BOEING SCIENTIFIC RESEARCH LABORATORIES DOCUMENT D1-82-0689

BOEING

SCIENTIFIC RESEARCH LABORATORIES

55

One-Line Random Number Generators and Their Use in Combinations

George Marsaglia

T. A. Bray

MATHEMATICS RESEARCH MARCH 1968

12

ONE-LINE RANDOM NUMBER GENERATORS AND THEIR USE IN COMBINATIONS

George Marsaglia, et al

Boeing Scientific Research Laboratories
Seattle, Washington

March 1968

D1-82-0689

ONE-LINE RANDOM NUMBER GENERATORS AND THEIR USE IN COMBINATIONS

by

George Marsaglia
and
T. A. Bray

Mathematical Note No. 551

Mathematics Research Laboratory

BOEING SCIENTIFIC RESEARCH LABORATORIES

March 1968

Summary

This is a discussion of some one-line random number generators, that is, generators requiring a single FORTRAN instruction, together with a description of some short FORTRAN programs which mix several such generators. Evidence suggesting that the simple congruential generators are unsatisfactory continues to grow; one of the most promising alternatives is to mix several simple generators. These composite generators do better in various tests for randomness than do the simple congruential generators used at many computer centers. If you wish to experiment with generators, or perhaps develop a more reliable generator than those currently available, you may want to consider some of the simple FORTRAN programs discussed here.

1. Introduction

If you need a random number in a computer, chances are you will call to a library subroutine which has coded, in machine language, a precise version of one of the simple congruential generators*, for example, $x_{i+1} = 5^{13}x_i$ modulo 2^{35}. By a precise version, we mean that the program is coded to give exact arithmetic modulo 2^{35}, 2^{32}, or whatever, with tests for overflow or sign bits, and appropriate adjustments. Alternatively, you may write your own generator in FORTRAN, carefully meeting the congruence relations with tests and adjustments for negative integers.

We would like to point out that one can take advantage of the way that FORTRAN handles integers in each particular computer to simplify congruential generators--in effect, generating random integers by means of a single FORTRAN instruction of the form I = I*K. Examples will follow for some particular computers with large numbers of users--IBM 360, IBM 7094 and SRU (Univac) 1108. If your computer is not one of those, you can easily make the necessary adjustments if you know details of how FORTRAN multiplies integers in your computer.

While a simplified FORTRAN generator which is faster and more convenient than the usual machine-language subroutines is an advantage in itself, the principal advantage seems to be the ease with which the one-line generators can be combined to produce composite generators. These will be discussed in Section 3.

---

*All of the references at the end of this article discuss congruential generators. Particularly good reviews are the book by Jansson [6], and papers by Chambers [1], and Hull and Dobell [5].

2. One-Line Generators

Consider first the IBM 360. Here FORTRAN integers are stored as 32 binary digits and multiplication of two integers produces a 32 bit integer which is the ordinary product modulo 2^{32}. However, when used in algebraic expressions, a stored integer I is considered positive or negative according to the relation

$$M(I) = \begin{cases} I & \text{if} \quad 0 \le I < 2^{31} \\ -2^{32} + I & \text{if} \quad 2^{31} \le I \le 2^{32} - 1. \end{cases}$$

Now if I is uniformly distributed over the interval $0 < I < 2^{32} - 1$, then the piecewise linear function M(I) will be uniform over its range, -2^{31} to $2^{31} - 1$. It follows that in the IBM 360 the single FORTRAN instruction I = K*I will, for each random integer I on $-2^{31} < I < 2^{31} - 1$, produce a new random integer in that interval, with cycle length and randomness or lack of randomness according to the standard properties of the congruential generator $x_{i+1} = kx_i$ modulo 2^{32}.

Combining a float instruction with our one-line generator, we then have this simple FORTRAN program for producing random uniform variates in the IBM 360:

Let I be the current random integer, and U the current random uniform variate. Then new values of I and U are given by:

```
I = I*K
U = .5+FLOAT(I)*.2328306E-9.
```

The constant in the second instruction is 2^{-32} in decimal form. The constant integer K can be chosen for maximum cycle length in the

form $K = 8m \pm 3$. The references contain suggestions on the choice of K. Almost any choice of odd K will serve when this generator is to be mixed with others in a composite generator.

For the IBM 7094, the situation is slightly simpler. There, positive integers are stored in FORTRAN in 35 bits, and multiplication of two positive integers is ordinary multiplication modulo 2^{35}. *Thus these two FORTRAN instructions will successively generate random integers I and random uniform variates U in the IBM 7094:*

```
I = I*K
U = FLOAT(I)*.291038305E-10.
```

The constant in the second instruction is 2^{-35} in decimal form. Remarks in the previous paragraph apply to the choice of K.

For the SRU 1108, the situation is again changed. There, FORTRAN integers are stored as 36 bits, and the product is as in ordinary arithmetic modulo 2^{36}, but when used in algebraic expressions or output, a 36 bit integer I is viewed as positive or negative according to the function

$$M(I) = \begin{cases} I & \text{if} \quad 0 \leq I < 2^{35} \\ -2^{36} + I + 1 & \text{if} \quad I \geq 2^{35}. \end{cases}$$

Thus these two FORTRAN instructions will successively generate random integers I and random uniform variates U in the SRU 1108:

```
I = I*K
U = .5+FLOAT(I)*.145519152E-10.
```

The constant in the second instruction is 2^{-36} and remarks above apply to the choice of K.

If you want one-line random integer generators, or two-line uniform variable generators, in computers other than those mentioned here, you must find out what happens in FORTRAN integer multiplication. You may be able to figure this out from a manual; we have found the easiest, and most foolproof, way is to merely compute, say, 3, 3^2, 3^3, 3^4,... in FORTRAN and have the results printed out. It is then easy to see what is going on; the reassuring fact is that in most computers the FORTRAN recurrence relation I = I*K will produce a full period of residues relatively prime to some modulus, 2^{32}, 2^{35}, 2^{36}, etc., and this set of residues can be appropriately adjusted with a float instruction. After all, the theory of congruential generators is based on equivalence classes of residues, and there is no need to use machine language subroutines or positive-negative tests to ensure that the representatives of the residue classes are the familiar least-positive ones.

3. Composite Generators

Numerous papers [1], [2], [4], [5], [6], [7], and [8] have reported unsatisfactory results for various simple congruential generators of the type $x_{i+1} = ax_i$ or $x_{i+1} = ax_i + b$ modulo 2^k or 10^k or some prime. While the search continues for simple generators which will pass increasingly more stringent batteries of tests, there are some who believe, (and we are among them), that no simple congruential generator is reliable enough to serve as the standard generator for a computer installation. A promising means of providing improved generators lies in mixing two simple generators, as suggested by MacLaren and Marsaglia [7], who used one generator to choose from 128 storage locations kept filled by a second generator; by Westlake [10], who wished to avoid the storage requirements of that method and instead used the sum of two generators after using a portion of one output for a random cyclic permutation of the bits of another. Then Gebhardt, [3], used the method of M and M, [7], but got good results with only 16 storage locations and a single Fibonacci sequence, $x_{i+2} = x_{i+1} + x_i \bmod 2^k$, to both fill the storage locations and select them.

By using the one-line FORTRAN generators discussed above, we can develop FORTRAN composite generators in a variety of ways. Short and fast programs will result even if three generators are mixed--one to fill, say, 128 storage locations; one to choose a location from the 128; and a third thrown in just to appease the gods of chance. Why be half (or $\frac{2}{3}$) safe?

Here are sample composite generators for the three computers mentioned above. In each case, assume we have assigned initial odd integer values to N(1),N(2),...,N(128),L,M, and K. Then these FORTRAN instructions will provide uniform random variables U on the interval 0 < U < 1:

IBM 360,

```
L = L*ML
M = M*MM
J = 1+IABS(L)/16777216
U = .5+FLOAT(N(J)+L+M)*.2328306E-9
K = K*MK
N(J) = K
```

IBM 7094,

```
L = L*ML
M = M*MM
J = 1+L/268435456
U = FLOAT(N(J)+L+M)*.291038305E-10
K = K*MK
N(J) = K
```

SRU 1108,

```
L = L*ML
M = M*MM
J = 1+IABS(L)/268435456
U = .5+FLOAT(N(J)+L+M)*.145519152E-10
K = K*MK
N(J) = K
```

In these programs, the integer J is used to choose from N(1) to N(128); J comes from the random integer L after division by the appropriate power of 2. In forming U, the argument of the float function is the sum (modulo 2^{32}, 2^{35}, or 2^{36}) of the randomly chosen N(J), the random integer L used to find J, and a third (gratuitous?) random integer M.

The random integers L, M and K are the outputs of the three one-line generators; constants ML, MM and MK can be chosen in the form 8m ± 3 to ensure long periods. Chances are that randomly choosing six-to eight-octal-digit integers ending in 3 or 5 will provide satisfactory multipliers. Van Gelder [9], found that these octal multipliers were promising: 10405, 20005, 105005, while we got excellent test results with decimal integers ML = 65539, MM = 33554433, and MK = 362436069.

The composite generator described above is short and easy to program in FORTRAN. Since the statistical properties of composite generators appear to be better than those of simple generators, the only drawbacks would seem to be storage requirements and speed. The programs above require stored values N(1),N(2),...,N(128). For modern computers, this seems a trifling requirement. How often do you, or your associates, run a Monte Carlo problem so big that you can't afford 128 storage locations? For such unusual situations, or for small special purpose computers where one still demands a generator beyond suspicion, the storage requirements can be reduced to 64, 32 or 16. As for speed even though the composite generator mixes 3 congruential

generators and uses a float instruction, the program is still fast enough to produce huge quantities of uniform variates should they be needed:

Rates for Producing Uniform Variates

Computer	Two-Line	Composite (In-line)	Composite (Subprogram)
IBM 360/44	7,200/sec	3,500/sec	2,000/sec
IBM 7094	18,800/sec	6,700/sec	4,900/sec
SRU 1108	86,100/sec	25,000/sec	19,200/sec

The composite generator (listed on page 6) has two rates--as a 6-line generator incorporated in a FORTRAN program, or as a separate FORTRAN subprogram.

References

[1] Chambers, R. P., "Random number generation on digital computers". IEEE *Spectrum 4* (February 1967), pp. 48-56.

[2] Coveyou, R. R., and MacPherson, R. D., "Fourier analysis of uniform random number generators". *J. Assoc. Comput. Mach. 14*, No. 1, (January 1967), pp. 100-119.

[3] Gebhardt, Friederich, "Generating pseudo-random numbers by shuffling a Fibonacci sequence". *Math. Comp. 21*, No. 100, (October 1967), pp. 708-709.

[4] Greenberger, M., "Method in randomness". *J. Assoc. Comput. Mach. 8*, pp. 177-179, 1965.

[5] Hull, T. E., and Dobell, A. R., "Random number generators". *SIAM Rev. 4*, pp. 230-254, 1962.

[6] Jansson, Birger, "Random Number Generators". Victor Pettersons Bokindustriab, Stockholm, 1966.

[7] MacLaren, M. D., and Marsaglia, G., "Uniform random number generators". *J. Assoc. Comput. Mach. 12*, pp. 83-89, 1965.

[8] Peach, P., "Bias in pseudo-random numbers". *J. Amer. Statist. Assoc. 56*, pp. 610-618, 1961.

[9] Van Gelder, A., "Some new results in pseudo-random number generation". *J. Assoc. Comput. Mach. 14*, No. 4, (October 1967), pp. 785-792.

[10] Westlake, W. J., "A uniform random number generator based on the combination of two congruential generators". *J. Assoc. Comput. Mach. 14*, No. 2, (April 1967), pp. 337-340.