UNITED STATES
DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY

# MACHINE-INDEPENDENT FORTRAN CODING OF LEHMER RANDOM NUMBER GENERATORS

By W. Kirby

Open-File Report 80-004

Reston, Virginia

1979

## Contents

# Machine-Independent Fortran Coding of Lehmer Random Number Generators

W. Kirby

## Abstract

Random number generators of the Lehmer multiplicative congruential type are easily coded in Fortran using double precision modular arithmetic. The technique rests jointly on double precision Fortran processing small whole numbers without any loss of precision and on the generator's not producing any numbers too large to be processed exactly. The double precision code has been used to implement two 35-bit generators and a 47-bit generator on the 32-bit IBM System/360 and to implement several System/360 generators on other machines. Time trials suggest that the Fortran-language generators yield machine independence with only modest increases in total simulation run times.

Keywords: Quasirandom numbers, Random numbers, Multiplicative congruential scheme, Lehmer scheme, Simulation, Monte Carlo method.

## Introduction

In practical simulation work one sometimes needs to replicate on one computer a simulation experiment or random number sequence originally run on a different computer. This need may arise, for example, during the installation of a simulation program developed for some other computer. The problem arose for us, on the other hand, when we wanted to test some assertions about a 35-bit random number generator with a 32-bit IBM System/360[1/] computer. Finally, a user having access to more than one computer may find it desirable, at least for small tests and demonstrations, to have a single random number source of proven reliability which can be run without modification on any of the available machines.

These problems are particularly troublesome because the need for the "foreign" random number generator typically is urgent, short-lived, and small-scaled. There is neither the time and skilled manpower nor the anticipated volume of use necessary for developing the kind of machine-language or tricky Fortran code typically used for random number generators.

Although the random number routines provided on most computers are not readily portable, most of them are machine independent in concept. Of these routines, many are Lehmer multiplicative congruential (or power residue) generators of the form

$$x_{n+1} \equiv kx_n \pmod{m}$$

in which the $x_n$ are a sequence of quasirandom integers, $k$ is a constant integer multiplier, and $x_{n+1}$ , the next quasirandom integer in the sequence, is the remainder left after dividing $kx_n$ by the modulus $m$. Uniform (0,1) numbers are obtained by dividing the $x_n$ by $m$. Different generators

[1/] The use of brand names in this report is for identification only and does not imply endorsement by the U.S. Geological Survey.

are obtained by different choices of modulus and multiplier. These choices usually are dictated by computational efficiency and the quasirandom behavior of the generated numbers. This paper does not consider these important choices; it deals only with the expedient implementation of existing generators in a language likely to be available to the simulation practitioner.

Because it is a strictly mathematical procedure, the multiplicative congruential scheme can be coded in Fortran--subject, of course, to the limitations of the available computer's version of the language. Machine-specific Fortran codes for several such generators are available (Nance and Overstreet, 1974, and IBM Corp., 1970, for example). Moreover, the use of Fortran to obtain machine-independent generators is not new--see, for example, Ahrens and others (1970), McGrath and Irving (1973), Fellows (1976), or Schrage (1979). Nonetheless, the usefulness of Fortran for implementing many existing multiplicative congruential generators seems not to be generally appreciated.

## Method

Two related conditions must be satisified if the multiplicative congruential scheme is to be implemented in Fortran (or any other mathematical language).

1) The values of all the integers $x_n$, $k$, $m$, and all the intermediate products $kx_n$ must be exactly representable by the Fortran data types being used.
2) Arithmetic operations on these data types, for integer values up to the maximum possible intermediate product, must be performed without any error.

A simple necessary although insufficient condition is that $k(m-1)$, the maximum possible intermediate product, not exceed $2^s-1$, where $s$ is the number of significant bits provided in the data type and preserved in the arithmetic operations. For example, the Fortran integer data type in the IBM System/360 has 31 significant bits, plus one sign bit, and can accomodate numbers up to $2^{31}-1$. The Fortran double precision data type, on the other hand, has 56 significant bits (in the floating point mantissa), plus 8 bits of sign and characteristic information, and can represent integral values as large as $2^{56}-1$ (about $7.2 \times 10^{16}$) without any loss of significant digits. Straightforward trials confirm that double precision arithmetic is exact for integral data of this magnitude. Although the Fortran integer data type often has insufficient capacity, the Fortran double precision arithmetic provided on a variety of machines does satisfy all of these conditions for several published generators, including those of IBM (1970), Lewis, et al. (1969), and Neave (1973).

The proposed machine-independent Fortran coding of the Lehmer generator is as follows:

```
DOUBLE PRECISION DI, DK, DM
.
.
.                                                    (1)
DI = DMOD(DI*DK, DM)
URAND = SNGL(DI)/SNGL(DM)
```

in which DI is the current random integer, DK the constant multiplier, and DM the modulus, all expressed as Fortran double precision variables.with integral values. URAND is the current unit uniform variate. Values appropriate to the generator being implemented must be supplied for DM and DK, and an appropriate initial value must be specified for DI.

In addition, this technique has been extended to generators such as those of Ahrens and Dieter (1972) and Payne and others (1969), which have maximum intermediate products too big to be represented exactly in the 56 significant bits of the IBM System/360 double precision word. The intermediate products are kept within the limits of exact double precision arithmetic by expressing the multiplier as $k = pq + a$ and by developing the intermediate products in four steps, $x_n k \equiv \{[x_n p (mod\ m)]\ q + x_n a\}\ (mod\ m)$, expressed as follows in Fortran:

```
      DI = DMOD (DMOD(DI*DP,DM)*DQ + DI*DA, DM)                    (2)
```

in which all variables must be declared double precision, and must be assigned integral values such that all steps can be carried out exactly. Specifically, DM*DP and DM*(DQ + DA) must not exceed the limits for exact double precision arithmetic (about $7.2 \times 10^{16}$ on the IBM System/360).

Finally, the same idea can be carried several steps farther by expressing the multiplier $k$ as a product of several factors plus an addend. In this way it was possible to implement a 47-bit generator with multiplier $5^{15}$ (or $125^5$) long used at Oak Ridge (McGrath and Irving 1973), in double precision Fortran on the IBM System/360 as follows:

```
      DI = DMOD(DMOD(DMOD(DMOD(DMOD(DI*DP,DM)
              *DP,DM)*DP,DM)*DP,DM)*DP,DM)                         (3)
```

in which DM = $2^{47}$ and DP = 125. This coding is not only far simpler than the multiple-precision integer arithmetic used by McGrath (1973), but also is practically as portable and fully twice as fast.

## Verification and efficiency

The proposed double precision Fortran coding has been used to implement several Lehmer-type generators which use a variety of machine-specific tricks to do the modular arithmetic. The characteristics of some of these generators and short sequences of the random numbers they produce are given in table 1. The numbers are given to help the user verify his Fortran coding. Each sequence except Oak Ridge was started with $x_0 = k$; the table begins with the values of $x_1$. The Oak Ridge sequence was started with $x_0 = 2001$ and the table shows only the uniform (0,1) numbers produced by that generator. The Fortran-coded generators were tested on the Burroughs B6700, CDC 6400, IBM 360, 370, and 7094, PDP-11, Univac 1108, and Honeywell MULTICS computers, although only the Lewis-Goodman-Miller (1969) generator was tested on all these machines. In all cases the random numbers produced by the double precision Fortran code were the same as those produced by the published generators, thus verifying that Fortran double precision arithmetic does satisfy the required conditions in the cases tested.

The machine independence of this Fortran code of course is bought at the price of computational efficiency. To illustrate the magnitude of this price, we generated several sets of random numbers on an IBM System/360 Model 65 using the Fortran statements displayed in equations (1), (2), and (3) above and the LLRANDOM (Learmonth and Lewis, 1973) machine-language version of the Lewis-Goodman-Miller (1969) generator. The Fortran generator was coded as a subroutine and was compiled at optimization level 2 of the IBM FORTRAN IV (H) compiler. For comparability with non-IBM Fortran , DMOD was coded as an external function. In all cases the numbers were generated in vectors of 100 or 1000 per generator call and at least 100,000 numbers were drawn from each generator. The central processor time requirements for generating 100,000 numbers are shown in table 2, rounded to the nearest 5 seconds. These times are subject to perhaps 15 percent variation from run to run. In interpreting these results, it should be remembered that the Fortran language generators are not intended as replacements for high-quality production routines but merely as supplementary sources for otherwise unavilable random number sequences. The first column of this table indicates that the machine-language LLRANDOM code is at least two or three times as fast as the double precision Fortran code of equation 1 which in turn is twice as fast as the four-step code in equation 2 and five times as fast as the five-step generator in equation 3. These impressive percentage differences are diminished in practice, however, because it takes time to <u>use</u> the numbers as well as to generate them. To gauge this effect, we used the Box-Muller (1958) transform (a logarithm, a square root, a sine, and a cosine per pair of random numbers), to convert uniform numbers to normal deviates, representing a minimal amount of additional processing of the generated numbers. The middle column of table 2 indicates that use of the double precision Fortran codes of equations 1 and 2 instead of LLRANDOM entails penalties of about 20 and 50 , respectively, when minimal additional processing is required. The third column of the table represents the results of four runs of a program used by Wallis and others (1974) to simulate the sampling properties of statistics of lognormal populations. Two of the runs used the LLRANDOM routine to supply uniform random numbers; the other two used the Fortran code of equation 1 . In this case, the differences in generator times are nearly obscured by the normal job-to-job variations in measured CPU time.

## Conclusion

These results suggest not only that the proposed Fortran code will run without change on many machines but also that it will perform nearly as well as machine-specific generators, in the sense that it will yield substantially the same overall program execution times and costs to the user. Thus, no simulation practitioner who is able to use Fortran double precision arithmetic need feel constrained to use a random number source of dubious quality simply because it is the only one available on his machine. By use of simple double precision Fortran coding, he can take his pick from the whole menu of published multiplicative congruential generators.

Acknowledgements. I am indebted to M. E. Jennings, J. C. Liebman, S. M. Longwill, C. ReVelle, and D. A. Woolhiser for running demonstration jobs on their computers and to J. R. Slack for several helpful comments.

## REFERENCES

Ahrens, J. H., and U. Dieter, 1972, Computer methods for sampling from the exponential and normal distributions: Communications of the ACM, v. 15, no. 10, p. 873-882.

Ahrens, J. H., U. Dieter, and A. Grube, 1970, Pseudo-random numbers, a new proposal for the choice of multiplicators: Computing, v. 6, p. 121-138.

Box, G. E. P., and M. E. Muller, 1958, A note on the generation of random normal deviates: Annals of Mathematical Statistics, v. 29, p. 610-611.

Fellows, D. M., 1976, Comments on a general FORTRAN emulator for IBM 360/370 random number generator RANDU: INFOR, Canadian Journal of Operations Research and Information Processing, v. 14, no. 2, p. 183-187.

IBM Corporation, 1970, System/360 Scientific Subroutine Package Version III Programmer's Manual: Order no. GH20-0205, 454 p.

Learmonth, G. P., and P. A. W. Lewis 1973, Naval Postgraduate School random number generator package $^{LL}$RANDOM: Monterey, Calif., U.S. Naval Postgraduate School, Research Report NPS55LW3061, 54p.

Lewis, P. A. W., A. S. Goodman, and J. M. Miller, 1969, A pseudo-random number generator for the System/360: IBM Systems Journal, v. 8, no. 2, p. 136-146.

McGrath, E. J., and D. C. Irving, 1973, Techniques for efficient Monte Carlo simulation, Vol. II. Random number generation for selected probability distributions: U. S. National Technical Information Service, AD-762 722.

Nance, R. E., and C. Overstreet, Jr., 1975, Implementation of FORTRAN random number generators on computers with ones complement arithmetic: Journal of Statistical Computation and Simulation, v. 4, no. 3, p. 235-243.

Neave, H. R., 1973, On using the Box-Muller transformation with multiplicative congruential pseudo-random number generators: Applied Statistics, v. 22, p. 92-97.

Payne, W. H., J. R. Rabung, and T. P. Bogyo, 1969, Coding the Lehmer pseudorandum number generator: Communications of the ACM, v. 12, no. 2., p. 85-86.

Schrage, L., 1979, A more portable Fortran random number generator. ACM Transactions on Mathematical Software, v. 5, no. 2, p. 132-138.

Wallis, J. R., N. C. Matalas, and J. R. Slack, 1974, Just a Moment!: Water Resources Research, v. 10, no. 2, p. 211-219.

Table 1. Characteristics of random number generators

| Source | | AHRENS-DIETER (1972) | IBM SSP RANDU* IBM (1970) | LEWIS-GOODMAN-MILLER (1969) | OAK RIDGE (McGrath, 1973) | NEAVE (1973) | PAYNE-RABUNG-BOGYO (1969) |
|---|---|---|---|---|---|---|---|
| Modulus | | $2^{32}$ | $2^{31}$ | $2^{31}-1$ | $2^{47}$ | $2^{35}$ | $2^{31}-1$ |
| Multiplier | | 663608941 | 65539 | 16807 | $125^{5}$ | 131 | 630360016 |
| Random | 1 | 4216535657 | 393225 | 282475249 | 0.43390 | 17161 | 1549035330 |
| Integers | 2 | 1508633781 | 1769499 | 1622650073 | 0.74887 | 2248091 | 264620982 |
| | 3 | 3546922769 | 7077969 | 984943658 | 0.99043 | 294499921 | 529512731 |
| | 4 | 2333349949 | 26542323 | 1144108930 | 0.80661 | 4219751283 | 1896697821 |
| | 5 | 1227634681 | 95552217 | 470211272 | 0.96466 | 3031604185 | 2116530888 |
| | 6 | 1132643077 | 334432395 | 101027544 | 0.41083 | 19183026187 | 1923129168 |
| | 7 | 1351376673 | 1146624417 | 1457850878 | 0.50793 | 4715529633 | 1674201058 |
| | 1000 | 1201153165 | 1328681315 | 2021703321 | 0.75058 | 15087572451 | 1756984821 |
| | 10000 | 2739478445 | 630196675 | 1589873406 | 0.48243 | 23322702403 | 1049380835 |
| | 100000 | 277609197 | 751391107 | 1121266256 | 0.39847 | 15316017667 | 1926525262 |

*This generator is included in this tabulation to emphasize that it is only one of several alternative generators, all programmable in simple double precision FORTRAN, available to the simulation user.

Table 2. Central processor times (IBM System/360 Model 65) for generating 100,000 random numbers (to nearest 5 CPU seconds).

| Generator | No additional processing | Box-Muller transformation | Simulation program |
|---|---|---|---|
| FORTRAN (eq. 3) | 35-45 | --- | --- |
| FORTRAN (eq. 2) | 20 | 45 | --- |
| FORTRAN (eq. 1) | 10 | 35 | 75-85 |
| LLRANDOM | 5 | 30 | 70-80 |