Algorithm 488

A Gaussian Pseudo-Random Number Generator [G5]

Richard P. Brent [Recd. 9 Nov. 1973, and 19 Dec. 1973]
Computer Centre, Australian National University,
Canberra, Australia

Key Words and Phrases: random numbers, pseudo-random numbers, Gaussian distribution, normal distribution
CR Categories: 5.39, 5.5
Language: Fortran

Description

Introduction. Successive calls to the Fortran function *GRAND* return independent, normally distributed pseudo-random numbers with zero mean and unit standard deviation. It is assumed that a Fortran function *RAND* is available to generate pseudo-random numbers which are independent and uniformly distributed on [0, 1). Thus, *GRAND* may be regarded as a function which converts uniformly distributed numbers to normally distributed numbers.

Outline of the method. GRAND is based on the following algorithm (Algorithm A) for sampling from a distribution with density function $f(x) = K \exp(-G(x))$ on $[a, b)$, where

$$0 \le G(x) \le 1 \qquad (1)$$

on $[a, b)$, and the function $G(x)$ is easy to compute:

```
Step 1. If the first call, then take a sample $u$ from the uniform distribution on $[0, 1)$; otherwise $u$ has been saved from a previous call.
Step 2. Set $x \leftarrow a + (b - a)u$ and $u_0 \leftarrow G(x)$.
Step 3. Take independent samples $u_1, u_2, \ldots$ from the uniform distribution on $[0, 1)$ until, for some $k \ge 1$, $u_{k-1} \le u_k$.
Step 4. Set $u \leftarrow (u_k - u_{k-1})/(1 - u_{k-1})$.
Step 5. If $k$ is even go to Step 2, otherwise return $x$.
```

The reason why Algorithm A is correct is explained in Ahrens and Dieter [2], Forsythe [4], and Von Neumann [6]. The only point which needs explanation here is that, at Step 4, we can form a new uniform variate u from u_{k-1} and u_k, thus avoiding an extra call to the uniform random number generator. This is permissible since at Step 4 it is clear (from Step 3) that $(u_k - u_{k-1})/(1 - u_{k-1})$ is distributed uniformly and independent of x and k. (The fact that it is dependent on u_k is irrelevant.)

Let a_i be defined by $(2/\pi)^{\frac{1}{2}} \int_{a_i}^{\infty} \exp(-\frac{1}{2}t^2)dt = 2^{-i}$ for $i = 0, 1, \ldots$. To sample from the positive normal distribution (Algorithm B), we may choose $i \ge 1$ with probability 2^{-i} (easily done by inspecting the leading bits in a uniformly distributed number) and then use Algorithm A to generate a sample from $[a_{i-1}, a_i)$, with $G(x) = \frac{1}{2}(x^2 - a_{i-1}^2)$. It is easy to verify that condition (1) is satisfied, in fact

$$\tfrac{1}{2}(a_i^2 - a_{i-1}^2) < \log(2). \qquad (2)$$

Finally, to sample from the normal distribution (Algorithm C), we generate a sample from the positive normal distribution and then attach a random sign.

Comments on the method. The algorithm is exact, apart from the inevitable effect of computing with floating-point numbers with a finite word-length. Thus, the method is preferable to methods which depend on the central limit theorem or use approximations to the inverse distribution function.

Let N be the expected number of calls to a uniform random number generator when Algorithm A is executed. If the expected value of k at Step 3 is E, and the probability that k is even is P, then $N = E + N P$, so $N = E/(1 - P)$. From Forsythe [4, eq. (11)], $E = (b - a)^{-1} \int_a^b \exp(G(x))dx$ and

$$1 - P = \frac{1}{b - a}\int_a^b \exp(-G(x))\,dx, \quad \text{so}$$

$$N = \int_a^b \exp\ (G(x))\,dx \Big/ \int_a^b \exp(-G(x))\,dx. \qquad (3)$$

From (3) and the choice of a_i, the expected number of calls to a uniform random number generator when Algorithm C is executed is

$$\sum_{i=1}^{\infty} 2^{-i} \int_{a_{i-1}}^{a_i} \exp(\tfrac{1}{2}(x^2 - a_{i-1}^2))\,dx \Big/ \int_{a_{i-1}}^{a_i} \exp(-\tfrac{1}{2}(x^2 - a_{i-1}^2))\,dx \simeq 1.37446. \qquad (4)$$

This is lower than 4.03585 for the algorithm given in Forsythe [4], or 2.53947 for the improved version (*FT*) given in Ahrens and Dieter [2]. It is even slightly lower than 1.38009 for the algorithm FL_4 of [2], and FL_4 requires a larger table than Algorithm C. Thus, Algorithm C should be quite fast, and comparable to the best algorithms described by Ahrens and Dieter [1]. The number (4) could be reduced by increasing the table size (as in the algorithms FL_4, FL_5, and FL_6 of [2]), but this hardly seems worthwhile. Exact timing comparisons depend on the machine and uniform random number generator used. (If a very fast uniform generator is used, then Step 4 of Algorithm A may take longer than generating a new uniform deviate.)

The loss of accuracy caused by Step 4 of Algorithm A is not serious. We may say that $\log_2 (1 - u_{k-1})^{-1}$ "bits of accuracy" are lost, and in our application we have, from (2) and Step 3 of Algorithm A, $\log(2) > u_0 > \cdots > u_{k-1}$, so the number of bits lost is less than $\log_2 (1 - \log(2))^{-1} < 2$.

Test results. If x is normally distributed then $u = (2\pi)^{-\frac{1}{2}} \int_{-\infty}^{x} \exp(-\frac{1}{2}t^2)\,dt$ is uniformly distributed on (0, 1). Hence, standard tests for uniformity may be applied to the transformed variate u. Several statistical tests were performed, using a Univac 1108 with both single-precision (27-bit fraction) and double-precision (60-bit fraction). For example, we tested two-dimensional uniformity by taking 10^6 pairs (u, u'), plotting them in the unit square, and performing the Chi-squared test on the observed numbers falling within each of 100 by 100 smaller squares. This test should show up any lack of independence in pairs of successive uniform deviates. We tested one-dimensional uniformity similarly, taking 10^6 trials and subdividing (0, 1) into 1,000 smaller intervals. The values of χ^2 obtained were not significant at the 5 percent level. It is worth noting that the method of summing 12 numbers distributed uniformly on $(-1/2, 1/2)$ failed the latter test, giving $\chi^2_{999} = 1351$. (The probability of such a value being exceeded by chance is less than 10^{-11}.)

Naturally, test results depend on the particular uniform generator *RAND* which is used. *GRAND* will not produce independent normally distributed deviates unless *RAND* supplies it with independent uniformly distributed deviates! For our tests we used an additive uniform generator of the form $u_n = u_{n-1} + u_{n-127} \pmod{2^w}$ with $w = 27$ or 60 (see Brent [3] and Knuth [5]), but a good linear congruential generator should also be adequate for most applications.

Comparison with Algorithm 334. The fastest exact method previously published in Communications is Algorithm 334 [7]. We timed function *GRAND*, subroutine *NORM* (a Fortran translation of Algorithm 334), and function *RAND* (the uniform random number generator called by *GRAND* and *NORM*). The mean execution times obtained from 500,000 trials on a Univac 1108 were 172, 376 and 59 μsec respectively. Since *NORM* returns two normally distributed numbers, *GRAND* was effectively 9 percent faster than *NORM*. Based on comparisons in [2], we estimate that the saving would be greater if both routines were coded in assembly language, for much of the execution time of *NORM* is taken up in evaluating a square-root and logarithm which are already coded in assembly language.

GRAND requires about 1.38 uniform deviates per normal deviate, and *NORM* requires $4/\pi + 1/2 \simeq 1.77$. Thus, we may estimate that if a uniform generator taking U μsec per call were used, the time per normal deviate would be $(91 + 1.38U)$ μsec for *GRAND* and $(83 + 1.77U)$ μsec for *NORM*. Hence, *GRAND* should be faster for $U \geq 20$.

References

1. Ahrens, J.H., and Dieter, U. Computer methods for sampling from the exponential and normal distributions. *Comm. ACM 15*, 10 (Oct. 1972), 873–882.
2. Ahrens, J.H., and Dieter, U. Pseudo-random Numbers (preliminary version). Preprint of book to be published by Springer, Part 2, Chs. 6–8.
3. Brent, R.P. *Algorithms for Minimization Without Derivatives.* Prentice-Hall, Englewood Cliffs, N.J., 1973, pp. 163–164.
4. Forsythe, G.E. Von Neumann's comparison method for random sampling from the normal and other distributions. *Math. Comp. 26*, 120 (Oct. 1972), 817–826.
5. Knuth, D.E. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, Reading, Mass., 1969, pp. 26, 34, 464.
6. Von Neumann, J. Various techniques used in connection with random digits. In *Collected Works, Vol. 5*, Pergamon Press, New York, 1963, pp. 768–770.
7. Bell, J.R. Algorithm 334, Normal random deviates. *Comm. ACM 11*, 7 (July 1968), 498.

Algorithm

```
      FUNCTION GRAND(N)
C EXCEPT ON THE FIRST CALL GRAND RETURNS A
C PSEUDO-RANDOM NUMBER HAVING A GAUSSIAN (I.E.
C NORMAL) DISTRIBUTION WITH ZERO MEAN AND UNIT
C STANDARD DEVIATION.  THUS, THE DENSITY IS  F(X) =
C EXP(-Ø.5*X**2)/SQRT(2.Ø*PI). THE FIRST CALL
C INITIALIZES GRAND AND RETURNS ZERO.
C THE PARAMETER N IS DUMMY.
C GRAND CALLS A FUNCTION RAND, AND IT IS ASSUMED THAT
C SUCCESSIVE CALLS TO RAND(Ø) GIVE INDEPENDENT
C PSEUDO- RANDOM NUMBERS DISTRIBUTED UNIFORMLY ON (Ø,
C 1), POSSIBLY INCLUDING Ø (BUT NOT 1).
C THE METHOD USED WAS SUGGESTED BY VON NEUMANN, AND
C IMPROVED BY FORSYTHE, AHRENS, DIETER AND BRENT.
C ON THE AVERAGE THERE ARE 1.37746 CALLS OF RAND FOR
C EACH CALL OF GRAND.
C WARNING - DIMENSION AND DATA STATEMENTS BELOW ARE
C           MACHINE-DEPENDENT.
C DIMENSION OF D MUST BE AT LEAST THE NUMBER OF BITS
C IN THE FRACTION OF A FLOATING-POINT NUMBER.
C THUS, ON MOST MACHINES THE DATA STATEMENT BELOW
C CAN BE TRUNCATED.
C IF THE INTEGRAL OF SQRT(2.Ø/PI)*EXP(-Ø.5*X**2) FROM
C A(I) TO INFINITY IS 2**(-I), THEN D(I) = A(I) -
C A(I-1).
      DIMENSION D(6Ø)
      DATA D(1), D(2), D(3), D(4), D(5), D(6), D(7),
     * D(8), D(9), D(1Ø), D(11), D(12), D(13),
     * D(14), D(15), D(16), D(17), D(18), D(19),
     * D(2Ø), D(21), D(22), D(23), D(24), D(25),
     * D(26), D(27), D(28), D(29), D(3Ø), D(31),
     * D(32) /Ø.67448975Ø,Ø.47585963Ø,Ø.383771164,
     * Ø.328611323,Ø.291142827,Ø.263684322,
     * Ø.2425Ø8452,Ø.225567444,Ø.211634166,
     * Ø.199924267,Ø.18991Ø758,Ø.181225181,
     * Ø.1736Ø14ØØ,Ø.1668419Ø9,Ø.16Ø796729,
     * Ø.155349717,Ø.15Ø4Ø9384,Ø.1459Ø2577,
     * Ø.14177ØØ33,Ø.137963174,Ø.134441762,
     * Ø.13117215Ø,Ø.128125965,Ø.12527909Ø,
     * Ø.12261Ø883,Ø.12Ø1Ø356Ø,Ø.1177417Ø7,
     * Ø.115511892,Ø.1134Ø2349,Ø.1114Ø272Ø,
     * Ø.1Ø95Ø3852,Ø.1Ø7697617/
      DATA D(33), D(34), D(35), D(36), D(37), D(38),
     * D(39), D(4Ø), D(41), D(42), D(43), D(44),
     * D(45), D(46), D(47), D(48), D(49), D(5Ø),
     * D(51), D(52), D(53), D(54), D(55), D(56),
     * D(57), D(58), D(59), D(6Ø)
     * /Ø.1Ø5976772,Ø.1Ø4334841,Ø.1Ø2766Ø12,
     * Ø.1Ø1265Ø52,Ø.Ø99827234,Ø.Ø98448282,
     * Ø.Ø971243Ø9,Ø.Ø95851778,Ø.Ø94627461,
     * Ø.Ø93448407,Ø.Ø923119Ø9,Ø.Ø91215482,
     * Ø.Ø9Ø156838,Ø.Ø89133867,Ø.Ø88144619,
     * Ø.Ø87187293,Ø.Ø8626Ø215,Ø.Ø85361834,
     * Ø.Ø8449Ø7Ø6,Ø.Ø83645487,Ø.Ø82824924,
     * Ø.Ø82Ø27847,Ø.Ø81253162,Ø.Ø8Ø499844,
     * Ø.Ø79766932,Ø.Ø79Ø53527,Ø.Ø78358781,
     * Ø.Ø77681899/
C END OF MACHINE-DEPENDENT STATEMENTS
C U MUST BE PRESERVED BETWEEN CALLS.
      DATA U /Ø.Ø/
C INITIALIZE DISPLACEMENT A AND COUNTER I.
      A = Ø.Ø
      I = Ø
C INCREMENT COUNTER AND DISPLACEMENT IF LEADING BIT
C OF U IS ONE.
   1Ø U = U + U
      IF (U.LT.1.Ø) GO TO 2Ø
      U = U - 1.Ø
      I = I + 1
      A = A - D(I)
      GO TO 1Ø
C FORM W UNIFORM ON Ø .LE. W .LT. D(I+1) FROM U.
   2Ø W = D(I+1)*U
C FORM V = Ø.5*((W-A)**2 - A**2). NOTE THAT Ø .LE. V
C .LT. LOG(2).
      V = W*(Ø.5*W-A)
C GENERATE NEW UNIFORM U.
   3Ø U = RAND(Ø)
C ACCEPT W AS A RANDOM SAMPLE IF V .LE. U.
      IF (V.LE.U) GO TO 4Ø
C GENERATE RANDOM V.
      V = RAND(Ø)
C LOOP IF U .GT. V.
      IF (U.GT.V) GO TO 3Ø
C REJECT W AND FORM A NEW UNIFORM U FROM V AND U.
      U = (V-U)/(1.Ø-U)
      GO TO 2Ø
C FORM NEW U (TO BE USED ON NEXT CALL) FROM U AND V.
   4Ø U = (U-V)/(1.Ø-V)
C USE FIRST BIT OF U FOR SIGN, RETURN NORMAL VARIATE.
      U = U + U
      IF (U.LT.1.Ø) GO TO 5Ø
      U = U - 1.Ø
      GRAND = W - A
      RETURN
   5Ø GRAND = A - W
      RETURN
      END
```