Technical Report CS74013-T

THE MIXED METHOD OF RANDOM NUMBER GENERATION: A TUTORIAL†

Claude Overstreet, Jr.*
and
Richard E. Nance

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

August 1974

†Research supported in part by a grant to Claude Overstreet, Jr. by the Department of Computer Science, V.P.I. and S. U.

*Department of Computer Science, Bowling Green State University, Bowling Green, Ohio.

Abstract

Several motivations are recognized for user-defined random number generators in preference to built-in generators. The mixed method of random number generation is discussed, and the conditions for achieving full period with a modulus of 2^b are explained. Implementation of mixed random number generators is affected both by the computer and language used. Guidelines are presented for realizing acceptable mixed generators on several machines using the FORTRAN, PL/1 and SNOBOL4 languages.

Keywords: mixed random number generator, maximum period, implementation, language effects, machine effects.

Computing Reviews categories: 3.9, 5.39

Introduction

Random number generation as discussed in the current literature normally refers to the generation of a sequence of independent values that are uniformly distributed over the interval from 0 to 1. These values in turn form an essential part of programs incorporating uncertainty in simulations or the production of music, poetry and other art forms.

Many programs that utilize random numbers rely on a built-in random number generator, supplied with the software support by the hardware manufacturer. While these built-in generators might be adequate for most purposes, a user might prefer to construct his own generator for several reasons:

1. A built-in random number generator is not available in the particular language favored by the user. This is often the case with languages suitable for non-numerical applications such as SNOBOL4.
2. Sometimes only a unique sequence of values can be generated with a built-in generator. By constructing his own generator, the user can enable the production of varied sequences based on the assignment of initial values or the redefinition of other parameters.
3. A program which utilizes a built-in generator on one machine seldom produces the same results when moved to a different machine. This problem is eliminated with a user-defined generator.
4. Some built-in generators do not appear to produce independent values drawn from a uniform distribution and prove inadequate with regard to the user's criteria for randomness.

It is the purpose of this paper to present a general discussion of one of the most commonly used methods of random number generation and to offer guidelines for its implementation. Additionally, the implementation of random number generators utilizing this method is illustrated for several different languages and machines.

Properties of the Mixed Method

Several methods of generating random numbers are currently in use. Among these methods are the additive congruential [1], the multiplicative and mixed congruential [2], the quadratic congruential [3], and the shift register or Tausworthe technique [4]. An extensive bibliography on random number generation [5] provides references to literature treating all of the methods above. The principal subject of this paper is one of the most commonly used, i.e. the mixed congruential method.

We deal with the particular form of the mixed method defined by the relation,

$$X_{n+1} \equiv aX_n + c \pmod{2^b} \quad .$$ [1]

The above relation indicates that to generate a succeeding value in the sequence (X_{n+1}), we take the previous value (X_n), multiply it by a and then add the constant c. This result is then taken modulo 2^b, or equivalently the integer remainder is kept after division by 2^b. We illustrate the method using a small modulus value to simplify matters

$$X_{n+1} \equiv 5X_n + 3 \pmod{8} \quad .$$

With the initial value $X_0 = 1$, then $X_1 \equiv 5+3 \equiv 8 \pmod{8} = 0$. Proceeding in this manner, the complete sequence of generated values is 1, 0, 3, 2, 5, 4, 7, 6, 1, ...

Beginning with the value X_0, the multiplication and addition modulo 2^b generates successive values until eventually $X_n = X_0$, at which point the sequence

[1]The relationship $Z \equiv Y \pmod{M}$, read as "Z is congruent to Y modulo M", indicates **that** the difference in the integers Z and Y is divisible by M. The congruence relationship employed in the mixed method is sometimes called "the power residue" method since the values produced are residues modulo M.

of values repeats itself. The number of values generated in a sequence before the sequence repeats is referred to as the <u>period length</u>. The maximum possible period length for the mixed generator modulo 2^b is 2^b. Obtaining a maximum period is possible only if certain conditions are met [2]:

1. The multiplier a must satisfy $a \equiv 1 \pmod 4$; the integer remainder of a/4 must be 1. A good rule of thumb is to pick a in the vicinity of $2^b/2$ where 2^b is the modulus.
2. The constant c must be an odd number.

Note that no requirement is placed on the initial value X_0.

The values produced by the above relation lie between 0 and 2^b-1. To transform these values to decimal fractions between 0 and 1 requires an additional step in which each of the generated values is divided by 2^b.[2] This final division by 2^b then produces the desired random numbers. In the example above, we would divide each generated value by 8 to get the desired sequence .125, 0, .375, .250, .625, ...

Before considering the implementation of the mixed generator, we note several important points concerning the general nature of the method:

First, if the multiplier and additive constant are unchanged, then using the specific initial value X_0 allows generation of an identical sequence of values for subsequent computer runs.

Second, each sequence repeats after 2^b values are generated. This could be of concern if 2^b is relatively small. Normally the period length will be more than adequate, e.g. for the IBM 360-370 a period length of 2^{31} = 2,147,483,648 can be produced.

[2]This result is accomplished in assembly language versions by a shift operation rather than a division, <u>i.e.</u> the radix point is moved from the right-most position (for the integer value) to the left-most position (for the decimal fraction). The effect is the same using either operation.

Third, for any initial value X_0, we generate the identical overall period of 2^b values. A different X_0 simply designates a new starting point in the sequence of 2^b values. Normally, the period is large relative to the set of numbers desired so that, by changing X_0, we produce an entirely new set of values.

Fourth, different sequences are generated for different values of the multiplier a and/or the additive constant c.

One final point that applies to all generators and all methods should be emphasized. No definitive set of rules exists to guarantee completely acceptable statistical behavior. The statistical behavior of a random number generator can be evaluated only after subjecting values produced by the generator to a series of statistical tests.

Effect of Machine and Language

On most contemporary digital computers integer values are given a base 2 representation. An integer value in base 2 is limited in size by the number of available binary places, or bits, available to store the number. For example, if we are limited to 3 decimal places to store a number in base 10, the largest value we can represent is $10^3-1 = 999$. If we are limited to 3 binary bits, the largest representable value is $2^3-1 = 7$.

The word-length of a machine is the number of bits available to store a signed integer value. One bit will be reserved to indicate whether the sign is positive or negative, and the remaining bits represent the magnitude of the value. For a machine with a word length of k bits, we choose 2^{k-1} as the modulus of the mixed generator. This particular choice of the modulus simplifies the determination of the integer remainder from the congruence relation. In fact, the remainder is produced automatically although two small problems can arise.

Since such large values are produced in generating random numbers, the multiplication and addition inadvertently can cause the result to "overflow" into the sign bit of the representation. A change in the sign bit indicates an incorrect negative value. If a negative value is produced, the change to a proper positive value is effected as follows:

1. If integer values are represented in a one's complement form, add $(2^{k-1} - 1)$ to the negative value.
2. If integer values are represented in a two's complement form, add $(2^{k-1} - 1) + 1$ to the negative value.

One's- and two's-complement representation are the two representations of negative values.[3] For further information on one's- and two's-complement representation, the reader is referred to Gear [6]. A description of a particular machine should indicate the representation in use. In the way of a warning, applying the absolute value to the result often leads to incorrect values.

A second potential problem is that an overflow from an arithmetic operation might be detected as an error condition causing execution of the program to be terminated. This happens in PL/1 on the IBM 360-370 and can be solved by directing the PL/1 compiler to ignore overflow as an error condition. This also occurs in SNOBOL4, and since the error condition is not easily disabled, we are forced to use a smaller modulus to avoid overflow.

[3]The most common large computers employing one's-complement representation are the UNIVAC 1107-1108 and the Control Data 6000 and CYBER series. Two's-complement representation is used in the IBM 360-370 series, PDP-10, and RCA SPECTRA 70 series.

Examples of the Mixed Random Number Generator

Below we illustrate the implementation of the mixed random number generator for several different machines and languages. Each of the generators presented has been subjected to a variety of statistical tests [7], and all of the generators appear to produce satisfactory results. Each of the generators presented requires one additional instruction, which is not included in the examples. The necessary instruction is an assignment of an initial value. This is accomplished by assigning the first value of IX in the following programs. The initialization of IX should be performed only once, at the start of the program. The random value (called a "random variate") lying between 0 and 1 is returned as the value of RAN in each of the examples below.

FORTRAN:

IBM 360-370, a two's-complement machine with a word-length of 32 bits ($2^{31}-1 = 2147483647$).

```
IX = 32949 * IX + 8237
IF(IX.LT.0) IX = (IX + 2147483647) + 1
RAN = IX / 2147483648.0
```

UNIVAC 1108, a one's-complement machine with a word-length of 36 bits ($2^{35}-1 = 34359738367$).

```
IX = 32949 * IX + 8237
IF(IX.LT.0)  IX = IX + 34359738367
RAN = IX / 34359738368.0
```

CDC-6000 series, a one's-complement machine with 48 bits available to represent an integer value. Overflow does not disturb the sign bit. ($2^{47}-1 =$ 140737490355327)

```
IX = 32949 * IX + 8237
RAN = IX / 140737490355328.0
```

PL/1:

IBM 360-370, overflow is detected as an error condition and must be disabled. Overflow does not disturb the sign bit.

```
DCL (IX) FIXED BINARY (31);
         :
(NOFIXEDOVERFLOW):  IX = 32949 * IX + 8237;
RAN = IX / 0.2147483648E+10;
```

SNOBOL4:

IBM 360-370, overflow is detected as an error condition that cannot be disabled. Consequently, we must resort to a smaller modulus. As a general guideline, if the representation of an integer value uses N bits, use as a modulus 2^k where k is the integer part of (N-1)/2. Thus on the IBM 360-370 we use $2^{31/2} = 2^{15}$, on the UNIVAC 1108 use 2^{17}, and on the CDC 6000 series use 2^{24} as a modulus.

```
IX = REMDR(IX * 32949 + 8237,32768)
RAN = IX / 32768.0
```

BASIC:

Unfortunately, most BASIC translators do not allow the representation of integer values; all values are stored as real values (a fraction with an exponent). This eliminates the construction of a random number generator by the user. The RND function is supplied as part of the BASIC language.

Conclusions

For the user who desires to construct his own random number generator we have presented guidelines and design considerations for implementing the mixed congruential methods. The mixed generator is illustrated using several different languages and machines. We feel that by adhering to these guidelines, a user should be able to construct a random number generator that is adequate for most applications.

References

1. Taussky, O. and J. Todd, "Generation and Testing of Pseudo-Random Numbers", Symposium on Monte Carlo Methods, John Wiley, 1956, pp. 15-28.
2. Hull, T. E. and A. R. Dobell, "Random Number Generators", SIAM Review, V. 4, 1962, pp. 230-254.
3. Overstreet, C. L. and Richard E. Nance, "A Random Number Generator for Small Word-Length Computers", Proceedings ACM Annual Conference, August 1973, pp. 219-223.
4. Tausworthe, R. C., "Random Numbers Generated by Linear Recurrence Modulo Two", Mathematics of Computation, V. 19, 1965, pp. 201-209.
5. Nance, Richard E. and C. L. Overstreet, "A Bibliography on Random Number Generation", Computing Reviews, October 1972, pp. 495-508.
6. Gear, C. W., Computer Organization and Programming, McGraw-Hill, 1969.
7. Overstreet, C. L.,"A FORTRAN V Package for Testing and Analysis of Pseudorandom Number Generators", Southern Methodist University, Technical Report CP-72009, 1972.