muller [13]). We can regard an algorithm as of practical value in many simulation experiments if it generates a well approximated normal sequence quickly. Now we propose one such algorithm below.

The dissatisfaction of the approximative algorithm mainly arises from the inaccuracy of approximation at the tail. In order to avoid this, the proposed algorithm uses the exact method for the tail based on Lemma 3. For the remainder, a random number is generated approximately by, what you call, the inverse function method. Now, two questions arise: How can we decide the interval where the exact method should be applied? How can we determine the inverse function? Again, we took a heuristic approach and obtained the satisfactory result. First, we divide the unit interval [0, 1] into n congruent subintervals and construct a quadratic function $g_i(x)$, $i=n_0, n_0+1, \cdots, n-n_0-1$ for some n_0 which passes through three points

$$\left\{\frac{i}{n}, \Phi^{-1}\left(\frac{i}{n}\right)\right\}, \quad \left\{\frac{2i+1}{2n}, \Phi^{-1}\left(\frac{2i+1}{2n}\right)\right\} \quad \text{and} \quad \left\{\frac{i+1}{n}, \Phi^{-1}\left(\frac{i+1}{n}\right)\right\}.$$

Let $G(x)$ be the piecewise quadratic function defined on $[n_0/n, (n-n_0)/n]$ as follows:

$$G(x)=g_i(x) \qquad \text{if } \frac{i}{n}\leqq x\leqq\frac{i+1}{n} \quad (i=n_0, n_0+1, \cdots, n-n_0+1).$$

$G^{-1}(x)$ is identical with $\Phi(x)$ at $x=\Phi^{-1}(i/2n)$, $i=2n_0, 2n_0+1, \cdots, 2(n-n_0)$ and approximates $\Phi(x)$ well for all $x\in[\Phi^{-1}(n_0/n), \Phi^{-1}((n-n_0)/n)]$ if n is sufficiently large and if n_0 is carefully chosen. Now, we use the tail method to generate a number less than $\Phi^{-1}(n_0/n)$ or greater than $\Phi^{-1}((n-n_0)/n)$, otherwise, we use the inverse function method with $G(x)$. As an example, the formal description of this algorithm for $n=64$ and $n_0=2$ is given below.

Algorithm II.

1. Set $v=u_1-0.5$ and $i=[64|v|]+1$. If $i>30$ then go to 2, else deliver $y=\text{sign}\,(v)(a_iv^2+b_i|v|+c_i)$.
2. Set $x=1.734868-\log(u_2)$. If $u_3^2x>1.734868$ then repeat 2 using another u_2 and u_3, else deliver $y=\text{sign}\,(v)\sqrt{2x}$.

(Here, $1.734868=(\Phi^{-1}(62/64))^2/2$ and $[x]$ denotes the maximum integer which does not exceed x).

Constants used in the above algorithm are contained in Table 2.1. Average number of uniform random numbers required to produce single normal random number is 1.151 in this case, and thus, the generation speed is very high. The maximum discrepancy between the resulting and the normal densities and between the corresponding distribution functions are, respectively, 2×10^{-3} at $x=1.68$ and of the order of 10^{-6}. Note that these functions are identical with each other for $|x|>\Phi^{-1}(62/64)\simeq1.86$. Such an aspect of the approximation is well satisfactory for many simulation experiments. The dimension of Table 2.1 is 90. If we double it, the maximum discrepancy between the densities will reduce to 4×10^{-4}.

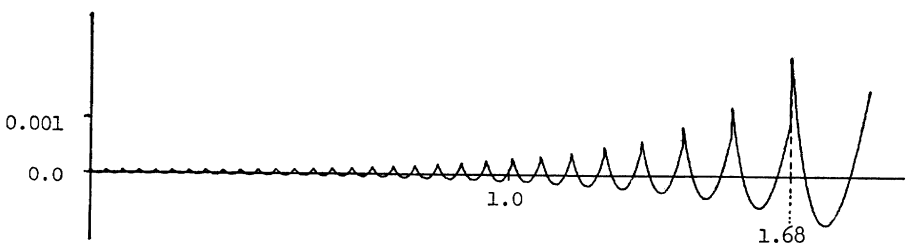

Fig. 2.2. Discrepancy between the approximate density and the normal density (with 64 subintervals).

Table 2.1. a_i's, b_i's and c_i's for Algorithm II

i	a_i	b_i	c_i
1	0.061531875	2.506324066	0.000000000
2	0.185457436	2.502448720	0.000030302
3	0.310607553	2.494626574	0.000152508
4	0.439866899	2.482515412	0.000436201
5	0.574073735	2.465740308	0.000960400
6	0.714563323	2.443790469	0.001817750
7	0.863567748	2.415849934	0.003127566
8	1.023856001	2.380775749	0.005046295
9	1.196429166	2.337595377	0.007747385
10	1.384198042	2.284758948	0.011464305
11	1.592263939	2.219712172	0.016548130
12	1.823859880	2.140066906	0.023396172
13	2.085205936	2.041990229	0.032596898
14	2.380960409	1.921753693	0.044817185
15	2.721912796	1.772497452	0.061151883
16	3.117560074	1.586917796	0.082913561
17	3.584029092	1.353535804	0.112104745
18	4.140651703	1.057630629	0.151432547
19	4.812440255	0.679473816	0.204648044
20	5.640401315	0.187520475	0.277724538
21	6.676103018	-0.460277197	0.379018566
22	8.009673837	-1.336163856	0.522838620
23	9.741249907	-2.527809750	0.727856833
24	12.100811489	-4.225193662	1.033115876
25	15.395378078	-6.698599578	1.497344668
26	20.258171357	-10.501530850	2.240861327
27	27.861081570	-16.686245664	3.498624547
28	40.812422555	-27.628516409	5.809834573
29	65.889434878	-49.606380015	10.625253011
30	125.601552561	-103.834168753	22.936996601

3. Other algorithms

There are many algorithms to generate normal pseudo-random numbers. Some of them are summarized below which are used in Section 4 for the purpose of timing tests.

[BM] (Box and Muller [5]). This is a well known and widely used classical algorithm using trigonometric and logarithmic functions. The programming effort may be the least among other existing algorithms, but generation speed is much slower because of using basic external functions in Fortran program.

[PO] (Marsaglia [9]). This algorithm is the same as [BM] in principle but it does not make use of trigonometric function.

[BR] (Brent [6]). This algorithm is a refinement of the Forsythe algorithm [7]. Generation speed is not fast in Fortran program because it requires a bit-level information.

Following algorithms are all constructed from three parts, 'quick-generation' part or principal part, rejection-method part and tail-method part, similar to our algorithm I. We calculate for each algorithm the probability P that a random number is generated by using the principal part.

[MB] (Marsaglia and Bray [11], Newman and Odell [14]). The density of a principal part is a mixture of two convolutions, a convolution of three uniform densities and a convolution of two uniform densities. $P=0.9745$.

[AH] (Ahrens and Dieter [1]). This algorithm is a special case of our algorithm with $k=1$. $P=0.91954$.

[KR] (Kinderman and Ramage [8]). The density of a principal part is triangular-shaped and the second part utilizes the Marsaglia-MacLaren-Bray technique ([12]). $P=0.88407$.

[SM] (Shimizu [15]). The density of a principal part is a convolution of a uniform density and a step function. $P=0.932$.

In the following, we code our algorithms as [SKk] for the one in Subsection 2.1 where k is the same notation as that section and [QD] for the other in Subsection 2.2.

4. Results of comparison tests

In this section, we summarize results of comparison tests for algorithms described in the preceding sections. For the tests, we used an electronic computer HITACHI H-8700/OS7 which is installed in the Institute of Statistical Mathematics. As it is drived under multiprogramming mode with multiplicity 5, elapsed time varies each occasion. We measured the CPU time to generate 20 thousand pseudo-random numbers ten times for each algorithm and averaged all 10 figures. All these algorithms are implemented as Fortran programs so we must use CALL-statement to get a normal random number. The time for linkage with main program and the generator-subprogram which takes 42.56 microseconds on an average is excluded, but the time for DO-loop which is negligibly small is included. For the uniform random number generator (UR generator, for short), we used the Lehmer (multiplicative) congruential method. The machine we used has 32 bits for one word and multiplication of two integers produces 32-bit integer which is the ordinary product modulo 2^{32}. That is to say, the Lehmer method is easily implemented as in-line generator. The first test was executed using this (in-line) generator. From the point of view of generation speed, this type of a generator is preferable. If uniform random numbers are furnished by some other algorithm, e.g. a physical random

Table 4.1. Results of timing tests

Generator	Test 1	Test 2	Generator	Test 1	Test 2
[SK4]	35.91	120.33	[SM]	45.98	140.04
[SK5]	32.48	119.12	[MB]	41.21	163.13
[SK6]	32.53	120.83	[BM]	138.87	179.66
[AH]	42.34	131.45	[BR]	65.58	—
[PO]	79.28	134.60	[QD]	25.85	89.40
[KR]	44.46	134.98			

device, the separate subroutine subprogram must be prepared. This causes the generation of normal random numbers much time-consuming because of the linkage requirement. Accordingly, the saving of uniform random numbers is quite important to reduce generation time in this case. The second test was executed using the (subroutine) generator. The results of two tests are listed in Table 4.1. They are summarized as follows.

(a) Excepting the approximate generator [QD], the fastest generator is [SK5] and the differences between [SKk] and other algorithms are significant.
(b) [QD] generator is extremely fast, so it is recommended as far as circumstances permit.
(c) [MB] generator, which is recommended by Atkinson et al. ([4]), is not good if the UR generator cannot built in, but it is rather good if in-line UR generator is available.
(d) [PO] generator is surprisingly good if the UR generator is provided by an exterior subroutine.

5. Conclusion

In this paper, we proposed two new algorithms to generate a normal sequence, the one generates exactly and the other generates approximately. Both algorithms work much better than the other algorithms and are recommended from a standpoint of generation speed.

On the other hand, the quality of the resulting sequence is the another problem. Almost all existing algorithms including those in this paper generate a normal sequence by transforming the uniformly random sequence and an 'exact' algorithm is exact only when the basic random sequence behaves as truly random. Our algorithms also share this limitation. It is conceivable that the non-uniformity and/or dependence of the basic sequence have a bad effect on the quality of the generated normal sequence, but all through our experiments, such a situation has not occurred. To know the relation between the quali-

ties of both sequences is very important unknown problem and further studies will be required.

Acknowledgement

The author would like to express his gratitude to the stimulative referee who helped him in revising the original version of his paper.

THE INSTITUTE OF STATISTICAL MATHEMATICS

REFERENCES

[1] Ahrens, J. H. and Dieter, U. (1972). Computer methods for sampling from the exponential and normal distributions, *Comm. ACM*, **15**, 873-882.

[2] Ahrens, J. H. and Dieter, U. (1973). Extension of Forsythe's method for random sampling from the normal distributions, *Math. Comp.*, **27**, 927-937.

[3] Ahrens, J. H. and Dieter, U. (1974). Computer methods for sampling from gamma, beta, Poisson and binomial distributions, *Computing*, **12**, 223-246.

[4] Atkinson, A. C. and Pearce, M. C. (1976). The computer generation of beta, gamma and normal random variables, *J. R. Statist. Soc.*, A**139**, 431-461.

[5] Box, G. E. P. and Muller, M. E. (1958). A note on the generation of normal random deviates, *Ann. Math. Statist.*, **29**, 610-611.

[6] Brent, R. P. (1974). A Gaussian pseudo-random number generator, *Comm. ACM*, **17**, 704-706.

[7] Forsythe, G. E. (1972). Von Neumann's comparison method for random sampling from the normal and other distributions, *Math. Comp.*, **26**, 817-826.

[8] Kinderman, A. J. and Ramage, J. G. (1976). Computer generation of normal random variables, *J. Amer. Statist. Ass.*, **71**, 893-896.

[9] Marsaglia, G. (1962). Improving the poler method for generating a pair of random variables, Boeing Sci. Res. Lab. D1-82-0203.

[10] Marsaglia, G. (1964). Generating a variable from the tail of the normal distribution, *Technometrics*, **6**, 101-102.

[11] Marsaglia, G. and Bray, T. A. (1964). A convenient method for generating normal variables, *SIAM Rev.*, **6**, 260-264.

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

[13] Muller, M. E. (1961). A comparison of method for generating normal deviates on digital computers, *J. ACM*, **6**, 376-383.

[14] Newman, T. G. and Odell, P. L. (1971). *The Generation of Random Variates*, Charles Griffin, 22-26.

[15] Shimizu, R. (1976). *Central Limit Theorem* (in Japanese), Kyoiku Shuppan, 232-243.