A More Portable Fortran Random Number Generator

LINUS SCHRAGE
University of Chicago

A Fortran implementation of a random number generator is described which produces a sequence of random integers that is machine independent as long as the machine can represent all integers in the interval [−2**31 + 1, 2**31 − 1]

Key Words and Phrases uniform random number generator, portable software
CR Categories 4.6, 4.9, 5.5

INTRODUCTION

There are a number of situations in which it is desirable to have a random number generator that is machine independent. In general, it is useful if a program written in a high-level language produces results which are the same from machine to machine as long as the input to the program is the same.

For example, the pseudorandom number generator used by the Control Data Corporation in its GPSS simulation program is the same as in IBM's GPSS, even though the generator is known to have defective statistical behavior. Apparently, compatibility is more valuable than statistical goodness.

The program described here is an implementation of the generator described by Lewis et al. [6] and indirectly attributed to D.H. Lehmer. The code for the program appears in Figure 1. The generator produces a sequence of positive integers, IX, by the recursion:

$$IX(i + 1) = A*IX(i) \bmod P$$

where P is the Mersenne prime number $2^{31} - 1 = 2147483647$ and $A = 7^5 = 16807$. Thus all integers IX produced will satisfy $0 < IX < 2^{31} - 1$. Most large computers can represent all integers in this range using INTEGER format. Many minicomputers, however, are fundamentally 16-bit machines and thus cannot represent integers in this range with the "native" INTEGER format. If, however, the minicomputer has another format, such as DOUBLE PRECISION which can represent integers in the range $[0, 2^{31} - 1]$, then the generator proposed here may

Author's address: Graduate School of Business, University of Chicago, 5836 Greenwood Ave , Chicago, IL 60637

```
      FUNCTION RAND(IX)
C  PORTABLE RANDOM NUMBER GENERATOR
C   USING THE RECURSION
C     IX - IX*A MOD P
C
C  SOME COMPILERS, E.G., THE HP3000, REQUIRE THE FOLLOWING
C   DECLARATION TO BE INTEGER*4
      INTEGER A,P,IX,B15,B16,XHI,XALO,LEFTLO,FHI,K
C
C  /**5, 2**15, 2**16, 2**31-1
      DATA A/16807/,B15/32768/,B16/65536/,P/2147483647/
C
C  GET 15 HI ORDER BITS OF IX
      XHI = IX/B16
C  GET 16 LO BITS OF IX AND FORM LO PRODUCT
      XALO=(IX XHI*B16)*A
C  GET 15 HI ORDER BITS OF LO PRODUCT
      LEFTLO = XALO/B16
C  FORM THE 31 HIGHEST BITS OF FULL PRODUCT
      FHI = XHI*A + LEFTLO
C  GET OVERFLO PAST 31ST BIT OF FULL PRODUCT
      K = FHI/B15
C  ASSEMBLE ALL THE PARTS AND PRESUBTRACT P
C   THE PARENTHESES ARE ESSENTIAL
      IX = (((XALO-LEFTLO*B16) - P) + (FHI-K*B15)*B16) + K
C  ADD P BACK IN IF NECESSARY
      IF (IX .LT. 0) IX = IX + P
C  MULTIPLY BY 1/(2**31-1)
      RAND = FLOAT(IX)*4.656612875E-10
      RETURN
      END
```

Fig 1. Portable Fortran random number generator

still be implemented. A version of RAND which uses DOUBLE PRECISION rather than the INTEGER format appears in Figure 2.

MACHINE INDEPENDENT USE OF THE GENERATOR

The generator is invoked as a function, that is, FX = RAND (IX). Before first use, IX must be set to some initial integral value in the range $0 < IX < 2147483647$. Subsequent values of IX are then obtained recursively from the preceding IX by invoking the function. RAND returns a random fraction in the interval $0 < RAND < 1$. One can use either this fractional value or IX itself.

Even though the generator is machine independent according to the earlier criterion, it is quite easy to use it in a way which destroys this independence.

To maintain as much machine independence as possible one should use IX rather than RAND where possible. For example, to generate a random integer K, in [I, J] one could use either:

(1) K = (J − I + 1)*RAND(IX) + I

or

(2) FX = RAND (IX)
K = (IX/(2147483647/(J − I + 1))) + I.

Method (2) is preferred because it is machine independent. The value of K in

```
      DOUBLE PRECISION FUNCTION DRAND(IX)
C  PORTABLE RANDOM NUMBER GENERATOR
C   USING THE RECURSION
C    IX = IX*A MOD P
C
      DOUBLE PRECISION A,P,IX,B15,B16,XHI,XALO,LEFTLO,FHI,K
C
C  /**5, 2**15, 2**16, 2**31-1
      DATA A/16807.D0/,B15/32768.D0/,B16/65536.D0/,P/2147483647.D0/
C
C  GET 15 HT ORDER BITS OF IX
      XHI = IX/B16
      XHI = XHI - DMOD(XHI,1.D0)
C  GET 16 LO BITS OF IX AND FORM LO PRODUCT
      XALO=(IX-XHI*B16)*A
C  GET 15 HI ORDER BITS OF LO PRODUCT
      LEFTLO = XALO/B16
      LEFTLO = LEFTLO - DMOD(LEFTLO,1.D0)
C  FORM THE 31 HIGHEST BITS OF FULL PRODUCT
      FHI = XHI*A + LEFTLO
C  GET OVERFLO PAST 31ST BIT OF FULL PRODUCT
      K = FHI/B15
      K = K - DMOD(K,1.D0)
C  ASSEMBLE ALL THE PARTS AND PRESUBTRACT P
C   THE PARENTHESES ARE ESSENTIAL
      IX = (((XALO-LEFTLO*B16) - P) + (FHI-K*B15)*B16) + K
C  ADD P BACK IN IF NECESSARY
      IF (IX .LT. 0.D0) IX = IX + P
C  MULTIPLY BY 1/(2**31-1)
      DRAND = IX*4.656612875D-10
      RETURN
      END
```

Fig 2 Portable Fortran random number generator using double precision arithmetic

method (1) may depend upon the manner in which the host machine converts integers to real.

For example, the widely used NETGEN portable problem generator [4] for network LP problems contains a slow but very portable random number generator. Unfortunately, method (1) is used when the generator is invoked. Thus, even though NETGEN takes integer input and produces integer output, it is not truly portable because it unnecessarily uses floating point arithmetic in intermediate calculations.

STATISTICAL PROPERTIES

As with all multiplicative congruential generators the output from the generator cycles. The generator is full cycle, that is, every integer from 1 to $2^{31} - 2 = 2147483646$ is generated exactly once in the cycle. This cycle length is about four times greater than that of other typical "portable" Fortran random number generators.

Lewis et al. [6] have subjected the IBM 360 machine language version of this generator to a battery of statistical tests. In their words the "generator has been found to be highly satisfactory." Hutchinson [3] points out that the low-order bits do not behave in the highly nonrandom fashion characteristic of generators

for which P is a power of 2. Gavish and Merchant [2] find this generator to be the best of the simple multiplicative generators they tested, including the generator with A = 630360016.

The IMSL library [8] uses this generator as the basis for its "shuffling" generator, GGU4, which has statistical properties which are empirically gratifying.

IMPLEMENTATION

There are two difficulties in implementing the generator in a high-level language:

(1) The product A*IX may have 46 bits in it, but we are only assuming that the host machine can correctly store and calculate 31-bit products.

(2) P is not a power of 2, in fact it is prime, so discarding high-order digits of A*IX is not a valid direct way of doing the mod P operation.

Difficulty (1) is resolved by the standard device of simulating double precision with software. The 31-bit integer IX can be written as $\alpha*2^{16} + \beta$ where α is a 15-bit integer and β is a 16-bit integer. The product A*IX can be written as $(A*\alpha)*2^{16} + A*\beta$. Because A (=16807) is represented by 15 bits, the product $A*\alpha$ has at most 30 bits and the product $A*\beta$ has at most 31 bits. Only these products are manipulated by the generator.

Difficulty (2) is resolved by using a procedure described by Payne et al. [7] and Fishman [1] and also attributed to D.H. Lehmer.

Suppose we wish to compute

$$\mathrm{IX}(i + 1) = \mathrm{A}*\mathrm{IX}(i) \bmod \mathrm{P} \tag{1}$$

where

$$\mathrm{P} = r^d - 1 \tag{2}$$

but that it is easier to compute

$$\mathrm{Z} = \mathrm{A}*\mathrm{IX}(i) \bmod r^d \tag{3}$$

where r is the radix of the host machine, e.g. 2, and d is the number of base-r digits of the host machine, e.g. 31. Equation (3) for an appropriate k is equivalent to

$$\mathrm{Z} = \mathrm{A}*\mathrm{IX}(i) - k*r^d. \tag{4}$$

If we add k to both sides of eq. (4), we obtain

$$\mathrm{Z} + k = \mathrm{A}*\mathrm{IX}(i) - k*(r^d - 1) = \mathrm{A}*\mathrm{IX}(i) - k*\mathrm{P}. \tag{5}$$

Thus, provided $\mathrm{Z} + k < \mathrm{P}$, $\mathrm{IX}(i + 1) = \mathrm{Z} + k$. In the Appendix it is shown that $\mathrm{Z} + k$ is always less than $2*P$, provided reasonable assumptions are put on r, d, P, and A; so if $\mathrm{Z} + k > \mathrm{P}$, we use $\mathrm{Z} + k - \mathrm{P}$ as $\mathrm{IX}(i + 1)$.

The appropriate value of k is

$$k = [\mathrm{A}*\mathrm{IX}(i)/r^d] \quad \text{(i.e. integer part)}. \tag{6}$$

Effectively, k is the set of overflow digits in the product, treated as a number.

The Fortran code uses $r^d = 2^{31}$. The code makes no assumptions about how the host computer does arithmetic other than that it does it correctly on all integers in $[-21^{31} + 1, 2^{31} - 1]$ and that it does truncation on integer division.

Table I. Time in Seconds for 100,000 Calls

Machine	RAND	Subroutine GGUBF	Machine language	DRAND
DEC 2050	6.3	9.1	2.5	25.4
IBM 168	2.94	2.43	.92	5.54
TI 59	570000*			

* This number is an extrapolation of results kindly supplied by W J. Cody. The TI 59 is a 13-decimal digit pocket calculator.

The reader may check the correctness of the implementation on his or her own computer by verifying that if IX(0) = 1, then IX(1000) = 522329230.

COMPARISONS WITH OTHER GENERATORS AND EXECUTION TIME

Kruskal [5] describes an "extremely portable random number generator"; however, its cycle length is only 2048 and is thus inappropriate for situations requiring large numbers of random numbers.

The IMSL scientific subroutine library [8] uses the same recursive formula as that described here in its subroutine GGUBF. The widely used SIMSCRIPT II.5 simulation system and the DEC 20 Fortran system use the same recursion but with the multiplier A = 630360016. The multiplier A = 16807 appears to have been more thoroughly tested by Lewis et al. [6].

The IMSL generator is written in Fortran and is fairly portable. It uses double precision, floating point arithmetic; however, some machines such as the HP3000/Series I can only represent at most 12 digits accurately in double precision, whereas the product of 16807 × 2147483646 has 14 digits. Thus GGUBF is not portable to the HP3000/Series I. RAND with the INTEGER*4 declaration as suggested in the program comments produces exactly the same sequence on the IBM 370, DEC 20, and HP3000.

The generator DRAND (Figure 2) is probably the most portable. It is appropriate for a computer such as the PRIME which carries less than 46 bits in DOUBLE PRECISION (so GGUBF cannot be used) and less than 31 bits in INTEGER (so RAND cannot be used) but does carry more than 31 bits in DOUBLE PRECISION (so DRAND works). DRAND has the drawback of making extensive use of DOUBLE PRECISION and thus being very slow.

For execution timings a Fortran program was written which called the random number generator 100,000 times and summed the numbers. The results appear in Table I. Supercomputer class machines such as the IBM 168 tend to have very efficient double precision, floating point algorithms, so GGUBF may execute slightly faster on such machines. Smaller machines may in fact resort to software to perform double precision arithmetic, so RAND may be considerably faster on them.

APPENDIX

General Version of the Method

The computing method for IX(i + 1) is a specific form of a more general method. Suppose we wish to compute

$$IX(i + 1) = A*IX(i) \bmod P \tag{A1}$$

but that it is easier to compute

$$Z = A*IX(i) \bmod E \tag{A2}$$

which would be true, for example, if $E = r^d$. Let $E > P$ and define g and k as follows:

$$\begin{aligned} g &= E - P > 0, \\ k &= [A*IX(i)/E] \quad \text{(i.e. the integer part)}, \end{aligned} \tag{A4}$$

so that k equals the number of integral multiples of E in $A*IX(i)$. Then, adding $k*g$ to both sides of eq. (A2) gives

$$Z + k*g = A*IX(i) - k*(E - g) = A*IX(i) - k*P. \tag{A5}$$

The general method is then: If eq. (A5) is less than P, then $IX(i + 1) = Z + k*g$. If not, then we wish to show that $IX(i + 1) = Z + k*g - P$.

This requires us to show that eq. (A5) is less than $2*P$. We note from eq. (A2) that $Z \leq E - 1$ so that for eq. (A5) to be less than $2*P$ it is sufficient to have

$$E - 1 + k*g < 2*P. \tag{A6}$$

Because $IX(i) < P$, from eq. (A4) we find that a sufficient condition for eq. (A6) is

$$E - 1 + (A*P/E)*g < 2*P, \tag{A7}$$

or also

$$P + g - 1 + (A*P/E)*g < 2*P, \tag{A8}$$

or also

$$(A*P/E) < \frac{P - g + 1}{g}. \tag{A9}$$

We finally obtain that a sufficient condition for the general method to work is

$$A < \frac{E*(P - g + 1)}{g*P} = E\left(\frac{1}{g} - \frac{1}{P} + \frac{1}{g*P}\right). \tag{A10}$$

For example, in RAND where $g = 1$ the sufficient condition is trivial: $A < E = 2^{31}$.

ACKNOWLEDGMENT

The presentation and arguments have benefitted substantially from the comments of Ed Battiste and Peter Lewis.

REFERENCES

1 FISHMAN, G.S. *Concepts and Methods in Discrete Event Digital Simulation.* Wiley, New York, 1973

2. GAVISH, B., AND MERCHANT, D.K. Binary level testing of pseudo random number generators. Tech. Rep., Graduate School of Management, U. of Rochester, 1978.
3. HUTCHINSON, D.W. A new uniform pseudorandom number generator. *Comm. ACM 9,* 6 (June 1966), 432–433.
4. KLINGMAN, D., NAPIER, A., AND STUTZ, J. NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost network problems. *Management Sci. 20,* 5 (Jan. 1974), 814–821.
5. KRUSKAL, J.B. Extremely portable random number generator. *Comm. ACM 12,* 2 (Feb. 1969), 93–94.
6 LEWIS, P A W , GOODMAN, A.S., AND MILLER, J M A pseudo-random number generator for the system/360. *IBM Syst. J. 8,* 2 (1969), 136–146.
7 PAYNE, W.H., RABUNG, J.R , AND BOGYO, T.P. Coding the Lehmer pseudo-random number generator *Comm ACM 12,* 2 (Feb. 1969), 85–86.
8. *The IMSL Library, Vol 1,* 6th ed., Int. Math. Stat. Libraries, Inc., Houston, Tex., July 1977.

Received January 1978, revised June 1978