J_ID: **z8m** Customer A_ID: **10-0145.R2** Cadmus Art: **NAV21474** KGL ID: **JW-NAVT110078** — 2011/12/23 — page 1 — #1

Rectangles Algorithm for Generating Normal Variates

Rui Zhang,[1] Lawrence M. Leemis[2]

[1] *Department of Decision and Information Technologies, The Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742–1815*

[2] *Department of Mathematics, The College of William & Mary, Williamsburg, Virginia 23187–8795*

Received 3 September 2010; revised 23 November 2011; accepted 29 November 2011
DOI 10.1002/nav.21474
Published online in Wiley Online Library (wileyonlinelibrary.com).

Abstract: We propose an algorithm for generating normal random variates that is based on the acceptance–rejection method and uses a piecewise majorizing function. The piecewise function has 2048 equal-area pieces, 2046 of which are constant, and the two extreme pieces are curves that majorize the tails. The proposed algorithm has not only good performance from correlation induction perspective, but also works well from a speed perspective. It is faster than the inversion method by Odeh and Evans and most other methods. © 2011 Wiley Periodicals, Inc. Naval Research Logistics 00: 000–000, 2011

Keywords: acceptance–rejection algorithm; correlation induction; normal distribution; random variate generation

1. INTRODUCTION

We propose an algorithm for generating normal random variates that is based on the acceptance–rejection method and uses a piecewise majorizing function. The piecewise function has 2048 pieces, 2046 of which are constant, and the two extreme pieces are curves that majorize the tails. The proposed algorithm is fast because the appropriate rectangle for generating is chosen quickly since all majorizing rectangles and the majorizing tails have the same area. The large number of pieces makes rejection very unlikely. Variates from the tails are generated by Marsaglia's [14] tail algorithm. The proposed algorithm is compared to the ziggurat, Ahrens and Dieter, Leva, Odeh and Evans (OE), Box–Muller (BM), improved Box–Muller (IBM), and Kinderman and Ramage (KR) algorithms. It ranks second in speed behind the ziggurat method. Furthermore, the proposed algorithm has good performance from correlation induction perspective. With minor changes suggested by Schmeiser and Kachitvichyanukul [22], the correlation between the uniform random variate and the normal random variate is approximately 0.996. The more important property is that the rectangles algorithm is much faster than the inversion method by OE.

Correspondence to: Lawrence Leemis (leemis@math.wm.edu)

1.1. Literature Review

A $N(\mu, \sigma^2)$ random variate X is generated by $X = \mu + \sigma Z$, where $Z \sim N(0, 1)$. So, the focus here is on generating standard normal variates. The earliest algorithms for generating standard normal random variates were approximate algorithms, such as summing 12 independent random numbers and subtracting 6, which matches the first three moments, gives a symmetric probability density function (PDF), and is approximately normal by the central limit theorem. The BM algorithm [2] was the first exact algorithm for generating standard normal random variables. Marsaglia and Bray [16] improved the BM algorithm with their "polar method" that avoids calls to trigonometric functions. Both of these algorithms generate two independent standard normal random variates, which means that one of the variates must either be stored for a subsequent generation or discarded. Kinderman and Ramage [9] inscribed a triangle under the PDF of a standard normal random variable, resulting in fast generation over 88% of the time, and an acceptance–rejection or tail algorithm the rest of the time. Ahrens and Dieter [1] adapted Walker's [23–25] "alias method" to generate standard normal variates. Leva [13] presented a modification of the ratio method of Kinderman and Monahan [8]. More recently, Wallace [26] proposed a method that deviates from common practices. Without transforming uniform variates, it directly generates a sequence of standard normal variates using a recurrence equation. Marsaglia and Tsang [17, 18]

J_ID: **z8m** Customer A_ID: **10-0145.R2** Cadmus Art: **NAV21474** KGL ID: **JW-NAVT110078** — 2011/12/23 — page 2 — #2

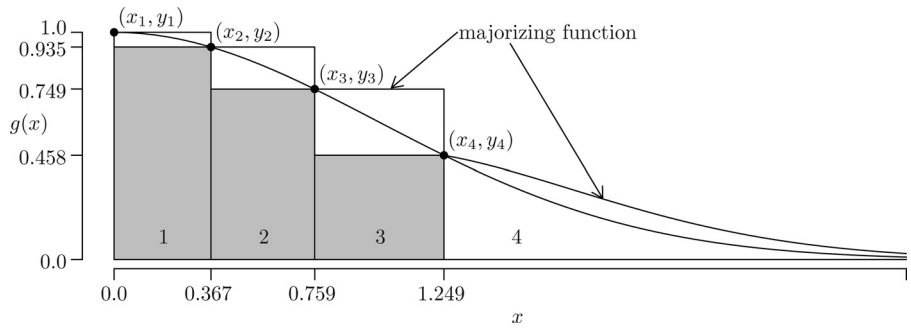

Figure 1. The geometry behind the algorithm for $n = 4$ with $a \cong 0.367$.

refined their ziggurat method to enhance efficiency. They simplified the generating procedure and compared its speed to existing methods. The ziggurat method seems to be the fastest known method based on their results and Doornik [6].

Another group of methods uses approximations of the inverse of cumulative distribution function (CDF) of the standard normal distribution to generate standard normal variates. Inversion can be advantageous for implementing certain variance reduction techniques. They can be assessed by computing the maximum absolute error. Brophy [3] performed a comparative study of them. Among these approximations, the algorithm of Odeh and Evans [20] has the best balance between accuracy and efficiency. Moro [19] contains an implementation of an inverse transformation algorithm. Also, the empirical distribution method of Chen and Asau [5] can be applied to standard normal distribution. Several other methods are also available for generating normal variates. Johnson et al. [7], Knuth [10], and Law [11] provide an extensive list of the algorithms.

2. ALGORITHM

2.1. Formulation

The PDF for a $N(\mu, \sigma^2)$ random variable is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

for $-\infty < x < \infty$. We will focus on generating a standard normal random variate Z with $\mu = 0$ and $\sigma = 1$ because we are able to transform it to a $N(\mu, \sigma^2)$ random variate via $X = \mu + \sigma Z$. Also, we develop our algorithm based on

$$g(x) = \exp(-x^2/2)$$

for simplicity and efficiency. Furthermore, because $g(x)$ is an even function, we can focus only on the right half of the PDF to generate variates and assign the sign to the variates in the last step.

We use $n - 1$ rectangles as the majorizing function for **F1** the function $g(x)$ on $0 \le x < \infty$ as illustrated in Fig. 1 for $n = 4$, where n is the number of equal-area pieces in the majorizing function. Denote the total area under the majorizing function from 0 to ∞ by A. Each majorizing rectangle and the majorizing tail have the same area. Denote the area under each piece of the majorizing function by a, which equals A/n. From left to right, label the rectangles as $1, 2, \ldots, n-1$. For rectangle i, there are two corresponding delimiting x-values, x_i and x_{i+1} for $i = 1, 2, \ldots, n-1$. So, we have $x_1 < x_2 < \cdots < x_n$. As $g(x)$ is a monotone decreasing function on $0 \le x < \infty$, we also have $y_i = g(x_i)$ for each x-value and $y_1 > y_2 > \cdots > y_n$. Regardless of n, the first (x_i, y_i) pair is always $(x_1, y_1) = (0, 1)$. The shaded regions in Fig. 1 are described subsequently.

2.2. Constructing the Rectangles

We use n equal-area pieces to majorize the curve in the Fig. 1 to achieve computational efficiency. So, $x_2 = a/y_1 = a$ and $y_2 = g(x_2)$. Then, $(x_3 - x_2)y_2 = a$ and $y_3 = g(x_3)$, and so on. Hence, we have the following system of equations in the $n-1$ unknowns $x_2, x_3, \ldots, x_n$:

$$\begin{gathered} x_2 = a \\ y_i = g(x_i), i = 2, 3, \ldots, n \\ (x_{i+1} - x_i)y_i = a, i = 2, 3, \ldots, n-1 \\ \int_{x_n}^{\infty} \frac{x}{x_n} \exp(-x^2/2)dx = a. \end{gathered}$$

For the tail portion, the majorizing function is

$$t(x) = \frac{x}{x_n} \exp(-x^2/2)$$

where $x \ge x_n$ as suggested by Bratley et al. [4].

The solution to this system of equations gives us the x-values. Based on this, we are able to calculate the corresponding y-values and construct the rectangles. Table 1 contains **T1** the x-values for $n = 2$, $n = 4$, and $n = 8$. The x-values for $n = 16$, $n = 32$, $n = 64$, $n = 128$, $n = 256$, $n = 512$, and $n = 1024$ and the MATLAB code to calculate the x-values are available at http://www.math.wm.edu/∼leemis.

J_ID: **z8m** Customer A_ID: **10-0145.R2** Cadmus Art: **NAV21474** KGL ID: **JW-NAVT110078** — 2011/12/23 — page 3 — #3

Table 1. The x-values for $n = 2$, 4 and 8.

	$n = 2$	$n = 4$	$n = 8$
x_1	0	0	0
x_2	0.838729648038265	0.366954072987679	0.173052714641246
x_3		0.759464987433795	0.348716152257777
x_4		1.249085306682130	0.532617182616474
x_5			0.732041896003936
x_6			0.958268897313993
x_7			1.232161452950940
x_8			1.601867114624050

There is some round-off error in the calculations, but it is smaller than $2^{-52} = 2.220446049250313 \times 10^{-16}$, the 32-bit machine double precision, when A and a are reconstructed. For instance, when $n = 1024$, the largest difference among the 1024 pieces is $2.033963275582806 \times 10^{-16}$. Therefore, the x-values are acceptable on a 32-bit machine.

The algorithm described in the next section is fast because the appropriate rectangle for generating is chosen quickly because all majorizing rectangles and the majorizing tails have the same area a.

2.3. The Algorithm

The acceptance–rejection based rectangles algorithm is given below. Step 1 determines which rectangle is chosen or whether the tail is chosen and the sign of the random variate. Step 2 generates a variate from the tail using Marsaglia's tail algorithm. Step 3 generates a variate from the chosen rectangle. Step 4 takes advantage of the shaded rectangle in Fig. 1 to avoid exponential computations to increase the speed. Step 5 executes the exponential computation for (x, y) pairs that lie above one of the shaded rectangles in Fig. 1. Step 6 returns the variate with the appropriate sign.

1. Generate $u \sim U(0,1)$. If $u \geq 0.5$, set $t = 2u - 1$ and the sign to be positive, $s = 1$. If $u < 0.5$, set $t = 1 - 2u$ and the sign to be negative, $s = -1$. Let $h = t \cdot n$ and $p = \lceil h \rceil$ be the pth piece chosen.
2. If $p = n$, then use Marsaglia's tail algorithm to generate a random variate x from the tail. Generate $v \sim U(0,1)$ and $w \sim U(0,1)$. Set $x = \sqrt{d \cdot d - 2\log(w)}$ where $d = x_n$. If $v \cdot x \leq d$, then go to step 6. Otherwise, go to step 1.
3. If $p < n$, set $v = h - (p - 1)$ and generate $w \sim U(0,1)$. Set $x = x_p + v \cdot (x_{p+1} - x_p)$ and $y = w \cdot y_p$.
4. If $y \leq y_{p+1}$, then, avoid the exponential computation and go to step 6. Otherwise, go to step 5.
5. If $y \leq \exp(-x^2/2)$, then go to step 6. Otherwise, go to step 1.
6. Return $x = s \cdot x$.

A structured algorithm is given in the appendix in pseudocode. Indentation denotes nesting.

3. ALGORITHM ANALYSIS

Table 2 shows the properties of the majorizing function and the rectangles algorithm as n increases. Here, all of the entries are scaled back to $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(x-\mu)^2}{2\sigma^2})$ and $\mu = 0$, $\sigma = 1$. As n increases, the total area under the majorizing function, A, and the area under each piece, a, are both decreasing. Furthermore, x_n increases with n. When $n = 1024$, x_n is 3.31775403783444 compared to 0.838729648038265 when $n = 2$. Its significance is that

Table 2. Algorithm properties as a function of n.

n	$a/\sqrt{2\pi}$	$2A/\sqrt{2\pi}$	x_n	$P(\text{rej})$	$P(\text{exp})$	Time
2	0.33460	1.33842	0.83873	0.25285	0.14827	1.05394
4	0.14639	1.17115	1.24909	0.14614	0.16397	1.02716
8	0.06903	1.10461	1.60187	0.09470	0.13712	0.99259
16	0.03329	1.06542	1.91504	0.06140	0.10022	0.90214
32	0.01625	1.04024	2.19700	0.03878	0.06722	0.86362
64	0.00800	1.02420	2.45414	0.02363	0.04260	0.83635
128	0.00396	1.01426	2.69147	0.01406	0.02594	0.81881
256	0.00196	1.00825	2.91275	0.00819	0.01535	0.81374
512	0.00098	1.00471	3.12082	0.00468	0.00888	0.81016
1024	0.00048	1.00265	3.31775	0.00264	0.00505	0.80727
∞	0	1	∞	0	0	NA

J_ID: **z8m** Customer A_ID: **10-0145.R2** Cadmus Art: **NAV21474** KGL ID: **JW-NAVT110078** — 2011/12/23 — page 4 — #4

Table 3. Run times for 10,000,000 variates.

Name	Ziggurat	Rectangles	KR	OE	IBM	BM
CPU	0.44207	0.80727	1.07094	1.30894	1.76370	1.76675
Ziggurat Eff	1	1.82611	2.42256	2.96093	3.98964	3.99654
Rectangles Eff	0.54761	1	1.32662	1.62144	2.18477	2.18855

we generate fewer variates from the tail portion and have higher acceptance rate when we generate variates from the tail. Therefore, the probability of rejecting a variate, P(rej), is decreasing and the efficiency is increasing as n increases. Also, the probability that an exponential computation is executed, $P(\text{exp})$, decreases in n. When $n = 1024$, the probability is 0.00505 compared to 0.14827 when $n = 2$. The right-most column shows the average run time for generating 10,000,000 variates, in seconds. The algorithm is written in C, and was tested on a machine with 2 Opteron 2.6 GHz, dual-core processors, 20 GB RAM, UNIX system and compiled by `gcc` compiler without optimization options. In the C implementation, we take advantage of bit computing and use the least-significant bits of the integer random number generator to choose the rectangle number to speed up the algorithm. The run times in the table are averages of 10 runs. The speed increases with n. In addition to the run time being $(1.05394 - 0.080727)/1.05394 \cong 23\%$ faster for the $n = 1024$ case over the $n = 2$ case, there are also improved correlation properties in the $n = 1024$ case that are discussed subsequently.

We compared the rectangles algorithm's efficiency to some popular algorithms: the ziggurat method, the OE method, the BM transformation, the IBM transformation (IBM), and the KR method. We only show the case when $n = 1024$ because it has the best performance by dividing the majorizing function into 2048 pieces.

T3 Table 3 contains the run times for generating 10,000,000 variates by these six algorithms sorted from left to right by CPU time. The test environment is as described previously. The first row is the CPU time in seconds, the second row is the efficiency relative to the ziggurat method, and the third row is the efficiency relative to the proposed algorithm. The rectangles algorithm has the second shortest run time. Although it is 1.8 times slower than the ziggurat method which is probably the fastest known algorithm, the rectangles algorithm is much faster than other methods. Furthermore, we could have compared the proposed algorithm to the Ahrens and Dieter [1] and Leva [13] methods. Based on the research done by Marsaglia and Tsang [18], the Ahrens and Dieter method is about three times, and the Leva method is about six times slower than the ziggurat method. Therefore, it is safe to conclude that our algorithm is faster than these two methods. From the perspective of speed, the proposed algorithm is inferior to the ziggurat method because it generally needs to call one more uniform random number than the ziggurat method. Therefore, the relative performance between them will vary if we use different random number generators, see Marsaglia [15]. Here, we use the Lehmer random number generator proposed by Park and Miller [21], and described in Leemis and Park [12].

4. CORRELATION INDUCTION

To reduce the variance in a simulation experiment, a random variate generation algorithm that is monotone, or approximately monotone, can be used. Although random variates generated by the inverse transformation have the best correlation induction, inversion methods are oftentimes slower than other methods, particularly when numerical methods are required. The OE method is about 62% slower than the rectangles algorithm. Furthermore, the rectangles algorithm is able to obtain a very good correlation induction by making minor modifications suggested by Schmeiser and Kachitvichyanukul [22]. For monotonicity, the way t was computed in step 1 of the rectangles algorithm ensures

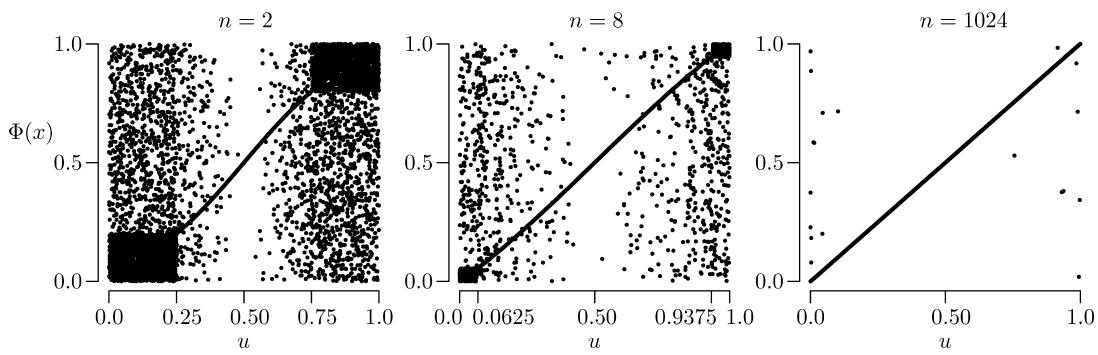

Figure 2. Correlation plots.

J_ID: **z8m** Customer A_ID: **10-0145.R2** Cadmus Art: **NAV21474** KGL ID: **JW-NAVT110078** — 2011/12/23 — page 5 — #5

Table 4. Correlation induction for the rectangles algorithm as n increases.

n	2	4	8	16	32	64	128	256	512	1024	∞
Correlation	0.667	0.802	0.868	0.910	0.942	0.962	0.977	0.986	0.992	0.996	1

Table 5. Run times of correlation induction rectangle algorithm.

Name	Ziggurat	Rectangles	RectanglesCI	KR	OE
CPU	0.44207	0.80727	0.87205	1.07094	1.30894
Ziggurat Eff	1	1.82611	1.97265	2.42256	2.96093
Rectangles Eff	0.54761	1	1.08025	1.32662	1.62144

monotonicity. For synchronization, we pass two random seeds to the rectangles algorithm for two random-number streams. In the first iteration, we generate uniform random numbers from stream 1. If we reject in the first iteration, we use stream 2 to generate uniform random variates until we generate a standard normal variate. The modified algorithm is as follows:

0. If this is the first iteration, use random number stream 1. Otherwise, use random number stream 2.
1. Generate $u \sim U(0,1)$. If $u \geq 0.5$, set $t = 2u - 1$ and the sign to be positive, $s = 1$. If $u < 0.5$, set $t = 1 - 2u$ and the sign to be negative, $s = -1$. Let $h = t \cdot n$ and $p = \lceil h \rceil$ be the pth piece chosen.
2. If $p = n$, then use Marsaglia's tail algorithm to generate a random variate x from the tail. Generate $v \sim U(0,1)$ and $w \sim U(0,1)$. Set $x = \sqrt{d \cdot d - 2\log(w)}$ where $d = x_n$. If $v \cdot x \leq d$, then go to step 6. Otherwise, go to step 0.
3. If $p < n$, set $v = h - (p-1)$ and generate $w \sim U(0,1)$. Set $x = x_p + v \cdot (x_{p+1} - x_p)$ and $y = w \cdot y_p$.
4. If $y \leq y_{p+1}$, then, avoid the exponential computation and go to step 6. Otherwise, go to step 5.
5. If $y \leq \exp(-x^2/2)$, then go to step 6. Otherwise, go to step 0.
6. Return $x = s \cdot x$.

F2 Figure 2 shows scatter plots of u and $\Phi(x) = \int_{-\infty}^{x} f(w)dw$ for one run with 10,000 variates when $n = 2$, $n = 8$, and $n = 1024$ where u is the uniform variate generated in the first iteration, $\Phi(x)$ is the CDF of the standard normal distribution and x is the standard normal random variate generated by the algorithm. The diagonal line from (0,0) to (1,1) contains the random variates that were accepted in a rectangle on the first iteration. The fraction of random variates that are accepted in a rectangle on the first iteration by the rectangles algorithm increases in n. The two rectangular-shaped clusters near (0,0) and (1,1) are caused by Marsaglia's tail algorithm. The rectangular-shaded clusters of dots decrease in size as n increases. The other dots are the random variates that were rejected in the first iteration. Furthermore, when u is close to 0.5, the probability of rejection is close to 0. This is apparent in Fig. 1 by the small ratio of the unshaded portion of the first rectangle to the area of the first rectangle.

Table 4 shows the average correlations between u and $\Phi(x)$ **T4** in 10 runs with 10,000 variates when n increases from 2 to ∞. The correlation increases with n and is approximately 0.996 when $n = 1024$.

Table 5 shows the run time of the rectangles algorithm **T5** with correlation induction (*RectanglesCI*) in the fourth column. The meaning of each row is the same as that in Table 3. The modified rectangles algorithm is about 8% slower than the original one. The main reason is that for maintaining the monotonicity, we cannot use the least-significant bits of the integer random number generator to choose the rectangle number. However, the modified rectangles algorithm is still faster than the OE, KR, and Ahrens and Dieter methods.

5. CONCLUSIONS

The rectangles algorithm for generating normal random variates has very high efficiency compared to other popular algorithms. With 2048 pieces for its majorizing function, the probability is 0.00264 that variates are rejected in the first iteration and is 0.00505 that an exponential computation is executed. The more important feature is that the rectangles algorithm has very high correlation induction, approximately 0.996, which could reduce the variance in simulation experiments and is faster than existing inversion methods and most other non-inversion methods. To summarize, the rectangles algorithm falls between the ziggurat and OE method having the speed advantage over the OE method and the correlation induction advantage over the ziggurat method.

APPENDIX

Let n be the number of equal-area pieces in the majorizing function and $x_1, x_2, x_3, \ldots, x_n$ be the interval boundaries. Let G be a flag such that $G = 1$ if the random variate is generated and $G = 0$ otherwise.

J_ID: **z8m** Customer A_ID: **10-0145.R2** Cadmus Art: **NAV21474** KGL ID: **JW-NAVT110078** — 2011/12/23 — page 6 — #6

Algorithm 1 Pseudocode for the Rectangles Algorithm

```
d ← x_n
G ← 0
while G ≠ 1 do
  generate u ~ U(0,1)
  if u ≥ 0.5 then
    t ← 2u − 1
    s ← 1
  else
    t ← 1 − 2u
    s ← −1
  end if
  h ← t · n
  p ← ⌈h⌉
  if p = n then
    generate v ~ U(0,1)
    generate w ~ U(0,1)
    x ← sqrt(d · d − 2 log(w))
    if v · x ≤ d then
      G ← 1
    end if
  else
    v ← h − (p − 1)
    generate w ~ U(0,1)
    x ← x_p + v · (x_{p+1} − x_p)
    y ← w · y_p
    if y ≤ y_{p+1} then
      G ← 1
    else if y ≤ exp(−x^2/2) then
      G ← 1
    end if
  end if
end while
return s · x
```

ACKNOWLEDGMENTS

The authors like to thank Bruce Schmeiser and Raghu Pasupathy for their helpful suggestions concerning this work. Also, the computational work was performed on the SciClone Cluster Project at The College of William & Mary.

REFERENCES

[1] J.H. Ahrens and U. Dieter, An alias method for sampling from the normal distribution, Computing 42 (1988), 159–170.
[2] G.E.P. Box and M.J. Muller, A note on the generation of random normal deviates, Ann Math Stat 29 (1958), 610–611.
[3] A.L. Brophy, Approximation of the inverse normal distribution function, Behav Res Methods 17 (1985), 415–417.
[4] P. Bratley, B.L. Fox, and L.E. Schrage, A guide to simulation, 2nd edition, Springer, New York, 1987.
[5] H.C. Chen and Y. Asau, On generating random variates from an empirical distribution, IIE Trans 6 (1974), 163–166.
[6] J.A. Doornik, An improved ziggurat method to generate normal random samples, Working Paper, Department of Economics, University of Oxford, 2005.
[7] N. Johnson, S. Kotz, and N. Balakrishnan, Continuous univariate distributions, 2nd ed., Volume 1, Wiley, New York, 1995.
[8] A.J. Kinderman and J.F. Monahan, Computer generation of random variables using the ratio of uniform deviates, ACM Trans Math Softw 3 (1977), 257–260.
[9] A.J. Kinderman and J.G. Ramage, Computer generation of normal random variables, J Am Stat Assoc 71 (1976), 893–896.
[10] D.E. Knuth, The art of computer programming, volume 2: Seminumerical algorithms, 3rd ed., Addison Wesley, Boston, 1998.
[11] A.M. Law, Simulation modeling and analysis, 4th ed., McGraw–Hill, Boston, 2007.
[12] L. Leemis and S. Park, Discrete-event simulation: A first course, Prentice–Hall, Upper Saddle River, New Jersey, 2006.
[13] J.L. Leva, A fast normal random number generator, ACM Trans Math Softw 18 (1992), 449–453.
[14] G. Marsaglia, Generating a variable from the tail of the normal distribution, Technometrics 6 (1964), 101–102.
[15] G. Marsaglia, Re: Good C random number generator, Posting, Usenet newsgroup sci.lang.c. 13-May-2003, 2003.
[16] G. Marsaglia and T.A. Bray, A convenient method for generating normal variables, SIAM Rev 6 (1964), 260–264.
[17] G. Marsaglia and W.W. Tsang, A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions, SIAM J Sci Stat Comput 5 (1984), 349–359.
[18] G. Marsaglia and W.W. Tsang, The ziggurat method for generating random variables, J Stat Softw 5 (2000), 1–7.
[19] B. Moro, The full Monte, Risk 8 (1995), 57–58.
[20] R.E. Odeh and J.O. Evans, The percentage points of the normal distribution, Appl Stat 23 (1974), 96–97.
[21] S. Park and K. Miller, Random number generators: Good ones are hard to find, Commun ACM 31 (1988), 1192–1201.
[22] B. Schmeiser and V. Kachitvichyanukul, Correlation induction without the inverse transformation, Proceedings the Winter Simulation Conference, Radisson Mark Plaza Hotel, Washington, DC, 1986, pp. 266–274.
[23] A.J. Walker, New fast method for generating discrete random numbers with arbitrary frequency distributions, Electron Lett 10 (1974), 127–128.
[24] A.J. Walker, Fast generation of uniformly distributed pseudo random numbers with floating point representation, Electron Lett 10 (1974), 553–554.
[25] A.J. Walker, An efficient method for generating discrete random variables with general distributions, ACM Trans Math Softw 3 (1977), 253–256.
[26] C.S. Wallace, Fast pseudorandom generators for normal and exponential variates, ACM Trans Math Softw 22 (1996), 119–127.