MULTIPLE SEQUENCE RANDOM NUMBER GENERATORS

Joe H. Mize
Oklahoma State University

Abstract

Many discrete event simulation models utilize a single random number sequence, regardless of the number of random variables included in the model. This paper illustrates the advantages of utilizing a unique random number sequence for each random variable. An example is included to demonstrate the resulting effects. A method is described for converting a single sequence random number generator to a multiple sequence generator. An application is included that compares simulation results obtained using a single sequence of random numbers to those obtained using a unique sequence for each random variable in the model.

Introduction

Most simulation studies are concerned with systems that contain random phenomena; interarrival times, service times, demand values, time to failure, etc. In order to produce useable results, a model of such a system must provide a mechanism for representing these random phenomena.

Considerable effort has been directed toward the development of random variate generators. Most of the generators in use today make use of uniformly distributed random numbers. For examples, see Naylor, et.al. (1966), Mize and Cox (1968), and Schmidt and Taylor (1970). These random numbers, in turn, are converted to a random deviate from an appropriate probability distribution. Thus, simulation modeling requires the availability of a random number generator that will produce numbers uniformly distributed on the interval zero to unity.

The most widely used method for generating such numbers is the "congruential method," first proposed by Lehmer (1959). Almost every computer center has a random number function or subroutine on its system library. Most of these generators create a single sequence of random numbers, beginning with a user provided "seed" value.

The purpose of this paper is to demonstrate that in many simulation models a single sequence of random numbers is inadequate. This rather crucial feature of

simulation modeling has not been explained sufficiently in most texts and other literature related to simulation. The two most widely used simulation languages, GPSS (see Gordon, 1969) and SIMSCRIPT (see Kiviat, et.al., 1968) provide a limited number of random number streams, however, the reasons for using multiple streams are not made clear.

Most random number generators based on the congruential method can be modified easily to produce as many parallel sequences as the user desires.

One of the prime advantages of using simulation as a means of studying complex systems is that it offers the analyst a means of comparing one system configuration to any number of other configurations. Furthermore, it is possible to construct the model such that the alternatives are evaluated using the identical sequences of event occurrences for each alternative.

Special care must be exercised in the construction of such a model, however. This is not always a straightforward process. A common pitfall of an inexperienced analyst is to believe that his simulation model compares competing alternatives against identical event occurrences, when in fact it does not. The example in the next section illustrates the nature of the problem.

Demonstrative Example

Consider the simulation of a simple queueing system in which customers arrive at random times and the length of time required for service is also random. Specifically, suppose that the time between customer arrivals is a uniformly distributed random variable over the interval 0 to 4 minutes. Service time is also uniformly distributed over the interval 1 to 6 minutes.

It is desired to simulate the performance of this system, first with one service facility and then with two. If the model is constructed properly, the performance of the system can be simulated for the two configurations such that the exact same patterns of arrivals and service times will occur. In this way, the two system configurations will be compared under identical conditions. Any differences in performance may then be attributed to the difference in system configuration (one service facility versus two) and not to "experimental error." This removes from consideration a source of variation which would be impossible to eliminate if the experiment were to be conducted in the real world.

Denote the time between customer arrivals as X, and service time as Y. It can be shown that the following process generators will produce random deviates for the two random variables, according to the probability functions specified above:

$$X = 4.0 * RN$$

$$Y = 1.0 + 5.0 * RN$$

RN is a uniformly distributed random variable on the unit interval. If our model were computerized, RN could be obtained by calling the system random number generator each time a new value of X and Y is needed.

We shall observe the performance of this model for ten arrivals to the system, first with one service facility and then with two. We will need a random number to determine the time of arrival of each customer and another to determine the service time of

each customer. We will use single digit random numbers to simplify computations, although in an actual experiment we would use much greater accuracy. We will use the following sequence of random numbers:

.3, .6, .5, .5, .0, .8, .9, .7, .9,
.6, .6, .5, .9, .8, .3, .3, .4, .4,
.1, .5, .0, .5, .6, .4, .8, .9, .0,
.7, .4, .0, .9, .4, .6, .9, .7, .1,
.3, .1, .9, .9, .2, .9, .6, .2, .2,
.2, .2, .5, .0, .4

We will perform our simulation in an "event-oriented" manner. In such a simulation, process generators are used to determine when particular events will occur and all events are allowed to occur in their natural sequence.

The simulation results using a single random number sequence and one server, are shown in Table 1. The random numbers used were taken in consecutive order from the source included above. There are three numbers in each cell of the second and fourth columns of Table 1. In the upper left hand corner is simply the sequence number of the random number taken from the above source and used in the determination of the occurrence time of the indicated event. In the upper right hand corner is the random number itself. The value in the lower portion of the cell in column two indicates the time of arrival of a particular customer and is determined by substituting the random number (RN) in the upper right corner into the equation $X = 4.0 * RN$ and adding the resulting value to the arrival time of the previous customer. The value in the lower portion of the cell in column four indicates the service time for a particular customer and is obtained by substituting the random number (RN) in the upper right hand corner into the equation $Y = 1.0 + 5.0 * RN$. Column three indicates when service may begin for a particular customer and is simply the larger of the two values, Time of Arrival of that customer and End Service of the previous customer. Column five is simply the sum of columns three and four for a particular customer.

It will be instructive to trace the occurrence of a few events in the model. The arrival of the first customer is obviously the first event. The time of its occurrence is determined by using the first random number from the basic source. This gives an arrival time of 1.2. Since no previous customer is in service, the first customer may go immediately into service. We must now determine the service time for this customer by using the second random number from the basic source; $Y = 1.0 + 5.0 * 0.6 = 4.0$. Thus, customer one begins service at 1.2 and ends at 5.2.

The arrival time of customer two is determined by using the third random number in the manner described previously. It is found that customer two arrives at 3.2. This is prior to the End Service time of customer one. Thus, the service time for customer two cannot be determined at this point in the simulation. The next-event orientation of our model requires that we place customer two in a holding pattern and go on to determine the arrival time of customer three, using the fourth random number from the basic source. This results in an arrival time of 5.2 for customer three, the exact same time that customer one completes service. We now use the fifth

Table 1. Single R.N. Sequence; One Server

Customer Number	Time of Arrival	Begin Service	Service Time	End Service
0	0	0	0	0
1	1 .3 1.2	1.2	2 .6 4	5.2
2	3 .5 3.2	5.2	5 .0 0	5.2
3	4 .5 5.2	5.2	6 .8 5	10.2
4	7 .9 8.8	10.2	9 .9 5.5	15.7
5	8 .7 11.6	15.7	12 .5 3.5	19.2
6	10 .6 14.0	19.2	14 .8 5.0	24.2
7	11 .6 16.4			
8	13 .9 20.0			
9	15 .3 21.2			
10	16 .3 22.4			

Table 2. Single R.N. Sequence; Two Servers

Customer Number	Time of Arrival	Begin Service Server No. 1	Begin Service Server No. 2	Service Time	End Service Server No. 1	End Service Server No. 2
0	0	0		0	0	
1	1 .3 1.2	1.2		2 .6 4	5.2	
2	3 .5 3.2		3.2	4 .5 3.5		6.7
3	5 .0 3.2	5.2		7 .9 5.5	10.7	
4	6 .8 6.4		6.7	9 .9 5.5		12.2
5	8 .7 9.2	10.7		11 .6 4.0	14.7	
6	10 .6 11.6		12.2	13 .9 5.5		17.7
7	12 .5 13.6	14.7		15 .3 2.5	17.2	
8	14 .8 16.8	17.2		17 .4 3.0	20.2	
9	16 .3 18.0		18.0	18 .4 3.0		21.0
10	19 .1 18.4					

random number to determine the service time of customer two. Since this random number is zero, customer two departs immediately and we use the sixth random number to determine the service time of customer three. The remaining events shown in Table 1 are determined by following this same procedure.

We now wish to observe the performance of this model when two service facilities are provided. Again using the event oriented simulation procedure, the results shown in Table 2 are obtained. The first three events are the same as in Table 1. However, when customer two arrives at time 3.2, he may go immediately into service at server number two. Thus, the fourth random number in this case is used to determine the service time of customer two, whereas in Table 1 it was used to determine the arrival time of customer three. Many other discrepancies exist between Tables 1 and 2.

The critical point is that the two system configurations are being evaluated under different streams of event occurrences. For example, customer number six, who might be John Doe, arrives at time 14.0 in Table 1 and at time 11.6 in Table 2. Such a result obviates one of the principal advantages of simulation analysis.

Clearly what is required is a unique sequence of random numbers for each random variable in the model. This will permit the generation of identical streams of event occurrences, no matter how the system configuration may be modified.

Converting to Multiple Sequence Generators

It is a relatively simple matter to convert a FORTRAN random number subroutine to a multiple sequence generator. Consider, for example, the following generator:

```
      SUBROUTINE SRAND (IS, R)
      TEMP = IS
      IF (IS) 3,10,3
    3 IS = 0
      ARG = 475918104.
      IF (TEMP - 2500.) 4,4,5
    5 TEMP = TEMP - 2500.
    4 ID = TEMP
   10 IF (ARG) 11,12,11
   12 ARG = 475918104.
      GO TO 14
   11 ARG = ARG*23.
   14 IF (ARG - 1000000010.) 15,16,16
   16 ARG = ARG - 1000000010.
      GO TO 14
   15 IF (ARG - 100000001.) 17,18,18
   18 ARG = ARG - 100000001.
      GO TO 15
   17 IF (ID) 20,20,19
   19 ID = ID - 1
      GO TO 10
   20 R = ARG/100000000.
      RETURN
      END
```

As it is written, this subroutine will generate a single sequence of random numbers. The user specifies an initial burn value (an integer called IS) in the calling program. The appropriate call is as follows:

```
CALL SRAND (IS, R)
```

When called the first time, the generator will "burn" and ignore the first IS random numbers in the sequence. Thereafter, on each call, the next number in the sequence is generated and returned as R. ARG = 475918104 is the initial seed value and 23 in statement 11 is the multiplier in the congruence relation. The generator will not allow more than 2500 values in the sequence to be burned.

The following subroutine shows the same generator, but modified to generate any number of parallel random number sequences:

```
      SUBROUTINE MRAND(IS,R,ARG,N)
      DIMENSION ARG(1)
      TEMP = IS
      IF(IS)3,10,3
  3   IS = 0
      ARG(N) = 475918104.
      IF(TEMP - 2500.)4,4,5
  5   TEMP = TEMP - 2500.
  4   ID = TEMP
 10   IF(ARG(N))11,12,11
 12   ARG(N) = 475918104.
      GO TO 14
 11   ARG(N) = ARG(N)*23.
 14   IF(ARG(N) - 1000000010.)15,16,16
 16   ARG(N) = ARG(N) - 1000000010.
      GO TO 14
 15   IF(ARG(N) - 100000001.)17,18,18
 18   ARG(N) = ARG(N) - 100000001.
      GO TO 15
 17   IF(ID)20,20,19
 19   ID = ID - 1
      GO TO 10
 20   R = ARG(N)/100000000.
      RETURN
      END
```

In this modified generator, the user specifies an array of burn values (called LSEED(n), all integers and each unique) in the calling program. In addition, the array ARG(n) is dimensioned in the calling program. The dimension size, n, of both LSEED and ARG is made equal to the number of unique random number streams desired. The appropriate call is as follows:

CALL MRAND (LSEED (i), R, ARG, i)

where "i" is the particular random number stream desired on this call. The initial portion of each stream is "burned" on the first call to the subroutine for that stream. On subsequent calls to that stream, the modified generator will produce the next number in the i^{th} sequence and return it to the calling program as R. It is the user's responsibility to use the proper sequence designation as he is programming the logic of each random phenomenon.

It is noted that the modified generator presented above actually produces different portions of the same random number sequence. Suppose, for example, that we set LSEED (1) = 5 and LSEED (2) = 10. MRAND would burn the first five numbers of the sequence for random variable 1 and the first ten numbers of the same sequence for random variable 2. Thus, the 6^{th}, 7^{th}, 8^{th}, etc., event occurrences for random variable 1 are governed by the same random numbers as the 1^{st}, 2^{nd}, 3^{rd}, etc., event occurrences for random variable 2. This problem can be overcome by specifying in the calling program an array of multiplier values. The name of this array would be passed through the argument list and used in place of the constant 23 in statement 11 in Subroutine MRAND. In this way, a truly unique sequence is generated for each random variable. The numerical values for the multiplier array must be selected carefully, in accordance with the theory underlying the congruential method. (See Mize and Cox, 1968, or Schmidt and Taylor, 1970).

A more commonly used random number generator is that supplied in the IBM System/360 Scientific Subroutine Package. This subroutine, slightly modified, is listed below:

```
      SUBROUTINE RANDU (IX, YFL)
      IY = IX* 65539
      IF (IY) 5,6,6
  5   IY = IY + 2147483647 + 1
  6   YFL = IY
      YFL = YFL* .4656613E - 9
      IX = IY
      RETURN
```

IX is initialized in the calling program to any odd integer value with nine or fewer digits, and YFL is a uniformly distributed random number on the unit interval $[0,1.0]$.

This generator is easily modified to allow the

generation of any number of parallel random number sequences:

```
      SUBROUTINE MRANDU (IX, YFL, N)
      DIMENSION IX(1)
      IY = IX(N)* 65539
      IF (IY) 5,6,6
5     IY = IY + 2147483647 + 1
6     YFL = IY
      YFL = YFL* .4656613 E - 9
      IX(N) = IY
      RETURN
      END
```

In this modified generator, the user specifies an array of seed values (called IX(n), all odd integers with nine or fewer digits and each unique) in the calling program. The dimension size, n, of IX is made equal to the number of unique random number streams desired. The appropriate call is:

CALL MRANDU (IX(i), YFL, i)

where "i" is the particular random number stream desired on this call.

Most random number generators can be modified in a similar way. Such modifications would be especially useful when using GASP or FORTRAN as the simulation language. GPSS provides the capability of generating up to eight parallel sequences, while SIMSCRIPT permits up to ten sequences.

An Application in GASP*

To illustrate the usefulness of the multiple sequence generator, the results of an application are included. The model was written in GASP (see Pritsker and Kiviat, 1969).

Two types of jobs arrive at a job shop for processing. Interarrival times and servicing times are exponentially distributed with the following means:

	Interarrival Times	Service Times
Type 1	1.25	0.50
Type 2	2.00	0.75

Type two jobs have a higher priority than type one jobs. Newly arrived type two jobs are scheduled ahead of type one jobs unless N or more type one jobs are waiting for processing, where N is an unknown number which we wish to find such that total waiting cost is minimized. Waiting costs for type one jobs and type two jobs are $1.00/minute and $3.00/minute, respectively.

Processing of type one jobs is never interrupted in order to process newly arrived type two jobs. Ten percent of type two jobs fail to pass inspection and are immediately reprocessed. Conceptually, they never move off the machine. A new service time is determined.

A GASP model was constructed for this system and run for varying values of N. Each run of the model was for 1000.0 time units.

The experiment was first conducted using a single sequence random number generator. It was then repeated using a unique random number sequence (produced by Subroutine MRAND) for each random variable in the system. The results for several values of N are plotted in Figures 1 and 2.

In Figure 1, the cost curves are very erratic and do not appear to be converging to any kind of optimum. In Figure 2, both component cost curves and the total cost curve behave very nicely, displaying

*The author gratefully acknowledges the assistance of Glenn C. Dunlap, Arizona State University, in the programming of this example and in the development of the multiple sequence generator, MRAND.

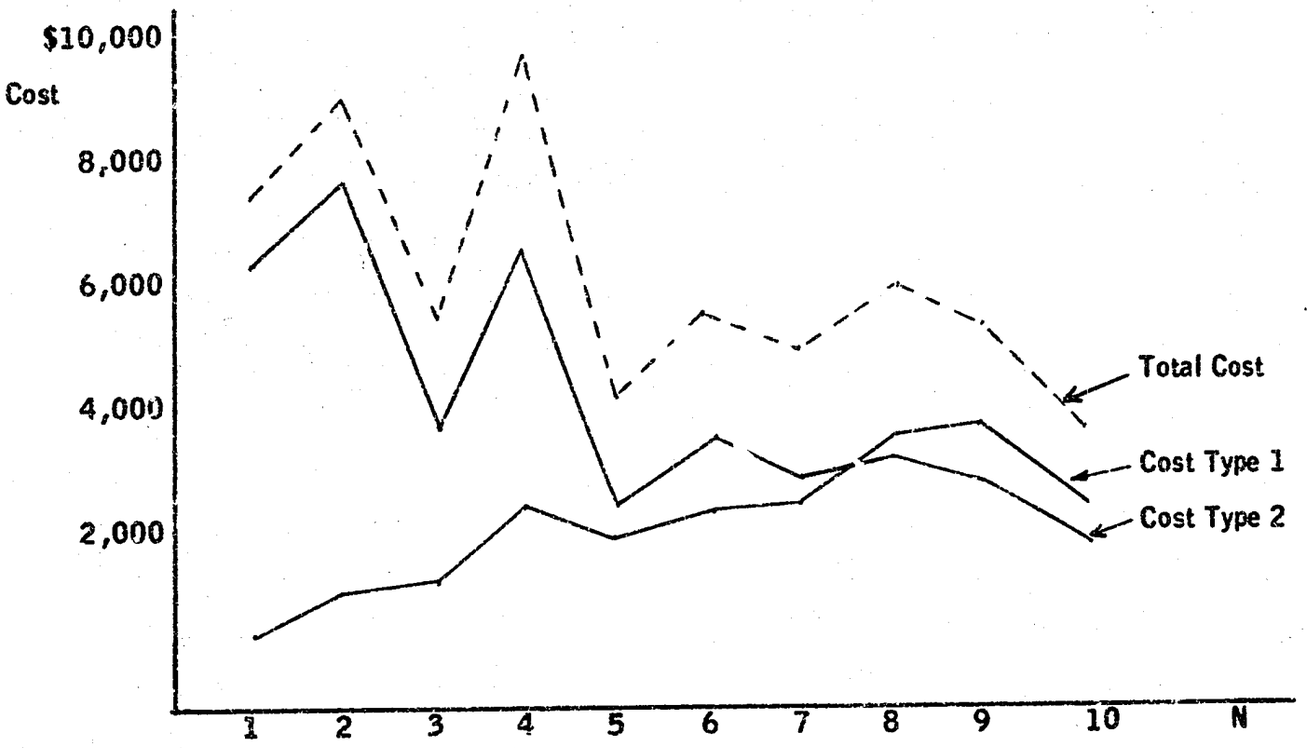

Figure 1. Cost Curves Using One R.N. Sequence

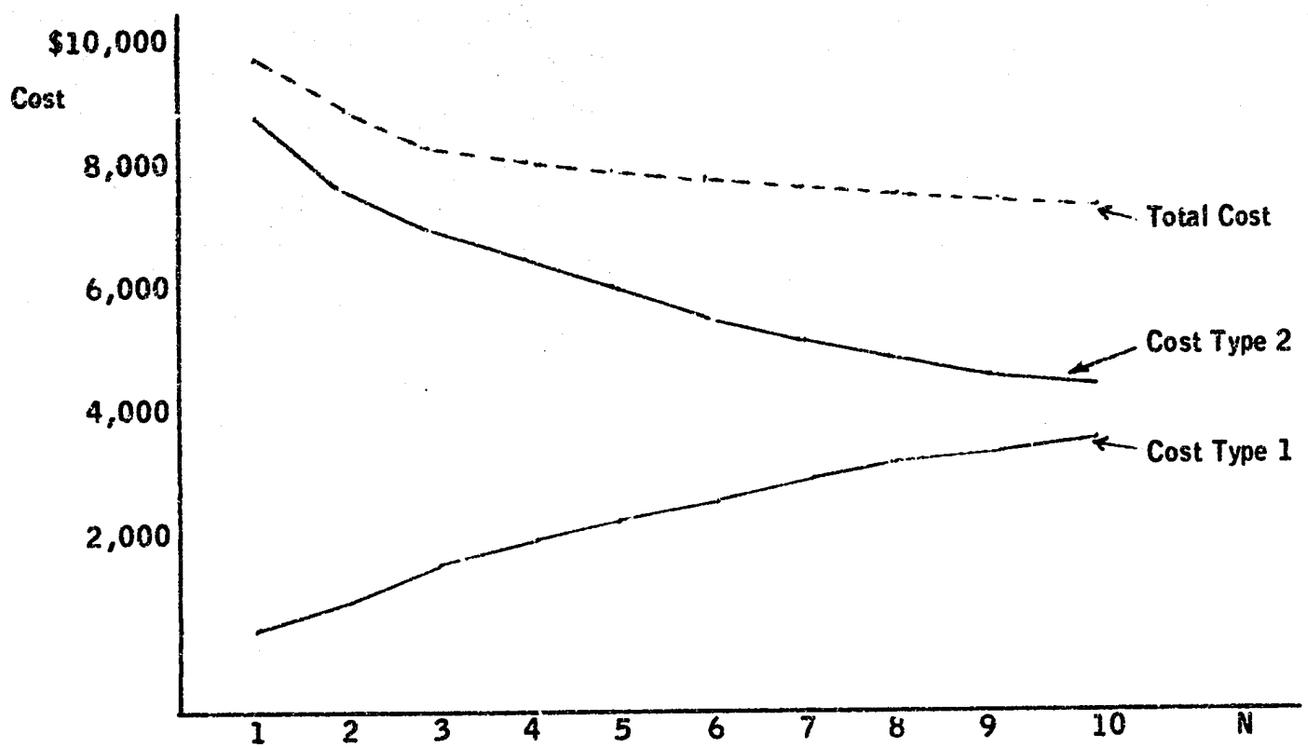

Figure 2. Cost Curves Using Multiple R.N. Sequences

the monotonic characteristics that one would expect. It is clear from Figure 2 that the total cost curve will eventually converge to an optimum value, which it does at N = 32.

Figure 3 shows the Average Number of Units in the System for several values of N. The solid line shows the results obtained using the single sequence random number generator SRAND. The dashed line shows the results obtained using the multiple sequence generator MRAND. The difference in variability is rather striking.

Summary

In many discrete event simulation models, a unique random number sequence is desirable for each random variable in the model. Failure to incorporate this feature may lead to ill-behaved models having a much larger variability than is possible with multiple sequences.

It is very easy to convert most single sequence random number generators to multiple sequence generators. The advantages of doing so are well worth the effort.

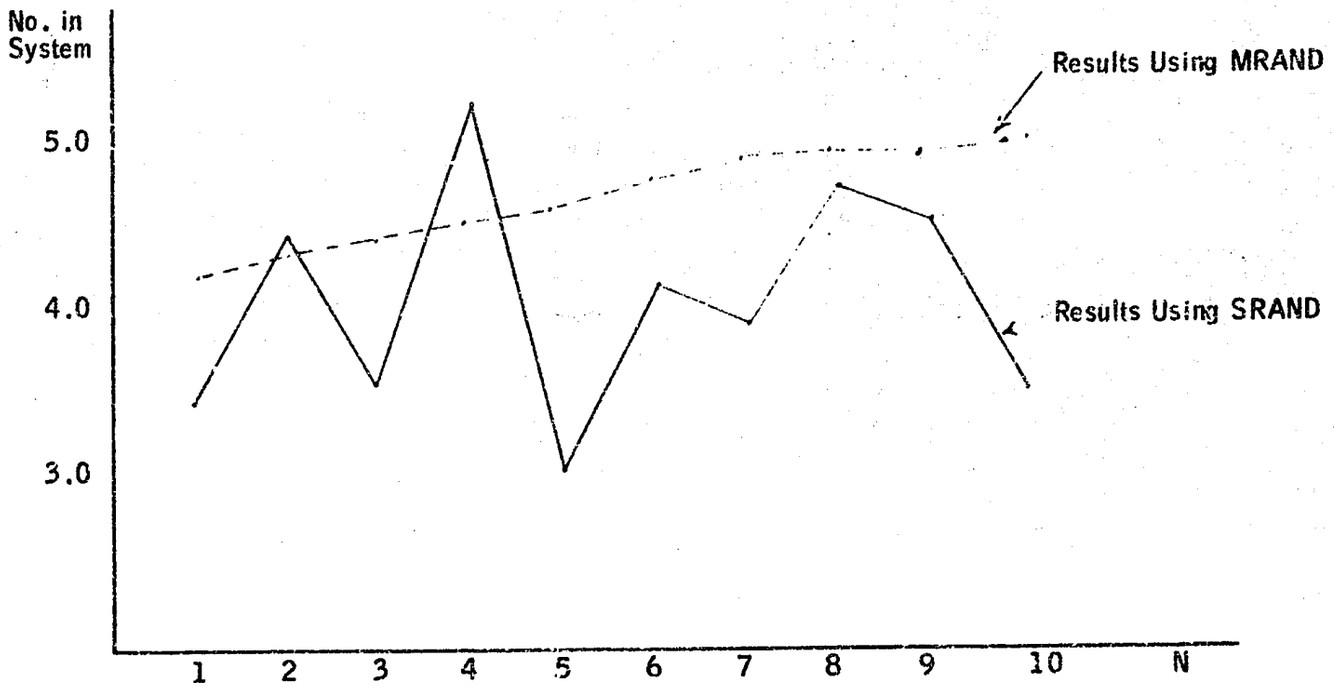

Figure 3. Average Number of Units in System for 100 Hours

References

Gordon, Geoffrey, System Simulation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.

International Business Machine Corporation, System/360 Scientific Subroutine Package (360 A-CM - 03X) Form H20-0205-0. White Plains, New York, 1966.

Kiviat, P.J., R. Villanueva, and H. M. Markowitz, The SIMSCRIPT II Programming Language. The RAND Corporation, Santa Monica, Calif., 1968.

Lehmer, D. H., "Mathematical Method in Large-Scale Computing Units"; Proceedings of the Second Symposium on Large-Scale Digital Computing Machinery. Harvard University Press, Cambridge, Mass., 1959.

Mize, J. H., and J. G. Cox, Essentials of Simulation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1968.

Naylor, T. H., J. L. Balintfy, D. C. Burdick, and K. Chu, Computer Simulation Techniques. John Wiley & Sons, Inc., New York, 1966.

Pritsker, A.A.B., and P. J. Kiviat, Simulation with GASP II. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.

Schmidt, J. W., and R. E. Taylor, Simulation and Analysis of Industrial Systems. Richard D. Irwin, Inc., Homewood, Illinois, 1970.