COBOL Simulation: Random Number Generation for Binary and Decimal Computers

by

Francis J. Brewerton
Middle Tennessee State University

Elias R. Callahan
Middle Tennessee State University

R. Wayne Gober
Middle Tennessee State University

ABSTRACT

Although the Common Business Oriented Language was originally designed for use in business data processing, the language is now being employed as a simulation language under certain limiting conditions. Factors influencing the application of the language to simulation studies include its popularity, its self-documenting characteristic, its "believability," and its efficiency in programming the triangular distribution.

A necessary requisite in any simulation study is the programming of random number generators to simulate the random occurrence of various events. Several methods of generating random numbers are available, but the technique most frequently used is the power residue method. Since most simulation studies are programmed with FORTRAN (a general purpose language), simulation languages (SIMSCRIPT, GPSS, GASP, etc.), or machine languages, little attention has been devoted to programming the power residue method of random number generation with COBOL.

A procedure is presented which describes and discusses COBOL programming of random number generation for the binary computer and the decimal computer utilizing the power residue method. Program excerpts are provided to illustrate the procedure, and comparative differences in COBOL programming for the two computer types are noted. Conditions most favorable to COBOL programming of simulation studies are also discussed, as well as the conditions under which COBOL programming is not recommended.

INTRODUCTION

As a programming language, COBOL (Common Business Oriented Language) has achieved wide acceptance among practitioners, particularly business practitioners. This widespread acceptability is based in large part on the general nature of the language and the ease with which it can be mastered. As a simulation language, COBOL has received considerably less attention than other languages which were designed especially for simulation work. Nevertheless, the frequency with which COBOL is being used for simulation work is increasing; the bases for its increased application to simulation studies include its popularity, its ease of documentation, its believability, and the use of the triangular distribution in management simulation studies.

COBOL'S GENERAL ATTRACTIONS

Obviously, COBOL is one of the most popular languages. This popularity hinges in large part on its ready absorption by students. Its largest single user is the

federal government, as COBOL is the official programming language used by federal agencies. Another characteristic contributing to the popularity of COBOL is its self-documenting aspect. Unlike FORTRAN, COBOL is almost perfectly self-documenting. Programming statements expressed in the COBOL language are highly explanatory and seldom ambiguous, a feature derived from the fact that the language is essentially conversational English.

Another attractive feature of COBOL is its "believability." The believability seems to be associated with intuitive validation of program statements. Many users of COBOL do not possess scientific backgrounds and consequently tend to be unsure of their capabilities in programming, modeling, and simulation. Consequently, it is at times difficult for an individual who does not have a scientific background to read a statement in FORTRAN and determine its validity. By contrast, when programmers without scientific backgrounds read COBOL statements, they generally experience some sense of comprehension and tend to be more confident their programs will work.

Probably the greatest impetus for applying COBOL to simulation studies has come from more widespread use of the triangular distribution in management simulations. The triangular distribution seems to be of particular interest to managers because of its capability to approximate other statistical distributions and because of its ease of parameter estimation.* The foregoing discussion may seem to have little to do with COBOL as a simulation language. However, the managerial popularity of the distribution, its ease of documentation, and its believability combined with the fact that it can be very easily programmed with COBOL provide a strong incentive for simulation applications of the language. The triangular distribution seems to have all the elements which are desirable for simulation work: it is flexible in shape, has easily estimated parameters, can approximate continuous or discrete data, and can be COBOL programmed without having to resort to writing subroutines. If distributions other than the triangular are used, this advantage is almost always lost. Furthermore, COBOL becomes very unattractive from an efficiency standpoint when other distributional forms are utilized in the simulation, as other general purpose languages such as FORTRAN are more efficient for this purpose. Whatever attractiveness COBOL has as a simulation language is thus conditional on the use of the triangular distribution to represent the distributions of the random variables contained in the simulation model.

RANDOM NUMBER GENERATION

With the conditions under which COBOL becomes attractive as a simulation language thus established, the discussion may now shift to the COBOL programming of random number generation. In any simulation in which stochastic processes are involved, the employment of a sequence of random or near-random numbers will be required to help simulate the random occurrence of various events in the simulation. By definition, a random number can occur only as a result of a random process. But truly random processes are difficult to identify, and consequently, truly random numbers are more difficult to produce than many individuals realize. Fortunately, the question of randomness is not as critical as it might appear, and near-random numbers can be used quite effectively with little loss of accuracy in simulation study results. Several schemes have been devised for mechanically, electronically, or otherwise artificially generating pseudo-random numbers. All of these schemes rely on recursive mathematical

*For a more complete discussion of the triangular distribution see "Management Science Applications of the Triangular Distribution: Some Pros and Cons," in 1975 Midwest AIDS Proceedings, April 1975, Indianapolis, Indiana, pp. 428-431.

relationships that produce sequences of numbers which demonstrate a pattern that is generally so obscure, so subtle, and so seldom recurring, that it cannot be readily detected. These numbers, although not truly random, may be used for simulating random events without injecting inordinately large amounts of error. For the remainder of this paper, the unqualified use of the term "random number" will refer to a uniformly distributed random number drawn from a pseudo-random number generator.

Many methods have been devised for random number generation, but the center square (midsquare) method and the Fibonacci Series method seem to be the most historically significant. The center square method squares a 2n digit number and selects the middle 2n digits from the 4n digit product for the next number in the sequence. Several variations of this method have been devised, but the method has lost popularity because of the relatively short periods between cycles and its slowness. The Fibonacci Series method selects two beginning numbers u_0 and u_1, and computes

$$u_{n+1} \equiv u_n + u_{n-1} \pmod{m}. \qquad (1)$$

Values of m which have proven to be satisfactory include $m = 2^b$ and $m = 10^d$. The method is simple and fast but produces sequences of numbers which demonstrate significant nonrandomness. The Fibonacci Series method has been recognized as essentially a power residue method with an insufficiently large multiplier.

Presently, virtually all computer library random number generators employ some variation of the power residue (congruence) method. The three most commonly used variations are the multiplicative, mixed, and additive methods; of these, the multiplicative method seems to be most popular. The multiplicative congruence procedure employs two constants a and m, both of which are nonnegative integers; these constants are used to derive the $(i + 1)$th number in the sequence. The $(i + 1)$th number is obtained by (1) multiplying the ith number by the constant, a, (2) dividing this product by the modulus, m, and (3) equating the residue or remainder as the $(i + 1)$th value in the sequence. This procedure is described notationally as

$$R_{i+1} \equiv aR_i \pmod{m}. \qquad (2)$$

Obviously, the choice of a, R_0, and m influences the integrity of the random numbers generated. An appropriate choice of the modulus depends upon the computer number system being used. Probably the most natural choice of values for m is one that equals the capacity of the computer word.

BINARY COMPUTERS AND DECIMAL COMPUTERS

For a binary computer, an appropriate choice of the modulus is 2^b, with b defined as the number of bits in the computer word. For a decimal computer an appropriate choice for m is 10^d, with d defined as the number of digits in the computer word. The maximum period or sequence of random numbers is realizable only if R_0 and a are chosen in a particular fashion. Again, the most appropriate choice is a function of the type of computer being utilized. For the binary computer, a should be selected so that

$$a = 8T \pm 3, \qquad (3)$$

in which T may be any positive integer and R_0 is selected as a positive odd integer. For the decimal computer, a should be selected so that

$$a = 200T \pm B, \qquad (4)$$

in which T may be any positive integer and B is assigned the value of 3, 11, 13, 19, 21, 27, 29, 37, 53, 59, 61, 67, 69, 77, 83, or 91. The seed value for R_0 in the decimal case may be any odd positive integer which is not perfectly divisible by 2 or 5.

COMPUTATIONAL PROCEDURE

When generating random numbers using COBOL, the programmer has the option of generating random numbers using a binary computer or a decimal computer, an option which does not exist when FORTRAN is used. Regardless of the programmer's choice of coding option, the computational process (for either type of coding) can be summarized by the following procedural steps:

(1) Select any integer which contains less than nine digits and designate it as the beginning value or seed, R_0.

(2) Multiply R_0 by an integer, a, which contains at least five digits.

(3) Multiply the product obtained in step 2 by a number which is equal to 1/m.

(4) Assign the decimal portion of the result obtained in step 3 as a random number on the unit interval.

(5) Use the random number (less decimal point) obtained in step 4 to re-seed R_0 in step 2.

The desired accumulation of random numbers is obtained by repeating steps 2 through 5 in the procedure above to obtain the necessary number of replications.

APPLICATION AND DISCUSSION

The COBOL syntax for generating random numbers via the binary computer method and the decimal computer method is illustrated in Figure 1 and Figure 2 respectively. The addition of computational usage elements into the COBOL syntax shown in these figures increases the efficiency of the routines, but is not included here because of system differences in usage computational representation. The user is encouraged to add these elements (using the appropriate representation) so that programming efficiency is maximized.

The decimal computer method shown in Figure 2 has certain features which recommend it over the binary method. To begin with, the programming of the routine to generate the numbers is straightforward; however, this is not unique to the decimal method. A more important feature of the decimal method is that the programmer need have little concern regarding the number of bits in the word size of the machine; this is not true of the binary computer method. Furthermore, when using the binary computer method in COBOL, difficulties may occur in making the proper selection of picture size, usages, and initial values. These same difficulties do not exist when employing the decimal method, as picture size is automatically determined by the number of digits in the random number; in short, many of the potential problem sources in the binary method are removed or their impacts are minimized when the decimal method is used.

Figure 1
Binary Computer Syntax

```
WORKING-STORAGE SECTION.
77  IX  PIC 9(6) VALUE 35797.
77  IY  PIC 9(6).
77  RANDX   PIC V9(6).
77  TWO-35-1  PIC 9(12) VALUE 34357938367.


PROCEDURE DIVISION.
100-OPEN.


200-GENERATE.
    COMPUTE IY = IX * 03125.
    IF IY IS NEGATIVE ADD TWO-35-1 TO IY.
    COMPUTE RANDX = IY / TWO-35-1.


    MOVE IY TO IX.
```

Figure 2
Decimal Computer Syntax

```
WORKING-STORAGE SECTION.
77  RANDOM  PIC  9(6).
77  IX-SEED  PIC 9(6) VALUE 35797.


PROCEDURE DIVISION.
100-OPEN.


200-GENERATE.
    COMPUTE RANDOM = IX-SEED * 1003.
    COMPUTE IX-SEED = RANDOM / 1000000.
    COMPUTE IX-SEED = RANDOM - (IX-SEED * 100000).
```

The relationship between the picture size and d deserves additional explanation. Since d is the number of digits in the computer word, it can also be related to the size of the random number. For example, if 6-digit random numbers are desired, then d must be set equal to six. The initial seed value, R_0, may be any integer of length d and not perfectly divisible by 2 or 5. The value of the constant to be used as a multiplier, a, can be determined by selecting a value for a which is close to $10^{d/2}$, and choosing the integer which is closest to this value; this approach is described in step 2 of the generation procedures discussed in an earlier section of this paper. In Figure 2, a 6-digit random number is desired. The initial value used for a is 1003; this value was obtained using equation 4, with the value of B fixed at 3. Successive iterations of the process will thus produce a sequence of random numbers without a noticeable pattern.

CONCLUSIONS

The preceeding discussion leads to the conclusion that decimal coding is easier, quicker, and more flexible than binary coding when programming simulation studies in COBOL. These distinct advantages should not be disregarded by the programmer, at least not indiscriminately. To be sure, circumstances may arise which dictate the use of the binary syntax, and the programmer must take these circumstances into consideration.

The procedures developed in this paper apply only when (1) the simulation study is being programmed in COBOL, (2) the triangular distribution is being used to represent the distribution of the random variables, and (3) other generally favorable conditions prevail. Obviously, the number of programmers operating within this special set of conditions will be small at any particular point in time. Nevertheless, the material presented here should be of some significance to those programmers to whom the conditions do apply, and is being offered in the interest of maximizing the efficiency of their programming effort.

REFERENCES

1. Brewerton, F. J., Callahan, E. R., and Gober, R. W., "Conditions, Criteria, and Caveats for Computer Simulation with COBOL," 1975 Winter Computer Simulation Proceedings, December, 1975, Sacramento, California.

2. IBM Corporation, "Random Number Generation and Testing," Reference Manual C20-8011, White Plains, New York, 1959.

3. Naylor, T. H. et al, Computor Simulation Techniques, John Wiley and Sons, New York, 1966.

4. Philippakis, Andreas S. and Kazmier, Leonard J., Information Systems Through COBOL, McGraw-Hill Book Co., New York, 1974.

5. Shannon, Robert E., Systems Simulation: The Art and Science, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.