# AN EMPIRICAL COMPARISON OF RANDOM NUMBER GENERATORS: PERIOD, STRUCTURE, CORRELATION, DENSITY, AND EFFICIENCY <br> <br> DISSERTATION 

 <br> <br> DISSERTATION}

Presented to the Graduate Council of the<br>University of North Texas in Partial<br>Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

## By

Jung Woong Bang, B.S., M.B.A., M.S.<br>Denton, Texas<br>August, 1995

# AN EMPIRICAL COMPARISON OF RANDOM NUMBER GENERATORS: PERIOD, STRUCTURE, CORRELATION, DENSITY, AND EFFICIENCY <br> <br> DISSERTATION 

 <br> <br> DISSERTATION}

Presented to the Graduate Council of the<br>University of North Texas in Partial<br>Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

## By

Jung Woong Bang, B.S., M.B.A., M.S.<br>Denton, Texas<br>August, 1995

Bang, Jung Woong, An empirical comparison of random number generators: period, structure, correlation, density and efficiency. Doctor of Philosophy (Educational Research), August, 1995, 96 pp., 20 tables, 38 illustrations, bibliography, 48 titles. Random number generators (RNGs) are widely used in conducting Monte Carlo simulation studies, which are important in the field of statistics for comparing power, mean differences, or distribution shapes between statistical approaches. Statistical results, however, may differ when different random number generators are used. Often older methods have been blindly used with no understanding of their limitations. Many random functions supplied with computers today have been found to be comparatively unsatisfactory.

In this study, five multiplicative linear congruential generators (MLCGs) were chosen which are provided in the following statistical packages: RANDU (IBM), RNUN (IMSL), RANUNI (SAS), UNIFORM (SPSS), and RANDOM (BMDP). Using a personal computer (PC), an empirical investigation was performed using five criteria: period length before repeating random numbers, distribution shape, correlation between adjacent numbers, density of distributions and normal approach of random number generator (RNG) in a normal function. All RNG FORTRAN programs were rewritten into Pascal which is more efficient language for the PC . Sets of random numbers were generated using different starting values.

A good RNG should have the following properties: a long enough period; a wellstructured pattern in distribution; independence between random number sequences; random and uniform distribution; and a good normal approach in the normal distribution. Findings in this study suggested that the above five criteria need to be examined when conducting a simulation study with large enough sample sizes and various starting values because the RNG selected can affect the statistical results. Furthermore, a study for purposes of indicating reproducibility and validity should indicate the source of the RNG, the type of RNG used, evaluation results of the RNG, and any pertinent information related to the computer used in the study. Recommendations for future research are suggested in the area of other RNGs and methods not used in this study, such as additive, combined, mixed and shifted RNGs.

## TABLE OF CONTENTS

Page
LIST OF TABLES ..... v
LIST OF ILLUSTRATIONS ..... vii
Chapter

1. INTRODUCTION ..... 1
OverviewMonte Carlo Simulation StudyTypes of Random Number Generators (RNG)
Statement of the Problem
Purpose of the Study
Delimitations of the Study
Limitations of the Study
2. REVIEW OF LITERATURE ..... 9
Currently Used Random Number Generators
Criteria for Comparing Random Number Generators
Algorithms for Random Number GeneratorsAlgorithms Not Used in Study
3. METHODS AND PROCEDURES ..... 25
Research Questions
Procedures
Algorithms Used
Random Number Set Generation
4. RESULTS ..... 36Period of the RNG SequenceStructure of the RNG Sequence
Correlation Between RNG Sequences

## TABLE OF CONTENTS--Continued

Density of RNG SequenceEfficiency of the Normal Approach
5. SUMMARY, CONCLUSIONS AND RECOMMENDATIONS ..... 55
SummaryConclusionsRecommendations
APPENDIX A ..... 61
APPENDIX B ..... 77
APPENDIX C ..... 88
BIBLIOGRAPHY ..... 93

## LIST OF TABLES

Table
Page
3-1. Multiplier, Modulus, and Period Values in Selected RNGs ..... 31
3-2. Multiplicative Linear Congruential Generator: $x_{n+1}=\left(a x_{n}+c\right) \operatorname{modm}$ ..... 32
4-1. Period of Random Number Generators ..... 36
4-2. Serial Correlations of $x_{t}$ with $x_{t+h}$ Number Sequence (seed $=1, n=100,000$ ) ..... 43
4-3. Serial Correlations of $x_{t}$ with $x_{t+h}$ Number Sequence (seed $=101, n=100,000$ ) ..... 43
4-4. Chi-square Values for Uniform Distribution (Seed $=1$ ) ..... 45
4-5. Chi-square Values for Uniform Distribution (Seed $=101$ ) ..... 45
4-6. Results of Normal Approach of Means and Standard Deviations (seed=1) ..... 50
4-7. Results of Normal Approach of Means and Standard Deviations (seed=101) ..... 50
4-8. Area Under the Distribution Curve for $n_{1}=1,000, n_{2}=10,000$ or $n_{3}=100,000$ ..... 51
B-1. RANDOM(BMDP): Seeds Apart $1,000,000$ ..... 78
B-2. RNUN(IMSL): Seeds Apart $1,000,000$ ..... 79
B-3. RANUNI(SAS): Seeds Apart 1,000,000 ..... 80
B-4. UNIFORM(SPSS): Seeds Apart 1,000,000 ..... 81
B-5. Chi-square Values: $N=100$ s and $N=1,000 \mathrm{~s}$ ..... 82
B-6. Serial Correlation: RANDU(IBM) ..... 83
B-7. Serial Correlation: RNUN(IMSL) ..... 84
B-8. Serial Correlation: RANUNI(SAS) ..... 85

## LIST OF TABLES--Contimued

Table Page
B-9. Serial Correlation: UNIFORM(SPSS) ..... 86
B-10. Serial Correlation: RANDOM(BMDP) ..... 87

## LIST OF ILLUSTRATIONS

## Figure

## Page

2-1. RANDU - FORTRAN code ..... 19
2-2. RNUN - FORTRAN code ..... 20
2-3. RANUNI - FORTRAN code ..... 21
2-4. UNIFORM - FORTRAN code ..... 21
2-5. RANDOM - FORTRAN code ..... 22
3-1. Plots of pairs $\left(X_{i}, X_{i+1}\right)$ ..... 28
3-2. Plots of triplets $\left(X_{\mathrm{i}}, X_{i+1}, X_{i+2}\right)$ ..... 28
4-1. RANDU: Plots of $\left(X_{b} X_{i+1}\right)$ ..... 38
4-2. RANDU: Plots of $\left(X_{\mathrm{i}}, X_{\mathrm{i}+1}, X_{\mathrm{i}+2}\right)$ ..... 38
4-3. UNIFORM: Plots of $\left(X, X_{i+1}\right)$ ..... 39
4-4. UNIFORM: Plots of $\left(X_{i} X_{i+1}, X_{i+1}\right)$ ..... 39
4-5. RNUN: Plots of $\left(X_{i}, X_{i+1}\right)$ ..... 40
4-6. RNUN: Plots of $\left(X_{i} X_{i+1}, X_{i+2}\right)$ ..... 40
4-7. RANUNI: Plots of $\left(X_{p}, X_{i+1}\right)$ ..... 41
4-8. RANUNI: Plots of $\left(X_{p} X_{i+1}, X_{i+2}\right)$ ..... 41
4-9. RANDOM: Plots of $\left(X_{i}, X_{i+1}\right)$ ..... 41
4-10. RANDOM: Plots of $\left(X_{i} X_{i+1}, X_{i+2}\right)$ ..... 41

## LIST OF ILLUSTRATIONS--Continued

Figure Page
4-11. Normal approach of NRNs from RANDU: $\boldsymbol{n}=1,000$ (seed=1 and seed=101) ..... 47
4-12. Normal approach of NRNs from RANDU: $n=10,000$ (seed $=1$ and seed $=101$ ) ..... 47
4-13. Normal approach of NRNs from RNUN: $n=1,000$
(seed=1 and seed=101) ..... 47
4-14. Normal approach of NRNs from RNUN: $n=10,000$ (seed=1 and seed $=101$ ) ..... 47
4-15. Normal approach of NRNs from RANUNI: $n=1,000$ (seed=1 and seed=101) ..... 48
4-16. Normal approach of NRNs from RANUNI: $n=10,000$ (seed=1 and seed=101) ..... 48
4-17. Normal approach of NRNs from UNIFORM: $n=1,000$ (seed $=1$ and seed $=101$ ) ..... 48
4-18. Normal approach of NRNs from UNIFORM: $n=10,000$ (seed=1 and seed=101) ..... 48
4-19. Normal approach of NRNs from RANDOM: $n=1,000$ (seed=1 and seed=101) ..... 49
4-20. Normal approach of NRNs from RANDOM: $n=10,000$ (seed=1 and seed=101) ..... 49
4-21. Normal approach of means: UNIFORM (seed=1 and seed $=101$ ) ..... 52
4-22. Normal approach of means: RANUNI (seed $=1$ and seed $=101$ ) ..... 52
4-23. Normal approach of standard deviation: UNIFORM (seed $=1$ and seed=101) ..... 52

## LIST OF ILLUSTRATIONS--Continued

Figure
4-24. Normal approach of standard deviation: RANUNI (seed=1 and seed=101) ..... 52
4-25. Normal approach of area under curve within $\pm 1 \sigma$ : RANDOM (Seed=1 and Seed=101) ..... 53
4-26. Normal approach of area under curve within $\pm 2 \sigma$ : RANDOM (Seed=1 and Seed=101) ..... 53
4-27. Normal approach of area under curve within $\pm 3 \sigma$ : RANDOM (Seed=1 and Seed=101) ..... 53
C-1. Find period of random number sequence: RANDU (IBM) (MLCG, $a=65539, m=2147483647$ ) ..... 89
$\mathrm{C}-2$. Find period of random number sequence: RNUN (IMSL) (MLCG; $a=950706376, m=2147483647$ ) ..... 90
C-3. Find period of random number sequence: RANUNI (SAS) (MLCG; $a=397204094, m=2147483647$ ) ..... 91
C-4. Find period of random number sequence: UNIFORM (SPSS) (MLCG; $a=16807, m=2147483647$ ) ..... 92

## CHAPTER 1

## INTRODUCTION

Overview

Statistical results may differ in Monte Carlo simulation studies when different random number generators are used. The different random number generators produce random numbers that are useful in many different kinds of applications: simulation, sampling, numerical analysis, computer programming, decision making, etc. (Knuth 1981). Findings, however, may differ simply because of the type of random number generator used in the application program.

Random numbers are typically generated using a deterministic algorithm that is implemented in the computer, and as such, one is really working with pseudorandom numbers (Niederreiter 1992). These traditional uses of random numbers have coined the name "Monte Carlo method," a general term used to describe any algorithm that employs random numbers (Knuth 1981).

Random number sampling is at the heart of the Monte Carlo method (Niederreiter 1992). Hamilton (1993) reported that many random number generators in use today are not very good. Quite often some old method that is comparatively unsatisfactory has been used blindly, passed down from one programmer to another, and today's users have no understanding of its limitations (Knuth 1981). It has been a widely accepted tradition to use the random number generator supplied by the manufacturer of the computer. The
main reasons for this were probably ease of access, the superior technical expertise of the manufacturer, the belief that any random number generator would do, and the fact that some RNGs exploit particular hardware features of the computer in order to produce, in mysterious ways, code in an unknown language whose source is not available (James 1990).

## Monte Carlo Simulation Study

The Monte Carlo method can be described as a numerical method based on random sampling (Niederreiter 1992). Monte Carlo tests typically compared actual data with simulated data from a supposed model. The similarity of the real and simulated data provided a test of goodness-of-fit (Ripley 1987). The two most important advantages of the Monte Carlo method are: (1) no advanced mathematics are required and (2) realistic simulation methods become possible (Kleijnen and Groenendaal 1992).

## Importance of Random Number Generators

A crucial task in the application of any Monte Carlo method is the generation of appropriate random samples. Generating random numbers uniformly distributed in a specific interval is fundamental to simulation (Bratley, Fox, and Schrage 1987). The success of a Monte Carlo calculation often stands or fails given the "quality" of the random samples that are used, where quality means how well the random samples reflect true randomness (Niederreiter 1992).

Most computers have functions in their program library for producing the required number of random numbers. All practical "random number" generators only produce a
finite sequence which is repeated. These periodic sequences are clearly not random (Bratley, Fox, and Schrage 1987).

## Differences in Random Number Generators

Random number generators for Monte Carlo calculations can be classified according to the type of numbers generated: true random numbers and pseudorandom numbers.

True random numbers are unpredictable in advance and must be produced by a random physical process, such as radioactive decay. True random number series are available on magnetic tape or published in books, but they are extremely cumbersome to use, and are generally insufficient in both number and accuracy for serious calculations (James 1990).

Pseudorandom numbers are produced by the computer through a simple numerical algorithm, and are therefore not truly random, but any given sequence of pseudorandom numbers is supposed to appear random to someone who doesn't know the algorithm (James 1990). Certain desirable properties of random number generators help distinguish the differences in them: good distribution, long period, repeatability (reproducibility), long disjoint subsequences, portability, and efficiency.

There are bad random number generators, especially on microcomputers (Modianos, Scott and Cornwell 1987; Park and Miller 1988). Other generators widely used on medium-sized computers are perhaps not so obviously flawed, but still fail some theoretical and/or empirical statistical tests, and/or generate easily detectable regular
patterns (L'Ecuyer 1988). Sometimes, using a not-so-good generator can give totally misleading results. This may happen only rarely, but can be disastrous (L'Ecuyer 1990).

## Types of Random Number Generators

Some common techniques for generating random numbers are the ten-sided die, throwing a coin, other physical devices (mechanical and electronic devices), random number tables, and pseudorandom numbers.

The disadvantage of a die, coin, or physical device is that the resulting sequence of numbers is not reproducible. Without reproducibility, it is difficult to debug the simulation program; upon adjusting the computer program and feeding in the same numbers, the program should yield similar results. Reproducibility is important because it permits other researchers to repeat the simulation experiment (Kleijnen and Groenendaal 1992).

Various mathematical techniques have been developed for pseudorandom number generation: (1) the midsquare method; (2) the congruential methods; (3) the additive congruential method; and (4) the Tausworthe method or shift-register generators.

## The Midsquare Method

This method was invented by John von Neumann (1952) in the 1940s. Given a starting number $x_{0}$ that consists of $m$ digits, when we square $x_{0}$, we get a number with up to $2 m$ digits. If the squared number has fewer than $2 m$ digits, we add zeros to the front. To obtain the next number, $x_{1}$, the middle $m$ digits of $x_{0}{ }^{2}$ are taken. To get a number in the interval 0 to $1, x_{1}$ is divided by $10^{m}$. If we repeat this procedure, it gives the sequence $x_{i}$ : $i=0,1, \ldots$ (Kleijnen and Groenendaal 1992).

## The Congruential Method

Currently, the congruential method is the most popular. Let $N$ represent the set of natural numbers (nonnegative integers). Let "mod" stand for modulus, so "x mod m" means that $x$ is divided by $m$ and the remainder is taken as a result. Now consider the relation:

$$
\begin{equation*}
n_{i+1}=\left(a n_{i}+b\right) \bmod m \tag{1-a}
\end{equation*}
$$

with $n_{0}, a, b, m \in N ; i=0,1,2, \ldots, m-1$.
The initial number $n_{0}$ is called the seed, $a$ the multiplier, $b$ the additive constant, and $m$ the modulo. The modulo operation, (1-a), means that at most $m$ different numbers can be generated, namely the integers $0,1, \ldots, m-1$. The actual number of different numbers, say $p$, where $p \leq m$, is called the cycle length, or period of the generators. When the additive constant $b$ is zero, the generator is called multiplicative; otherwise, it is called a mixed generator. A congruential generator produces all $\boldsymbol{m}$ different numbers (and thus has maximum cycle length) only if the constants $a, b, m$, and $n_{0}$ meet a number of requirements. The constants $a, b$, and $m$ have important effects on the independence of pseudorandom numbers (Kleijnen and Groenendaal 1992).

## The Additive Congruential Method

The additive congruential method is defined by:

$$
\begin{equation*}
n_{i+1}=\left(n_{i}+n_{i-k}\right) \operatorname{modm} \tag{1-b}
\end{equation*}
$$

This method can yield a cycle longer than $m$, because the pair $\left(n_{i}, n_{i-k}\right)$ must be reproduced; it does not suffice that either $n_{i}$ or $n_{i-k}$ is reproduced. Furthermore, after $m$ cycles, $n_{0}$ and
$n_{0-k}$ are not necessarily equal to $n_{m}$ and $n_{m, k}$. This approach, for $k=1(1-\mathrm{b})$, is called the "Fibonacci method." In practice, the Fibonacci method is no longer applied since applications have been developed that behave better (Marsaglia 1985).

## Tausworthe Generator or Shift-register Generators

The Tausworthe (1965) developed a generator that operates on bits as defined by:

$$
\begin{equation*}
b_{j}=\left(\sum_{j=1}^{q} c_{j} b_{i . j}\right) \bmod 2 \tag{1-c}
\end{equation*}
$$

with $c_{q}=1$ and $c_{j} \in(0,1)$ for $j=1,2, \ldots, q-1$, with at least one $c_{j}=0$. Tausworthe generators are mostly of the simpler form:

$$
\begin{equation*}
b_{i}=\left(b_{i-h}+b_{i-q}\right) \bmod 2 \tag{1-d}
\end{equation*}
$$

with $0<h<q$. The first $q$ bits, $b_{i}$, must be specified which is analogous to specifying the seed for other generators. The maximum period of the bits is set at $2^{q}-1$. An important advantage of the Tausworthe generators is that they are independent of the word size of the computer (Kleijnen and Groenendaal 1992).

## Statement of the Problem

Random number generators are widely used in conducting Monte Carlo simulation studies. Monte Carlo simulation studies are important in the field of statistics for comparing power, mean differences, or distribution shapes between statistical approaches. Results, however, may differ giving different interpretations, depending upon the random number generator used.

Random number sampling is at the heart of the Monte Carlo method. The success of a Monte Carlo calculation depends on the appropriateness of the underlying stochastic model and, to a large extent, on how well random numbers used in the computation simulate the random variable in the model (Niederreiter 1992). Unfortunately, many of the so-called random functions supplied with computers today are far from random, and many simulation studies have been invalidated as a consequence (Ripley 1987). Many random number generators in use today are not very good. Quite often some old method that is comparatively unsatisfactory has been blindly used and passed down from one programmer to another, and today's users have no understanding of its limitations (Knuth 1981).

## Purpose of the Study

The purpose of this study was to examine presently used random number generators on five basic criteria, with and without a widely used adjustment technique. More specifically, the following random number generators: (a) RANDU (IBM); (b) RNUN (IMSL); (c) RANUNI (SAS); (d) UNIFORM (SPSS); and (e) RANDOM (BMDP), were compared on: (1) random number sequence length before repeating numbers; (2) distribution shape; (3) correlation between adjacent numbers; (4) density of distribution; and (5) implementation of random number generator in a normal function.

## Delimitations of the Study

This study examined only well-known multiplicative linear congmential generators which are provided in the following statistical packages: RANDU (IBM), RNUN (IMSL),

RANUNI (SAS), UNIFORM (SPSS), and RANDOM (BMDP).

## Limitations of the Study

Findings in this study were limited to personal computers (PCs) based on the Intel Corporation's 8086 processor and the pseudorandom number generators that are provided for use with these computers. The findings can therefore be generalized to the following commonly used microcomputers: IBM PS2/90 and IBM compatible 486 DX2/50. The operating system under which the random number generator program is executed is limited to MS-DOS or the equivalent, supporting Microsoft software or Borland Turbo Pascal compilers. The programming language used in this study was Borland Turbo Pascal 7.0 (Borland 1992).

## CHAPTER 2

## REVIEW OF LITERATURE

In the past, researchers who needed random numbers in their scientific work would draw balls out of a "well-stirred urn," would roll die, or would deal out cards. Many researchers today still use a table of over 40,000 random digits which was published in 1927. Since 1939, a number of devices have been built to generate random numbers mechanically, and in 1955, the RAND Corporation published a widely used table of a million digits (Knuth 1981; Sobol 1974; RAND 1955).

Shortly after computers were introduced, people began to search for efficient ways to obtain random numbers within computer programs. A table generated by a computer could be used, but this method was of limited utility because of the memory space and input time requirement. The table was also too short and impractical to reproduce calculations exactly a second time when checking out a program (Knuth 1981).

The first algorithm for obtaining pseudorandom numbers was proposed by John von Neumann (1952) in about 1946. It was called the "middle-of-squares" method. His idea was to take the square of previous random numbers and to extract the middle digits (Sobol 1974). This method is unfortunately not suitable because it tends to give too many small numbers (Sobol 1974). It has proved to be a comparatively poor source of random numbers. Using this method, the sequence tends to get into a short cycle of repeating elements (Knuth 1981).

The overwhelming majority of computations currently performed by the Monte Carlo method use pseudorandom numbers. Sequences generated in a deterministic way are usually called pseudorandom or quasirandom sequences. Random numbers generated deterministically on computers have worked quite well in nearly every application, provided that a suitable method has been carefully selected (Knuth 1981).

By far the most popular pseudorandom number generators in use today are special cases of the following scheme, introduced by D.H. Lehmer in 1949. He chose four numbers: $m$ the modulus, $a$ the multiplier, $c$ the increment, and $x_{0}$ the starting value, where $m>0,0 \leq a<m, 0 \leq c<m, 0 \leq x_{0}<m$. The desired sequence of random numbers $\left\{x_{n}\right\}$ is then obtained by setting the following:

$$
\begin{equation*}
x_{n+1}=\left(a \cdot x_{n}+c\right) \text { mod } m, \quad n \geq 0 \tag{2-a}
\end{equation*}
$$

This is called a linear congruential sequence. The congruential sequences always get into a loop, that is, there is ultimately a cycle of numbers that is repeated endlessly. The repeating cycle is called the period. A useful sequence will have a relatively long period (Knuth 1981). When $c=0$, the generator (2-a) is usually referred to as a multiplicative congruential generator. Since a computer can represent a real number with only finite accuracy, a sequence of random fractions, that is, random real number $U_{n}$, shall be generated by integers $x_{n}$ between zero and some number $m$. Thus, the fraction,

$$
\begin{equation*}
U_{n}=x_{n} / m \tag{2-b}
\end{equation*}
$$

will then lie between zero and one. Usually $m$ is the word size of the computer (Knuth 1981).

In recent years, three classes of simple generators have been used the most. These generators are generally known as the multiplicative linear congruential generator (MLCG), the Fibonacci generator (additive generator), and the shift register generator (also known as the Tausworthe generator) (James 1990).

## Currently Used Random Number Generators

A simple generator can be defined as one for which the maximum period is limited by the number of states that can be represented in one computer word. Thus, for the popular 32 -bit computers, simple generators are limited to a period of about $2.2 \times 10^{9}$. The general purpose generators combine two or more simple generators to attain a longer period and better distribution (James 1990). These are described below.

## RANDU

The RANDU general purpose generator was distributed by IBM for use with its System $/ 360$ series computers and has the modulus $m=2^{31}=2147383648$, multiplier $a=$ 65539, and increment $c=0$; as illustrated in the following equation:

$$
\begin{equation*}
x_{n+1}=65539 x_{n} \bmod 2^{31} \tag{2-c}
\end{equation*}
$$

This generator was based on a theoretical expression which showed that this multiplier should produce the smallest possible serial correlation. Unfortunately, it turns out to have catastrophic higher-order correlation, which many users have observed (James 1990).

Many multiplicative linear congruential generators are descendants of the RANDU formula defined by (2-c). This generator was first introduced in the early 1960 s ; its use soon became widespread. The non-prime modulus selected to facilitate the mod operation
and the multiplier, 65539 , which is equal to $2^{16}+3$, was selected primarily because of the simplicity of its binary representation. Research and experience have now made it clear that RANDU represents a flawed generator with no significant redeeming feature. It does not have a full period, and it has some distinctly non-random characteristics. As noted by Park and Miller (1988), Knuth (1981) described it as really horrible. Because of its widespread use at the time, RANDU was commonly found in the literature of the 1960s and early 1970s. The inadequacies of this generator became so well known, however, that it was never recommended in the computer science literature of the 1980s (Park and Miller 1988).

## RNUN

The routine RNUN in the IMSL generators uses the congruential method with modulus $m=2^{31}-1=2147483647$, increment $c=0$, and three different multipliers, namely, $a=16807, a=397204094$, or $a=950706376$. It uses a very simple subroutine for retrieving the current value of the seed so that simulation can be restarted (namely RNGET) to initialize with a fixed seed, or with a clock-generated seed (RNSET), and to shuffle the numbers (RNOPT). The routine RNUN generates uniform numbers between 0 and 1.

Fishman and Moore's study (1986) indicated that the performance of $a=$ 950706376 is best among these three choices, but the choice of 16807 will result in the fastest execution time. If no selection is made explicitly, the routine uses the multiplier $a=$ 16807, which has been used for some time (Lewis, Goodman, and Miller 1969). The
seed of the generator is an integer value between 1 and 2147483646 . If the seed is not initialized, a random seed is obtained from the computer system clock. The generator has a maximal period of $2^{31}-2$ (IMSL 1991).

## RANUNI

The RANUNI function in SAS returns a number generated from the uniform distribution on the interval $(0,1)$ using a prime modulus multiplicative generator with modulus $2^{31}-1$ and multiplier 397204094 (Fishman and Moore 1982). This generator is

$$
\begin{equation*}
x_{n+1}=397204094 x_{n} \bmod \left(2^{31}-1\right) \tag{2-d}
\end{equation*}
$$

The seed is an integer less than $2^{31}-1$. If the seed is $\leq 0$, then the time of day is used to initialize the seed. The generator has a maximal period of $2^{31}-2$ (SAS 1990).

## UNIFORM

The UNIFORM routine, a SPSS pseudorandom number generator, produces a set of random numbers from a uniform distribution with a minimum of 0 and a user-specified maximum with modulus $2^{31}-1$, and multiplier 16807:

$$
\begin{equation*}
x_{n+1}=16807 x_{n} \bmod \left(2^{31}-1\right) \tag{2-e}
\end{equation*}
$$

Uniform numbers are generated using the algorithm of Lewis, Goodman, and Miller (1980). Within a session, the seed value changes each time a random number series is needed in a session. The seed can be any positive integer value up to $2,000,000,000$, which approaches the limit on some computers. With SPSS for Windows, the seed value is up to $999,999,999$. To duplicate the same series of random numbers, the seed should be set before the series is generated for the first time. Since SPSS resets the seed as it
generates a series of random numbers, it is virtually impossible to determine what seed value was used previously, unless the value was specified (SPSS 1990).

## RANDOM

The BMDP random number generator, RANDOM, generates one random number for each case. The generator starts by using a integer between 1 and 30,000 as a seed number. It then generates uniform pseudorandom numbers on the interval from zero to one. BMDP provides a FORTRAN statement in the subroutine BIMEDT. The FORTRAN code used in the uniform random generator is from an algorithm by Wichman and Hill (1982). The algorithm uses three simple multiplicative congruential generators:

$$
\begin{align*}
& x_{n+1}=171 x_{n} \bmod 30269  \tag{2-f}\\
& x_{n+1}=172 x_{n} \bmod 30307  \tag{2-g}\\
& x_{n+1}=170 x_{n} \bmod 30323 \tag{2-h}
\end{align*}
$$

Each uses a prime number for its modulus and a primitive root for its multiplier. The three results are added, and the fractional part is taken (BMDP 1983).

## Criteria for Comparing Random Number Generators

## Period Length (Random Number Sequence Repetition)

Pseudorandom number generators always have a period, after which they begin to generate the same sequence of numbers over again. Traditional pseudorandom number generators are based on a single integer "seed," which means that the period is limited to the number of different states that can be represented in one computer word. Two bits are
usually lost (for positivity and to avoid even integers), so for a 32-bit computer, a simple generator can have a maximum period of $2^{30}$, or about $10^{9}$. James (1990) insisted that although it is easy to achieve this maximum, it is no longer enough for any present day problems in simulation study. Also he suggested that traditional methods can be extended, even on 32-bit computers, to give periods equal to the number of states representable in 60 bits. Some modern methods have periods much longer than $2^{60}$.

Knuth (1981) stated that the period of a generator cannot exceed the size of its bitstate for a computer word. For optimal memory use, it should be close to that size. So, if $b$ bits are required to represent a computer word, the period will be close to $2^{b}$. Maximal period linear congruential generators (LCGs), in scalar or matrix form, as well as Tausworthe generators, inverse non-linear generators, and many kinds of combined generators, have periods equal (or very close) to $2^{b}$ for a $b$-bit state, if the parameters are chosen appropriately (Knuth 1981; L'Ecuyer 1990).

Becuase of fast computers, modern computer simulations are getting increasingly challenging, and require more and more random numbers. Any generator must have a very long period before deserving any further consideration for general use. L'Ecuyer (1992) insists that standard LCGs with modulo near $2^{31}$, which are still recommended in most simulation books, should be discarded because their period is too short and anything less than $2^{50}$ for the period is too low. In fact, with the latest developments in random number generation, there is no reason for not taking a much longer period than that, for example, over $2^{200}$. L'Ecuyer (1992) has stated that no generator should be used for any serious purpose if its period (or a low bound on it) is unknown.

It is well known that generators have a full cycle, generating every integer in [1, $m$ 1] before repeating, if multiplier $a$ is a primitive element modulo $m$, that is, if $a^{j}-1$ is a multiple of $m$ for $i=m-1$, but for no smaller $i$ (Bratley et al., 1987).

## Shape of Distribution (Lattice Structure)

None of the random number generators are truly random in the classical sense. A set of empirical statistical tests can be applied for testing randomness. If the generator passes all the tests, it proves nothing formally, but improves confidence in the simulation results that could be obtained by using that generator. Some "standard" statistical tests for random number generators are described in Dudewicz and Rally (1981) and Knuth (1981). Besides the empirical tests, most generators can also be analyzed theoretically. For example, in some cases computation can be bounded on the serial correlation, bounded on the discrepancy, or characterized by the geometrical behavior of the set of all $t$ dimensional vectors formed by taking $t$ successive values produced by the generator over its full period (L'Ecuyer 1992). Randomness provides a sequence of independent uniform random variables suitable for all reasonable applications. In particular, the uniform random variable passes all the latest tests for randomness and independence (Marsaglia and Zaman 1991).

## Correlation between Random Nnumbers (Serial Correlation)

The correlation for two stochastic variables, say $x$ and $y$, is usually denoted by $\rho$. The well-known relation between the correlation coefficient and the covariance is $\rho=$ $\operatorname{cov}(x, y) / \delta_{x} \delta_{y y}$ where $\operatorname{cov}(x y)=E[\{x-E(x)\}\{y-E(y)\}]$ and $\delta_{x}^{2}=E\{x-E(x)\}^{2}$. Let the symbol
$\gamma$ represent the covariance. If the $i$-th and the $(i+j)$ th pseudorandom numbers in the sequence $\left(r_{i}\right)$ are distributed independently, then

$$
\begin{equation*}
\gamma_{j}=E\left\{\left(r_{i}-0.5\right)\left(r_{i+j}-0.5\right)\right\}=0 \quad \text { for } j>0, \tag{2-i}
\end{equation*}
$$

where $E\left(r_{i}\right)=0.5$ and $E\left(r_{i+j}\right)=0.5$ because $r_{i}$ and $r_{i+j}$ are assumed to be uniformly distributed on $[0,1)$. The ' $\operatorname{lag} j$ ' covariance $\gamma_{j}$ can be estimated through

$$
\begin{equation*}
\gamma_{j}=\{1 /(n-j)\} \sum_{i=1}^{n-j}\left\{\left(r_{i}-0.5\right)\left(r_{i+j}-0.5\right)\right\} \tag{2-j}
\end{equation*}
$$

Kleijnen and Gronendaal (1992) indicated that even if a specific generator passes a number of statistical tests, there is no guarantee that it is a good generator. Park and Miller (1988) demonstrated that constructing a good generator is very difficult.

## Density of Distribution (Uniform Distribution)

In a typical simulation, one needs a large number of random numbers with the proper statistical properties. All the methods to be presented for generating random variates transform uniformly distributed random numbers. Most computer languages have built-in functions for producing random variables uniform over the interval ( 0,1 ). Generators may also rate differently, depending on whether they are implemented in a high-level language or in an assembly language (Bratley, Fox, and Schrage 1987).

## Implementation of Random Number Generator in Normal Function (Efficiency)

Uniform random numbers are often used to generate nonuniform random numbers. The most important nonuniform continuous distribution is the normal distribution with mean 0 and standard deviation 1 , given by the equation:

$$
\begin{equation*}
F(s)=\frac{1}{\sqrt{2 \pi}} \int_{-\infty}^{x} e^{-t^{2} / 2} d t \tag{2-k}
\end{equation*}
$$

Many of the methods for the generation of independent random variables with a given distribution function, $F$, or probability density function (pdf), $f$, were originally suggested by John von Neumann in the early 1950s, and they have been gradually improved upon by others, for example, Marsaglia, Ahrens, and Dieter (Knuth 1981). The best-known "exact" method for the normal distribution is that of Box and Muller developed in 1958 (Ripley 1987).

The rejection method, first suggested by von Neumann (1951), can be used when $f$ is known. In its simplest form the rejection method requires that the f value be bounded and nonzero only on some finite interval (Bratly, Fox, and Schrage 1987). Ripley (1987) recommended some simple methods for normal distributions: Marsaglia's polar method and the ratio-of-uniform method which is supported by others (Knuth 1981; Ripley 1983).

Leva (1992a) introduced an algorithm for a fast normal RNG which modified the ratio-of-uniform deviates method by Kinderman and Monahan. The FORTRAN function, RANDN, returns normally distributed pseudo-RNs with mean of zero and unit standard deviation (Leva 1992b).

## Algorithms for Random Number Generators

The algorithms of the following computer programs were used to generate data for comparative purposes in this study.

## IBM (RANDU)

A FORTRAN code for RANDU, a uniform random number generator, is presented as fig. 2-1 (Bratley, Fox, and Schrage 1987).

## FUNCTION RANDU(IX)

C INPUT: IX, A RANDOM NUMBER.
C $\quad 0<$ IX $<2^{* *} 31-1$
$\mathrm{M}=65539$
C $\mathrm{M}=2^{* *} 16+3$
IX $=\mathrm{M}^{*}$ IX
IF (IX .LT. 0) IX=IX $+2147483647+1$
RANDU=FLOAT(IX)*.4656613E-9
RETURN
END

Fig. 2-1. RANDU-FORTRAN code

Algorithm:
a. Let $I X$ be a large odd integer.
b. Then $I X$ is multiplied by $65539\left(=2^{16}+3\right)$.
c. This yields an integer $\left(\bmod 2^{31}\right.$, still called $\left.I X\right)$.
d. This integer is now turned into a uniform random number (RANDU) by dividing by $2^{31}$ (multiplying by $0.4656613 \times 10^{-9}$ ).

IMSL (RNUN)
A FORTRAN code for RNUN, a uniform random number generator with a double precision in real mode, is presented as fig. 2-2 (IMSL 1991). RNUN is a single precision
in real mode and its FORTRAN code is the same as double precision except that it uses $D M O D$ instead of $M O D$.

FUNCTION DRNUN(IX)
C INPUT: IX, A RANDOM NUMBER.
C $\quad 0<$ IX $<2 * * 31-1$
$\mathrm{M}=950706376 \mathrm{D} 0$
IX=DMOD(M*IX, 2147483647D0)
RNUN=IX/2147483647.0
RETURN
END

Fig. 2-2. RNUN - FORTRAN code

Algorithm:
a. Let $I X$ be a large odd integer.
b. Then $I X$ is multiplied by 950706376 .
c. This yields an integer ( $\bmod 2^{31}-1$ still called $I X$ ).
d. This integer is now turned into a uniform random number (RNUN) by dividing by $2^{31}-1$.

SAS (RANUNI)
A FORTRAN code for RANUNI, a uniform random number generator with a double precision in real mode, is presented as fig. 2-3 (SAS 1990). This generator has a multiplier 397204094 , a modulus $2^{31}-1=2147483647$, and the range of starting value, seed between 0 and $2^{31}-1$.

FUNCTION RANUNI(IX)
C INPUT: IX, A RANDOM NUMBER.
C $0<$ IX $<2 * * 31-1$
M=397204094D0
$\mathrm{IX}=\mathrm{DMOD}\left(\mathrm{M}^{*} \mathrm{IX}, 2147483647 \mathrm{D} 0\right)$
RNUN $=$ IX/2147483647.0
RETURN
END
Fig. 2-3. RANUNI - FORTRAN code

Algorithm:
a. Let $I X$ be a large odd integer.
b. Then $I X$ is multiplied by 397204094 .
c. This yields an integer $\left(\bmod 2^{31}-1\right.$ still called $\left.I X\right)$.
d. This integer is now turned into a uniform random number (RANUNI) by dividing by $2^{31}-1$.

## SPSS (UNIFORM)

A FORTRAN code for UNIFORM, a uniform random number generator with a double precision in real mode, is presented as fig. 2-4 (SPSS 1990).

FUNCTION UNIFORM(IX)
C INPUT: IX, A RANDOM NUMBER.
C $0<$ IX $<2 * * 31-1$
$\mathrm{M}=16807 \mathrm{D} 0$
IX=DMOD(M*IX, 2147483647D0)
UNIFORM=IX/2147483647.0
RETURN
END

Fig. 2-4. UNIFORM - FORTRAN code

The original program introduced by Lewis, Goodman and Miller (1969) was written in assembly language and it was translated into FORTRAN code (Bratley, Fox, and Schrage 1987). This generator has a multiplier 16807, a modulus 2147483647, and the range of starting value seed between 0 and 2147483647.

Algorithm:
a. Let $I X$ be a large odd integer.
b. Then $I X$ is multiplied by 16807 .
c. This yields an integer $\left(\bmod 2^{31}-1\right.$ still called $\left.I X\right)$.
d. This integer is now turned into a uniform random number (UNIFORM) by dividing by $2^{31}-1$.

## BMDP (RANDOM)

RANDOM, a uniform random number generator written in FORTRAN code, was introduced by Wichmann and Hill (1982) and is presented as fig. 2-5. (BMDP 1983, 1992).

FUNCTION RANDOM(IX)
C INPUT: IX, IY, IZ RANDOM NUMBERS.
C $0<$ IX, IY, IZ $<30000$
IX=MOD(171 * IX, 30269)
IY=MOD (172 * IY, 30307)
$\mathrm{IZ}=\mathrm{MOD}(170$ * $\mathrm{IZ}, 30323)$
RANDOM $=$ AMOD(FLOAT(IX)/30269.0+FLOAT(IY)

* $/ 30307.0+$ FLOAT(IZ)/30323.0, 1.0)

RETURN
END

Fig. 2-5. RANDOM - FORTRAN code

Algorithm:
a. Let $I X, I Y, I Z$ be integers.
b. Then $I X$ is multiplied by $171, I Y$ is multiplied by 172 and $I Z$ is multiplied by 170.
c. These yield integers $(\bmod 30269$ called $I X$, mod 30307 called $I Y$, and $\bmod$ 30323 called $I Z$ ),
d. The integers are now turned into a uniform random number (RANDOM) by dividing by 30269 or 30307 or 30323 and adding the results.

Algorithms Not Used in Study

## Lagged-Fibonacci Generators

$F(r, s, \diamond)$ starts with $r$ initial (seed) elements $x_{1}, x_{2}, \ldots, x_{r}$ from some set $X$, then successive elements are generated by the recursion $x_{n}=x_{n-r} \diamond x_{n s}$, where $\diamond$ is some binary operation on the set $X$. It is a generalization of the classical Fibonacci sequence with $X$ the set of integers, $r=2, s=1$, and $\diamond$ the binary operation of addition (Marsaglia and Zaman 1991).

## Subtract-with-Borrow Generator(SWB) for PC

Marsaglia and Zaman (1991) introduced the SWB generators. These are related to lagged-Fibonacci generators. The SWB $\mathrm{x}_{\mathrm{n}}=\left(x_{n-r}-x_{m s}-c\right)$ mod $b$ has period $b^{r}-b^{s}$ if $b^{r}-b^{s}$ +1 is a prime and has $b$ as a primitive root: for example, $b=2^{32}-5=4294967291$ and $\mathrm{r}=$ 43, $s=22$. The principal component of combination generator is the SWB generator $x_{n}=$
$\left(x_{n-22}-x_{n-43}-c\right)$ mod $b$, with $b=2^{32}-5$. With an initial set of seed values $x_{1}, x_{2}, \ldots, x_{43}$, each a 32-bit integer in the inclusive range 0 to $2^{32}-6$, and an initial carry bit $c \in\{0,1\}, m=b^{43}$ $b^{22}+1$ is a prime and $b$ is a primitive root. Thus the period is $m-1=b^{43}-b^{22}$, or about $2^{1376}$ or $10^{414}$.

Algorithm:
a. Form $t=x_{n-22}-x_{n-43}-c$.
b. If $t \geq 0$ put $x_{n}=t$ and $c=0$.
c. if $t<0$ put $x_{n}=t+4294967291$ and $c=1$.
d. Then the new $c$ is ready for forming the next $x$.

Many researchers developed RNGs for fast and portable implementation (Campagner, 1992; Carta 1990; Clark 1985; Marsaglia, Narasimhan, and Zaman 1990; Schrage 1979) with various technical methods (Deng and Chu 1991; Haas 1987; L'Ecuyer, Blouin, and Couture 1993) and studied structures of RNGs (Coveyou and MacPherson 1967; Tezuka and L'Ecuyer 1992; Tezuka, L'Ecuyer, and Coutre 1993). Since all the generators are pseudo-random number generators, one of the tasks for RNG remains to find a single way to generate a uniform and normal random number, and to develop a near-true random number generator.

## CHAPTER 3

# METHODS AND PROCEDURES 

## Research Questions

The research questions for investigating the random number generators (RNG) are:

1. At what sample size does the RNG sequence repeat? (Period)
2. What shape does the distribution of RNG have? (Structure)
3. What are the correlations between adjacent numbers? (Correlation)
4. What is the density of the distribution of random numbers? (Density)
5. When do the random numbers reach a normal distribution? (Efficiency)

## Procedures

A set of random numbers(RNs) were produced using algorithm and Pascal programs which were translated from FORTRAN coding. In the RNGs the seed had a starting value of 1 or 101 for all RNGs: RANDU, RNUN, RANUNI, UNIFORM, and RANDOM. It was assumed that these random numbers were independent and come from a particular specified distribution. This assumption was tested statistically for randomness, correlation, and distribution if the observed numbers did not indicate this assumption.

The first research question examined the sample size at which the RNG sequence repeats itself (period). It was investigated as follows. Every pseudo-RNG used in
computers has a sequence or sequences of random draws called cycles or periods. Once all of the numbers in the cycle have been produced, the numbers repeat in the same sequence. Usually the problem of repeating a sequence in a given study is avoided by having a cycle size which is so large that the user will not use more that a small portion of the numbers in the cycle. Because this is a crucial consideration with any generator, the cycles on the RNGs should be checked (Modianos, Scott, and Comwell 1984).

It is known that the linear congruential method will produce a sequence of numbers of full period $m$, if and only if, the following three conditions are present:

1. The constants $m$ and $c$ are relatively prime (i.e., $g c d(m, c)=1$ ).
2. The constants $m$ and $a$ are selected such that all prime factors of $m$ also divide by $a-1$ (i.e., $a=1 \bmod p$ for each prime factor $p$ of $m$ ).
3. If the constant $m$ is divisible by 4 , then 4 also divides by $a-1(a=1 \bmod 4$ if 4 divides $m$ ).

If $c$ is 0 , this would save some computation time in the generation of pseuorandom numbers such as provided by the multiplicative linear congruential generators RANDU, RNUN, RANUNI, UNIFORM, and RANDOM. However, the sequences generated can not be of full period $m$. They have a maximum period of $m$ - 1 only if $m$ is prime. Then the period is divided by $m-1$ and is $m-1$, if and only if, $a$ is a primitive root, that is, $a \neq 0$ and $a^{\left(m-1 y_{p}\right.} \neq 1 \bmod m$ for each prime factor $p$ of $m$-1 (Ripley 1987).

RNUN, RANUNI, and UNIFORM have modulus $m=2147483647=2^{31}-1$ and $m$ is prime, and $m-1=2147483646=2^{31}-2=2 \cdot 3^{2} \cdot 7 \cdot 11 \cdot 31 \cdot 151 \cdot 331$. UNIFORM has a multiplier $16807\left(=7^{5}\right)$ and 7 is a primitive root, hence so is $7^{5}=16807$. Then UNIFORM has a
period of $m-1=2147483646$. RNUN and RANUNI have the multiplier $950706376=2^{3}$ $\cdot 118838297$ and $397204094=2 \cdot 72 \cdot 4053103$, respectively. Therefore RNUN and RANUNI have a period of $m-1$. RANDU has the shortest period, $1.61 \times 10^{9}$, among generators and RANDOM has the longest period, $9.27 \times 10^{12}$. The nonprime modulus for RANDU can not reach the maximum length of period, $2^{31}-2=2647483646$. RANDOM is not a simple generator but a combined generator with three prime moduli which reach the period of $(30269 \cdot 30307 \cdot 30323) / 3 \approx 9.27 \times 10^{12}$. The period (sample size) at which each RNG repeats the same sequence of numbers will be presented in chapter 4 .

The second research question involving the shape of the distribution of random number generators (structure) was investigated as follows. A program was written to call a generator repeatedly, the resulting values were grouped into pairs that represented points in a unit square, and then they were plotted. It is normally necessary to magnify the image by selecting only those points that fall into some smaller square, and drawing only that region, so as to cover the full plotting surface. The resulting pattern of points will be a lattice pattern produced by the algorithm from the random number generator, where Marsaglia's "planes" can be observed (Hamilton 1993). Marsaglia (1968) pointed out that the $k$-tuples $\left(U_{i}, \ldots, U_{i+k-1}\right)$ will always lie on a finite number of hyperplanes in $[0,1]^{k}$ (Ripley 1987). The pairs, triples, and so forth from most congruential pseudo-RNGs are known to lie in the lattice pattern, and the "uniformity" of these lattices is reflected in the quality of the generators (Ripley 1983).

Pairs $\left(x_{i}, x_{i+1}\right)$ of random numbers were generated from a Pascal program in the range of $0<x_{i} \leq 0.000001,0<x_{i+1}<1$. In this range, 2,136 to 2,147 pairs were selected,
and these pairs were plotted in the rectangle, as shown in fig. 3-1. The specific choice of the smaller square was purely arbitrary, and the same kind of image can be seen anywhere in the unit square, if appropriate magnification is applied.

Also, triples $\left(x_{i}, x_{i+1}, x_{i+2}\right)$ of random numbers were selected in the range of $0<x_{i} \leq$ $0.000001,0<x_{i+1}<1,0<x_{i+2}<1$. In this range, 2,136 to 2,147 points were selected, and these points were plotted in the cube, as shown in fig. 3-2. The lattice structure of the RNGs will be presented in chapter 4.


Fig. 3-1. Plots of pairs $\left(X_{i}, X_{i+1}\right)$


Fig. 3-2. Plots of triplets $\left(X_{i}, X_{i+1}, X_{i+2}\right)$

The third research question involving the correlation between sequences of random numbers was examined as follows. After generating disjoint sequences, Knuth's serial test was applied (Knuth 1981). This test measures the relationship between $x_{t}$ and $x_{t+h}$. It is a correlation coefficient that measures the extent to which they covary. The serial correlation coefficient is given by the equation:

$$
\begin{equation*}
C=\left(N \cdot S U M I-S U M^{2}\right) /\left(N \cdot S U M 2-S U M^{2}\right) \tag{3-a}
\end{equation*}
$$

where $N=$ sample size, $S U M=\operatorname{sum}$ of $\mathrm{x}_{t}, S U M I=\operatorname{sum}$ of $x_{t} \cdot x_{t+h}, S U M 2=x_{t} \cdot x_{r}$. The coefficient $C$ will vary from -1 to +1 , and $C=0$ indicates no relationship and $C= \pm 1$ for perfect relationship.

If a new seed happens to be a number used in one of the preceding runs, then these two runs use the same pseudorandom numbers and become dependent. This dependence violates the assumptions of the statistical analysis techniques which form the basis for the simulation. Therefore, different starting seed values are used for each type of generator.

Starting numbers will be separated by numeric values, $h=1$ to 45 . For a multiplicative generator, the first two seeds, $s_{0}$ and $s_{1}$, will be related by the expression $s_{1}=$ $\left(a^{h} s_{0}\right) \bmod m$. Tables with these $h$ values will be presented in chapter 4.

The fourth research question examined the density of the distribution of random numbers. The density research question is the most commonly cited, used, and the most versatile procedure for evaluating distributional assumptions because it uses a chi-square goodness-of-fit test (Payne 1982). The data were grouped into $k$ intervals and the number of samples in each interval counted. Using these frequency values, a chi-square statistic was calculated which has a chi-square distribution with $k-1$ degrees of freedom.

For a sample size of $N>30$, the following formula (Selby 1975) was used to calculate the chi-square value at the 0.05 level of significance:

$$
\begin{equation*}
\chi^{2}=D \cdot\left(1-A+Z \cdot A^{0.5}\right)^{3} \tag{3-b}
\end{equation*}
$$

where $D=$ degree of freedom, $Z=$ the normal deviate, and $A=2 /(9 D)$. The chi-square values were calculated using the above formula, and the chi-square values and associated
degrees of freedom are presented in chapter 4. Since chi-square tables of significance typically do not include values beyond $D>30$, the above formula was also used to generate chi-square significant values for various $N$ sizes between 100 and 40,000 (see table B-5).

The last research question investigated when the random numbers reached a normal distribution (efficiency), and at what sample size. This was tested by calculating the area mean (average of random numbers generated in a specific interval) and the standard deviation of the random numbers. A number of algorithms that generate the normal distribution of random numbers are available today, and they vary in speed, complexity, and machine space requirements (Leva 1992a). Leva introduced an algorithm for a fast normal RNG, which returns normally distributed pseudo-RNs with zero means and unit standard deviations (Leva 1992b). Given a normal distribution function with mean $(\mu) 0$ and standard deviation ( $\sigma$ ) 1 , the area under the curve of this function is equal to 1 . In each interval of the standard deviation, the area under the normal curve is known: $0.68,0.95$ or 0.99 for $-1<\sigma<1,-2<\sigma<2$, or $-3<\sigma<3$, respectively. Means, standard deviations, and areas under the curve in the normal distribution are calculated and presented in chapter 4.

## Algorithms Used

Each FORTRAN algorithm for the RNG used in the statistical package was rewritten in Pascal. This was necessary to compare the RNGs of each package and to make it feasible to run the programs on a personal computer (see appendix A for Pascal
algorithm and programs). The Pascal programs were checked against the FORTRAN programs to assure that they yielded equivalent results. The same seed values were used when comparing all RNGs in the packages.

The RNGs were compared on various combinations of criteria: modulo( $m$ ), multiplier ( $\alpha$ ), increment ( $c$ ), and length of period ( $p$ ), in the following five statistical packages: RANDU (IBM), RNUN (IMSL), RANUNI (SAS), UNIFORM (SPSS), and RANDOM (BMDP). The $a, m$, and $p$ values used are presented in table 3-1.

TABLE 3-1.
MULTIPLIER, MODULUS, AND PERIOD VALUES IN SELECTED RNGS *

| Generator | Multiplier $(a)$ | Modulus $(m)$ | Period $(p)$ |
| :--- | ---: | ---: | ---: |
| RANDU | 65539 | 2147483648 | 1610612736 |
| RNUN | 950706376 | 2147483647 | 2147483646 |
| RANUNI | 397204094 | 2147483647 | 2147483646 |
| UNIFORM | 16807 | 2147483647 | 2147483646 |
| RANDOM $^{\mathrm{b}}$ | 171 | 30269 | 9272395201440 |
|  | 172 | 30307 |  |
|  | 170 | 30323 |  |

a
The increment, $c$, for all generators was set to 0 .
b
RANDOM is a combined generator; therefore, different criteria for $a$ and $m$ are possible.

The various types of multiplicative linear congruential generators (MLCG) from various sources are presented in table 3-2, in which the RNGs in table 3-1 were included. Most of RNGs are written in FORTRAN, and some old program for RNG code were written in assembly language.

TABLE 3-2
MULTIPLICATIVE LINEAR CONGRUENTIAL GENERATOR:

$$
X_{n+1}=\left(A X_{n}+C\right) \bmod M
$$



The RNGs in table 3-2, found in simulation computer programs, can be modified in various formulae to avoid the limitation of the computer word size, and obtain larger periods in the random number sequences, which yields more speed and more portablility. MLCGs can also have their capabilities expanded by the techniques; combining, shuffling, or shifting methods.

## Random Number Set Generation

Random number set generation for investigating the period of number sequences was performed for each RNG. While generating number sequences, the same seed value was used for RANDU, RNUN, RANUNI and UNIFORM (seed=1), but for RANDOM the seed values were: seed $1=1$, seed $2=1$ and seed $3=1$. The next seed and random numbers were produced from the previous seed, repeating seeds and random numbers were checked if the first values were detected. (Results of RNG are in Chapter 4. Computer programs and sample results are in appendix A and appendix B, tables A-1 through A-4).

For the structure of the RNGs, sets of random numbers with dependent pairs and triplets were generated with seed values equal to 1 and saved into memory. All the generated data were imported and translated to SPSS (Microsoft-Windows version) format to use the graphic function which produced the graphical figures. Using the SPSS graphic function, Scatterplot, two-dimensional and three-dimensional graphics were produced. For more visual effect, the three-dimensional graphic was produced by rotation of various angles of view in windows (see figures in chapter 4).

For the serial correlation test on adjacent numbers, $x_{i}$ and $x_{i+h}$, random number sets were generated by various starting numbers, $h=1$ to 45 , and the size of random numbers in the test was 100,000 . For a multiplicative generator, the first two seeds, $s_{0}$ and $s_{1}$, were related by the expression $s_{1}=\left(a^{h} s_{0}\right)$ mod $m$. Pascal programs for the serial correlation test were written for each generator (Results are in chapter 4, and programs 11 and 12 in appendix A).

In the density test, random number sets were generated by grouping them into 100 and 1,000 cells from $1,000,000$ and $10,000,000$ random numbers. Pascal programs were written to generate the numbers and compute the chi-square statistic. Conventional tables of chi-square values are for degrees of freedom $<30$, but since the degrees of freedom in the chi-square test exceeded 30 , the Pascal programs were written using the Knuth's formula (3-b )and chi-square values were calculated . (see program 13 in appendix A and table B-5).

For testing the normal approach of RNGs, 100 sets of random normal numbers between 1,000 and 100,000 were generated by Leva's random normal generator (Leva 1992a, 1992b). In the generation of random normal numbers, seed $=1$ and seed $=101$ were the starting values for each generator. The numbers were counted by four intervals based on standard deviation ( $\sigma$ ) for estimation of the normal distribution: $-1 \leq \sigma \leq 1,-2 \leq 0 \leq$ $2,-3 \leq \sigma \leq 3$. Also Pascal programs were written for generating the random normal number correlated with the distribution of the normal density function. Each set of numbers were imported in SPSS mode for drawing the estimated normal curve if this curve matched with the curve from the theoretical normal density function. For testing
normality, the mean, standard deviation, and total area under the curve were generated and plotted.

## CHAPTER 4

## RESULTS

The results of this study are presented for the five criteria chosen as research questions using five different random number generator: RANDU, RNUN, RANUNI, UNIFORM and RANDOM.

## Period of the RNG Sequence

The sample size at which the RNG sequence repeats (period) was determined as follows. After generating random numbers for each random number generator, periods were detected in the repeating sequence. Periods for each generator are shown in table 4-1. Initial random numbers and last random numbers near the period within repeating sequences are listed in appendix $C$. Also, the Pascal programs used to generate the random numbers are listed in appendix A .

TABLE 4-1
PERIOD OF RANDOM NUMBER GENERATORS

| Generator | Period (Sample size) |
| :--- | ---: |
| RANDU (IBM) | $1,610,612,736$ |
| RNUN (IMSL) | $2,147,483,646$ |
| RANUNI (SAS) | $2,147,483,646$ |
| UNIFORM (SPSS) | $2,147,483,646$ |
| RANDOM (BMDP) | $9,272,395,201,440$ |

According to the conditions for modulus and multipliers in chapter 3, RNUN, RANUNI, and UNIFORM have prime modulus $m=2^{31}-1$ and also have a multiplier which is a primitive root. Therefore, these three generators have a maximum period of $m-1=2^{31}$ $2=2147483646$. Since RANDU has a nonprime modulus, it can not reach a maximum length of period. RANDU has the shortest period, $1.61 \times 10^{9}$, among the generators. RANDOM is not a simple generator, but a combined generator with three prime moduli which reach the longest period of $(30269 \cdot 30307 \cdot 30323) / 3 \approx 9.27 \times 10^{12}$.

Findings indicated that RNUN, RANUNI and UNIFORM had the same length period, $2.17 \times 10^{9} ;$ RANDU had the shortest period, $1.61 \times 10^{9}$; and RANDOM had the longest period, $9.27 \times 10^{12}$. Therefore, in a simulation study, if the sample size exceeds $2 \times 10^{9}$, then RANDOM should be used. If the sample size is less than $2 \times 10^{9}$, then any RNG can be used, with the exception of RANDU.

## Structure of the RNG Sequence

The shape of the distribution of random number generators (structure) in twodimensional and three-dimensional space was determined as follows.

Pairs $\left(X_{i}, X_{i+1}\right)$ of random numbers and triples $\left(X_{i}, X_{i+1}, X_{i+2}\right)$ of random numbers were generated from a Pascal program in the range of $0<X_{i}<=0.000001,0<X_{i+1}<1$. In this range, 2,136 to 2,147 , points were selected and these points were plotted in the rectangle, as seen in fig. 4-1 through 4-10. The specific choice of the smaller square or cube was purely arbitrary, and the same kind of image can be seen anywhere in the unit square or cube, if appropriate magnification is applied. In fig. 4-1 and fig. 4-2, RANDU
had a very simple and linear structure in both two-dimensional and three-dimensional space, with a range of $0<X_{i}<0.0000001$. The linear tendency was a result of the relatively small multiplier, 65539 .


Fig. 4-1. RANDU: Plots of $\left(X_{i}, X_{i+1}\right)$


Fig. 4-2. RANDU: Plots of $\left(X_{\mathrm{i}}, X_{i+1}, X_{i+2}\right)$

The lattice pattern for RANDU can be computed algebraically. Since $a=65539=2^{16}+3, c=0$ and $m=2147483648=2^{31}$, then:

$$
\begin{aligned}
X_{i+2} & =\left(2^{16}+3\right) X_{i+1}+c_{1} 2^{31} \\
& =\left(2^{16}+3\right) X_{i}+c_{1} 2^{31}\left(2^{16}+3\right)+c_{1} 2^{31} \\
& =\left(6.2^{16}+9\right) X_{i}+\left\{\left(2^{16}+3\right) c_{1}+c_{2}+2 X_{i}\right\} 2^{31} \\
& =6\left(2^{16}+3\right) X_{i}-9 X_{i}+c_{3} 2^{31} \\
& =6 X_{i+1}-9 X_{i}+C_{4} 2^{31}
\end{aligned}
$$

where each $\mathrm{c}_{\mathrm{i}}$ is an integer. Thus $U_{i+2}-6 U_{i+1}+9 U_{i}$ is an integer and $\left(U_{i}, U_{i+1}, U_{i+2}\right)$ lies
on one of 15 planes in the unit cube (Ripley 1987).
In fig. 4-3 and fig. 4-4, UNIFORM also had a very simple and linear structure in both two-dimensional and three-dimensional space. The relatively small multiplier, 16807, caused a monotonic linear tendency in the lattice pattern. In fig. 4-4 a sliced parallel plane containing points can be shown in the cube.


Fig. 4-3. UNIFORM: Plots of $\left(X_{\mathrm{i}} X_{i+1}\right)$


Fig. 4-4. UNIFORM: Plots of $\left(X_{i}, X_{i+1}, X_{i+2}\right)$

The cube represents a very small part of the total space which was magnified by 1,000,000 times in an axis in the total space. In two-dimensional space, UNTFORM had a similar shape to that of RANDU, but in three-dimensional space, UNIFORM had a different shape, a plane, compared to RANDU 's lattice pattern, which was a line.

Lattice structures from RNUN in figs. 4-5 and 4-6 had orderly scattered points over the plane. Two-dimensional space can be covered by a finite number of lines. These
lines can be observed in fig. 4-5, and a finite number of planes can be observed in the cube. Both in two-dimensional and three-dimensional space, horizontal axes were magnified by $1,000,000$ times for visual observation. These figures reflect a uniform distribution of random numbers over the plane.


Fig. 4-5. RNUN: Plots of $\left(X_{i}, X_{i+1}\right)$


Fig.4-6. RNUN: Plots of $\left(X_{i}, X_{i+1}, X_{i+2}\right)$

In figs. 4-7 and 4-8, lattice structures from RANUNI also had orderly scattered points over the plane. As figs. 4-5 and 4-6 show, the shape of RANUNI and RNUN were similar. RNUN and RANUNI have relatively larger multipliers, 950706376 and 397204094, respectively, than RANDU and UNIFORM; thus well-ordered hyperplanes can be observed in the structure. Parallel lines from two-dimensional space and sliced planes from three-dimensional space can be observed in figs. 4-7 and 4-8, respectively.


Fig. 4-7. RANUNI: Plots of $\left(X_{i}, X_{i+1}\right)$

RANDOM had a very disordered scattering of points as noted in figs. 4-9 and
4-10. This lattice pattern indicates a more randomized structure thus producing random numbers with a very large period in its repeating sequence.


Fig. 4-9. RANDOM: Plots of $\left(X_{i}, X_{i+1}\right)$


Fig. 4-10. RANDOM: Plots of $\left(X_{i}, X_{i+1}, X_{i+2}\right)$

In the algorithm, RANDOM is a combined generator with three simple MLCGs. Each generator has a maximum length of period, so period is one third of a multiplication of these three periods.

Based on the lattice pattern in two-dimensional and three-dimensional space, there was evidence that RANDU had a poor structure. UNIFORM had a small multiplier, so it yielded easy and fast computations, but the lattice pattern was relatively poor compared to RNUN, RANUNI, and RANDOM. The combined prime modulus linear generator, RANDOM, had a well-scattered lattice structure thus producing more randomized numbers.

## Correlation Between RNG Sequences

The correlation between sequences of random numbers was determined as follows. All serial correlation coefficients from the five generators with seed $=1$ and seed $=101$ are listed in tables 4-2 and 4-3, respectively. In each sequence, 100,000 random numbers were generated with distance $h=1$ to 40 , but only five distances were presented. The other distances are presented in tables B-6 through B-10. The serial correlations were calculated using Knuth's formula (1982), as shown in chapter 3. The serial correlation programs are in appendix A (see programs 11 and 12 ).

In table 4-2, serial correlations of five RNGs, in which random numbers were generated with the starting value seed $=1$, are presented. None of the RNGs had a significant correlation between adjacent sequences with distance $h$. Sequences generated with seed $=101$ were investigated for correlation between adjacent sequences, and the
results are presented in table 4-3. None of these sequences with seed=101 had a significant correlation with other adjacent ones.

TABLE 4-2
SERIAL CORRELATIONS OF $X_{t}$ WITH $X_{t+h}$ NUMBER SEQUENCE $(\text { SEED }=1, N=100,000)^{a}$

|  | Distance between Sequence $(h)$ |  |  |  |  |  |  |
| :--- | :---: | ---: | ---: | ---: | :---: | :---: | :---: |
| Generator | 1 | 5 | 10 | 15 | 20 | 25 |  |
| RANDU | 0.0008 | 0.0055 | 0.0003 | 0.0031 | -0.0071 | -0.0061 |  |
| RNUN | 0.0066 | -0.0004 | -0.0031 | 0.0023 | 0.0005 | -0.0050 |  |
| RANUNI | -0.0012 | 0.0023 | 0.0035 | 0.0056 | -0.0029 | -0.0030 |  |
| UNIFORM | 0.0024 | -0.0009 | -0.0014 | 0.0023 | -0.0006 | 0.0036 |  |
| RANDOM | 0.0040 | 0.0020 | 0.0022 | -0.0018 | 0.0016 | -0.0002 |  |

Correlation values are reported to the fourth decimal place in the table.

TABLE 4-3
SERIAL CORRELATIONS OF $X_{t}$ WITH $X_{t+h}$ NUMBER SEQUENCE $(\mathrm{SEED}=101, N=100,000)^{2}$

|  | Distance between Sequence $(h)$ |  |  |  |  |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
| Generator | 1 | 5 | 10 | 15 | 20 | 25 |
| RANDU | -0.0058 | -0.0059 | 0.0011 | 0.0023 | -0.0026 | -0.0020 |
| RNUN | -0.0016 | -0.0008 | 0.0013 | 0.0005 | -0.0024 | 0.0006 |
| RANUNI | -0.0056 | -0.0036 | -0.0008 | -0.0008 | 0.0042 | -0.0002 |
| UNIFORM | -0.0024 | 0.0038 | -0.0001 | 0.0022 | -0.0002 | -0.0077 |
| RANDOM | 0.0065 | 0.0001 | 0.0027 | 0.0013 | -0.0002 | 0.0017 |
| a |  |  |  |  |  |  |
| Correlation values are reported to the fourth decimal place in the table. |  |  |  |  |  |  |

Based on serial correlations of random number sequences from five RNGs, all sequences are independent of each other. In a simulation study with a set of sequences, the serial correlation should still be investigated to determine if different seeds will generate different results, because the above tabular results are only the results from two different starting values, seed $=1$ and seed $=101$.

## Density of RNG Sequence

To evaluate the density of the distribution of random numbers, a chi-square goodness-of-fit test was computed (Payne 1982). In a normal distribution with mean 0 and standard deviation 1 , if $p=0.95(\alpha<0.05)$, then $Z=1.645$. Given the degrees of freedom, $d f=99, d f=999$, or $d f=9,999$, then the chi-square values are $123.23,1073.65$, or 10232.8, respectively, from formula (3-a) in chapter 3 (chi-square values for $d f>30$ are in table B-5). Given, $n=1,000,000$ generated numbers divided by 100 cells and 1,000 cells, and $n=5,000,000$ and $n=10,000,000$ generated numbers divided by 1,000 cells and 10,000 cells, then the expected frequency in each cell should be 10,000 or 1,000 , respectively. A chi-square value less than a given criteria indicates a uniform distribution of random numbers in the range of total generated numbers. The chi-square values for the distributions of each generator are listed in table 4-4 for seed=1 and table 4-5 for seed $=101$.

In table 4-4, RANUNI was significant with the generated number $n=5,000,000$ divided by 1,000 cells. Also, RANDOM was significant with the generated number $n=1,000,000$ with 1,000 cells.

TABLE 4-4
CHI-SQUARE VALUES FOR UNIFORM DISTRIBUTION $(S E E D=1)^{2}$

| Generator | $n=1,000,000$ |  | $n=5,000,000$ |  | $n=10,000,000$ |  |
| :--- | ---: | :---: | :---: | :---: | :---: | :---: |
|  | 100 <br> cells | 1,000 <br> cells | 1,000 <br> cells | 10,000 <br> cells | 1,000 <br> cells | 10,000 <br> cells |
|  | 80.04 | 925.15 | 954.51 | 10121.12 | 982.60 | 9911.94 |
| RNUN | 109.54 | 920.62 | 966.49 | 9925.34 | 991.54 | 9804.65 |
| RANUNI | 83.73 | 992.54 | $1103.10^{*}$ | 10160.54 | 1054.22 | 9757.99 |
| UNIFORM | 115.98 | 1053.05 | 975.03 | 9876.76 | 930.81 | 9675.34 |
| RANDOM | 96.46 | $1123.37^{*}$ | 988.15 | 9961.99 | 1003.67 | 9789.38 |
| a |  |  |  |  |  |  |

For chi-square critical values with $p<0.05$ : $d f=99$, chi-square $=123.23$; $d f=999$, chi-square $=1073.65 ; d f=9999$, chi-square $=10232.76$

Significant at $d f=999$, chi-square $>1073.65$.

TABLE 4-5
CHI-SQUARE VALUES FOR UNIFORM DISTRIBUTION (SEED=101) ${ }^{\text {a }}$

| Generator | $n=1,000,000$ |  | $n=5,000,000$ |  | $n=10,000,000$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | $\begin{gathered} 100 \\ \text { cells } \end{gathered}$ | 1,000 cells | $\begin{aligned} & 1,000 \\ & \text { cells } \end{aligned}$ | $\begin{aligned} & 10,000 \\ & \text { cells } \end{aligned}$ | $\begin{aligned} & 1,000 \\ & \text { cells } \end{aligned}$ | $\begin{gathered} 10,000 \\ \text { cells } \end{gathered}$ |
| RANDU | 108.38 | 1031.40 | 1049.26 | 9932.03 | 972.05 | 9663.82 |
| RNUN | 72.94 | 966.52 | 977.19 | 9896.14 | 1007.02 | 9832.75 |
| RANUNI | 93.56 | 1015.74 | 973.66 | 9894.08 | 970.87 | 9765.99 |
| UNIFORM | 82.07 | 1008.62 | 983.86 | 9931.54 | 1019.42 | 10033.51 |
| RANDOM | 104.39 | 1029.65 | 1009.22 | 9939.92 | 959.76 | 9923.83 |
| a |  |  |  |  |  |  |
| For chi-square critical values with $p<0.05$ : $d f=99$, chi-square $=123.23$; $d f=999$, chi-square $=1073.65 ; d f=9999$, chi-square $=10232.76$ |  |  |  |  |  |  |

None of the other generators produced a significant result in the chi-square test when seed $=101$.

Therefore, this finding indicates that different sample sizes and different seed values can have an effect on certain intervals of random numbers, so a selection of random numbers in any generator needs to be investigated for uniformness and randomness when research is performed.

## Efficiency of the Normal Approach

The research question pertaining to when the RNGs reached a normal distribution and at what sample size was also investigated.

For testing the normal approach of RNGs, sets of 1,000 to 100,000 normal random numbers (NRNs) with seed $=1$ and seed $=101$ were generated using Leva's normal random number generator. The numbers were then grouped into four intervals based on the standard deviation of the normal distribution, $-1 \leq \sigma \leq 1,-2 \leq \sigma \leq 2,-3 \leq \sigma \leq 3, \sigma<3$ and $\sigma>3$. Each set of numbers was imported into SPSS and distributions plotted to see if the samples approximated normality. In figs. 4-11 to 4-20, each of the sample sizes for the NRNs, $n=1,000$ and $n=10,000$, from the five generators are presented with two different starting values, seed $=1$ and seed $=101$. The curves from the five generators are plotted with the curve from the normal density function. In figs. 4-21 to 4-27, the means, standard deviations and area under the curve more closely approximated normality as the number of cases increased, surprisingly requiring 100,000 cases for most RNGs.


Fig. 4-11. Normal approach of NRNs from RANDU: $n=1,000$ (seed $=1$ and seed=101)


Fig. 4-13. Normal approach of NRNs from RNUN: $\boldsymbol{n}=1,000$ (seed=1 and seed=101)

Fig. 4-12. Normal approach of NRNs from RANDU: $\boldsymbol{n}=10,000$ (seed=1 and seed=101)


Fig. 4-14. Normal approach of NRNs from RNUN: $n=10,000$ (seed=1 and seed $=101$ )


Fig. 4-15. Normal approach of NRNs from RANUNI: $n=1,000$ (seed=1 and seed=101)


Fig. 4-17. Normal approach of NRNs from UNIFORM: $n=1,000$ (seed $=1$ and seed $=101$ )

Fig. 4-16. Normal approach of NRNs from RANUNI: $n=10,000$ (seed=1 and seed $=101$ )


Fig. 4-18. Normal approach of NRNs from UNIFORM: $n=10,000$ (seed=1 and seed=101)


Fig. 4-19. Normal approach of NRNs from RANDOM: $n=1,000$ (seed $=1$ and seed $=101$ )


Fig. 4-20. Normal approach of NRNs from RANDOM: $n=10,000$ (seed=1 and seed $=101$ )

The figs. 4-11 through 4-20 indicated that when $n=1,000$, the curve is very rough and not well fitted to the normal curve, but when $\boldsymbol{n}=10,000$ the curves closely approach the normal curve. This implies that when sampling from NRN generators, sample size needs to be large enough, and $n>10,000$ is preferred.

In tables 4-6 through 4-8, the sample means and standard deviations departed from the expected normality; mean $=0$, standard deviation $=1$, and total area $=1$. In table 4-8, to estimate the area, $p$, under the normal curve within three intervals, a count of the NRNs in the interval was divided by the total number of NRNs generated. The estimated area should be $0.6826,0.9544$, or 0.9927 in the intervals with $\sigma, 2 \sigma$, or $3 \sigma$, respectively.

Means and standard deviations were calculated after generating either 1,000 , 10,000 or 100,000 random numbers using two different seed values: 1 and 101 .

TABLE 4-6
RESULTS OF NORMAL APPROACH OF MEANS AND STANDARD DEVIATIONS (SEED=1) ${ }^{\text {a }}$

|  | $n=1,000$ |  | $n=10,000$ |  | $n=100,000$ |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: |
| Generator | Mean | StdDev | Mean | StdDev | Mean | StdDev |
| RANDU | 0.0239 | 0.9999 | 0.0141 | 1.0023 | 0.0024 | 1.0035 |
| RNUN | -0.0221 | 0.9842 | -0.0139 | 0.9962 | -0.0016 | 0.9997 |
| RANUNI | 0.0091 | 0.9192 | -0.0031 | 0.9978 | 0.0006 | 1.0018 |
| UNIFORM | -0.0558 | 0.9764 | -0.0104 | 0.9946 | 0.0023 | 0.9975 |
| RANDOM | 0.0239 | 0.9856 | -0.0035 | 0.9924 | 0.0052 | 1.0000 |

a
Expected values in the table are mean $=0$ and standard deviation=1.

TABLE 4-7
RESULTS OF NORMAL APPROACH OF MEANS AND STANDARD DEVIATIONS (SEED=101) ${ }^{\text {a }}$

|  | $n=1,000$ |  | $n=10,000$ |  | $n=100,000$ |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: |
| Generator | Mean | StdDev | Mean | StdDev | Mean | StdDev |
| RANDU | -0.0087 | 1.0068 | -0.0002 | 1.0031 | 0.0024 | 0.9998 |
| RNUN | -0.0350 | 0.9792 | -0.0258 | 1.0063 | 0.0012 | 1.0015 |
| RANUNI | 0.0082 | 0.9803 | 0.0116 | 1.0105 | -0.0019 | 1.0006 |
| UNIFORM | 0.0077 | 1.0144 | 0.0060 | 0.9888 | -0.0001 | 0.9951 |
| RANDOM | 0.0031 | 0.9872 | 0.0025 | 0.9948 | 0.0005 | 0.9980 |
| a |  |  |  |  |  |  |

TABLE 4-8.
AREA UNDER THE DISTRIBUTION CURVE FOR $N_{1}=1,000, N_{2}=10,000$ AND $N_{3}=100,000$

| Generator | $-1<\sigma<1(P=0.6826)$ |  |  | $-2<0<2(P=.9544)$ |  | $-3<\sigma<3(P=0.9974)$ |  |  |  |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | $n_{1}$ | $n_{2}$ | $n_{3}$ | $n_{1}$ | $n_{2}$ | $n_{3}$ | $n_{1}$ | $n_{2}$ | $n_{3}$ |
|  | 0.6860 | 0.6761 | 0.6802 | 0.9550 | 0.9567 | 0.9541 | 0.9970 | 0.9971 | 0.9972 |
| RNUN | 0.6940 | 0.6836 | 0.6821 | 0.9580 | 0.9554 | 0.9553 | 0.9970 | 0.9972 | 0.9973 |
| RANUNI | 0.7290 | 0.6833 | 0.6827 | 0.9650 | 0.9571 | 0.9547 | 0.9980 | 0.9969 | 0.9970 |
| UNIFORM | 0.6940 | 0.6833 | 0.6847 | 0.9620 | 0.9552 | 0.9549 | 0.9980 | 0.9981 | 0.9975 |
| RANDOM | 0.6930 | 0.6876 | 0.6826 | 0.9560 | 0.9563 | 0.9549 | 0.9960 | 0.9973 | 0.9972 |

Based on the above tables and figures, a small number of sample sizes from a random number generator causes unstable means and standard deviations. Sample sizes greater than 10,000 random numbers are recommended.

When seed $=1, n=10,000$ and $n=100,000$, RNUN and RANUNI had good approximations to normality. But when seed $=101$ and $n \geq 10,000$, RANDOM was best. Choosing different seeds results in different means and standard deviations; therefore, for normality purposes, means and standard deviations should be checked before using normal random numbers in an actual study.

The distribution of means and standard deviations from UNIFORM and RANUNI are presented in figs. 4-21 through 4-24. Each distribution from RANDU, RNUN, and RANDOM had a similar graph as UNIFORM and RANUNI. Sample sizes, therefore, need to be large enough to insure a smaller departure from the expected mean $=0$ and standard deviation $=1$. Different RNGs also have different approaches to normality when various seed values and sample sizes are being utilized.

In figs. 4-25 through 4-27, the area under the normal curve within three different intervals using the RANDOM approach is presented. The estimated area is theoretically $0.6826,0.9544$, or 0.9927 in intervals with $\sigma, 2 \sigma$, or $3 \sigma$, respectively.


Fig. 4-21. Normal approach of means: UNIFORM (seed=1 and seed=101)


Fig. 4-23. Normal approach of standard deviation: UNIFORM (seed=1 and seed $=101$ )


Fig. 4-22. Normal approach of means: RANUNI (seed $=1$ and seed=101)


Fig. 4-24. Normal approach of standard deviation: RANUNI (seed=1 and seed $=101$ )



Fig. 4-27. Normal approach of area under curve within $\pm 3 \sigma$ : RANDOM (seed $=1$ and seed $=101$ )

From the figures and tables, it is clear that the normal approach of NRNs from each generator vary with the type of generator, seed value, sample size, and interval of confidence. Therefore, prior investigations of normality are crucial if the results of a research study using random numbers from the RNGs are to be valid and meaningful. Findings indicated that normality will differ based on the size of sample, and the area under the normal curve will differ for each of the three intervals investigated ( $1 \sigma, 2 \sigma$, and $30)$. Departures from the expected normality, mean $=0$, standard deviation $=1$, and area under curve, needs to be reported in any simulation study. Also, a sample size $n>10,000$ is recommended.

## CHAPTER 5

## SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

Summary

In this study, five multiplicative linear congruential generators (MLCG) were chosen in which each random number generator (RNG) had a prime modulus, a multiplier as a primitive root, and an increment 0 . Using a personal computer ( PC ), an empirical investigation was performed involving five criteria: period of generator, random number structure, serial correlation, density in the distribution, and normal approach of normal random numbers (NRN). All RNG FORTRAN programs were rewritten into the Pascal language to facilitate comparison on the PC. Sets of random numbers were generated with two different starting values which were arbitrarily selected. Figures were produced using SPSS for Windows 6.1 (SPSS 1994).

The period of random number sequence is determined based on multiplier (a) and modulus ( $m$ ), and is not affected by increment (c). The size of period from a simple MLCG can not exceed the size of modulus which is limited by the size of the computer word. Most users can not recognize the full period because the period of RNGs is much larger than they actually use for the sampling in a study. After empirical investigation, RNUN, RANUNI, and UNIFORM reached the same period ( $2.17 \times 10^{9}$ ), RANDU had the shortest period ( $1.61 \times 10^{9}$ ), and RANDOM to the longest period $\left(9.27 \times 10^{12}\right)$. Therefore, in a simulation study, if the sample size exceeds $2 \times 10^{9}$, then RANDOM should be used. If
the sample size is less than $2 \times 10^{9}$, then any RNG can be used, with the exception of RANDU.

The structure pattern in two-dimensional and three-dimensional space differs by multiplier and modulus. RANDU had a poor structure and UNIFORM had a structure similar to that of RANDU in two-dimensional space, but a better structure in threedimensional space. UNIFORM had a small multiplier, so it yielded easy and fast computations, but the lattice pattern was relatively poor compared to those of RNUN, RANUNI, and RANDOM. The combined prime modulus linear generator, RANDOM, had a well scattered lattice structure, thus producing more randomized numbers.

The serial correlation in the random number sequences from the five RNGs in this study were not significant, so each sequence was independent of other distanced sequences. In this study, forty-five different distances were investigated with two starting values; however, more extended distances can be investigated using varied starting values and sample sizes. In a simulation study with a set of sequences, the serial correlation should still be investigated to determine if different seeds will generate different results.

The density of random numbers was affected by the multiplier, the starting values, and the sample size used in this study. Significant cases were derived from RNUN and RANDOM, both of which are used widely in the research area. This indicates that a careful investigation is necessary in research even if a generator is well known.

The normality of random number has the most concern in simulation studies. The means, standard deviations, and area under the the curve more closely approximated normality as the number of cases increased, surprisingly requiring 100,000 cases for most

RNGs. The figs. 4-11 through 4-20 indicated that when $n=1,000$, the curve is very rough and not well fitted to the normal curve, but when $n=10,000$ the curves closely approach the normal curve. This implies that when sampling from NRN generators, sample size needs to be large enough, and $n>10,000$ is preferred. Based on the tables and figures, a small number of sample sizes from a random number generator causes unstable means and standard deviations. Sample sizes greater than 10,000 random numbers are recommended.

When seed $=1, n=10,000$ and $n=100,000$, RNUN and RANUNI had good approximations to normality. But when seed $=101$ and $n \geq 10,000$, RANDOM was best. Choosing different seeds results in different means and standard deviations; therefore, for normality purposes, means and standard deviations should be checked before using normal random numbers in an actual study. Different RNGs also have different approaches to normality when various seed values and sample sizes are being utilized.

From the figures and tables, it is clear that the normal approach of NRNs from each generator varies with the type of generator, seed value, sample size, and interval of confidence. Therefore, prior investigations of normality are crucial if the results of a research study using random numbers from the RNGs are to be valid and meaningful. Findings indicated that normality will differ based on the size of sample, and the area under the normal curve will differ for each of the three intervals investigated (10,2 $2 \sigma$, and 30). Departure from the expected normality, mean $=0$, standard deviation $=1$, and area under curve, needs to be reported in any simulation study.

For reliable results, the sample size should be large enough, $n \geq 10,000$, but even a large sample size is not a guarantee of normality in all intervals of the NRN range.

Therefore, an intensive investigation of normality is necessary in any type of study using NRNs.

## Conclusions

When a "Monte Carlo study" is conducted, any type of computer can be used in the study, including a main frame or personal computer (PC) with very different algorithms. A high-speed PC with a floating-point chip affords advanced scientific computing without the traditional headaches of a mainframe, interference from other uses, and support from a large main frame in the computer center (Hamilton 1993). However, when using a PC in a Monte Carlo study, the following criteria should be observed:

1. Period of RNG should be long enough for the simulation.
2. Weil-structured RNG should be chosen to avoid unexpected troubles.
3. Serial correlation should be tested when more than two sequences are used.
4. Uniformity of RNG needs to be investigated.
5. Normality should be checked in statistical decision making, including mean, standard deviation, area mean and distribution in normal curve.
6. Sample size should be large enough; $n>10,000$ is preferred.
7. Various starting values need to be tested for reliable results.

When reporting results from a simulation study, the following information should be provided to the reader for reproducibility and validity:
a) Sources of the RNG: type of RNG, multiplier (a), modulus ( $m$ ), increment (c), period ( $p$ ), and starting value (seed) $\left(X_{0}\right)$.
b) Evaluation results of the RNG: period, structure, correlation, uniformity, and
normality.
c) Computer information: type of computer used, language used, and program code.

## Recommendations

Ripley (1987) suggested that one should choose a generator for which theoretical tests are available and have been performed before it is put to serious use. It is better to use simple and well-understood algorithms. Therefore, a well-tested RNG is recommended because unknown RNGs could cause serious problems. In addition to Ripley's suggestion, any RNG needs to be tested before or after research is performed, because although an RNG has passed certain criteria, there is no guarantee the RNG has validity for the study. As findings in this study, the following recommendations are therefore made:

1. Other MLCGs not used in this study can be tested.
2. Other more high level computer languages can be used in the algorithm, for example, C++.
3. Other types of generators can be used, for example, combined, shifted, and additive generator.
4. Computer word size greater than 32 -bits can be used with Pentium or more powerful PCs with 64-bit chips, or more powerful chips.
5. Various testing techniques for investigating uniformness of RNGs, for example, Run-Up Run-Down, Run-Over Run-Below, and Gap might be utilized.
6. Various nonuniform methods can be used, for example, Box-Muller, Rejection, and Alias.
7. More portable RNGs, for example, SWC-AWC, Tausworthe can be adopted.
8. Generate graphical images and use faster computers; computer clock time greater than 100 Mhz .
9. The testing level can be extended to a multidimensional level.

In current Monte Carlo study, many findings were reported without enough information for the RNG used in research. Therefore, the validity of findings were solely depended on researcher's professionalism. All the generators are pseudorandom number generators, and random numbers can be reproduced if sources of the RNG are provided. Without reporting the information on the RNG, researchers will be unable to replicate other studies and further research their findings.

In this study five well known RNGs were chosen and empirical comparisons were performed on five criteria. Each criteria can be adopted and enforced depending on the characteristics of the study, but the suggestion from Ripley should be considered. Because a simple and well-tested RNG can help researches avoid unexpected troubles.

Other testing methods and criteria can be performed on comparison of RNGs. Different methods yields different outcomes; therefore, the outcomes that past studied might be different from this results. But as a result of findings in this study, RANDOM is the most recommendable generator among five RNGs.

APPENDIX A
COMPUTER PROGRAMS USED IN STUDY

## Program 1 <br> Pascal Function Code: RANDOM (BMDP)

Function RandomR(seed1,seed2,seed3 : longint) :double;
\{* Real version with double precision. *\}
(* RANDOM (BMDP) is a combined RNG with three *\}
(* simple MLCGs. Seed value should be less *\}
(* than 30,000 . Period is $9.27 \times 10^{12}$.
Const
$\mathrm{A} 1=171 ; \quad \mathrm{A} 2=172 ; \quad \mathrm{A} 3=170 ;$
M1 = 30269; M2 = 30307; M3 = 30323;
Var \{ seed < 30,000 \}
nseed1, nseed2, nseed3 : longint;
temp1, temp2, temp3 : 1ongint;
I : Integer;
xr : double;
Begin
temp1 := Al * seed1;
temp2 : = A2 * seed2;
temp3 $:=$ A3 * seed3;
nseed1 := temp1 - M1 * Trunc(temp1 / M1);
nseed2 := temp2 - M2 * Trunc (temp2 / M2);
nseed3 : $=$ temp3 - M3 * Trunc (temp3 / M3);
seed1 : = nseed1; seed2 $:=$ nseed2; seed3 := nseed3;
xr := seed1 / M1 + seed2 / M2 + seed3 / M3;
randomr := xr - trunc(xr);
End;

Program 2
Pascal Function Code: RANDU (IBM)
Function RandomR(seed : longint) :double;
\{* Real version with double precision. *\}
(* RANDU(IBM) is a simple MLCGs. Period is *\}
(* $1.61 \times 10^{9}$.
Const
$a=65539$;
$\mathrm{m}=2147483647$;
Var
temp : longint;
Begin
temp := a * seed;
If temp < 0 then
temp :=1 + (temp +m );
seed := temp;
RandomR := seed*0.4656613E-9 ;
End;

## Program 3 <br> Pascal Function Code: RNUN (IMSL)

```
Function RandomR(seed : longint) :double;
    {* Real version with double precision.
    {* RNUN(IMSL) is a simple MLCGs. Period is *}
    {* 2.72\times10'.
Const
\(a=950706376.0\);
\(\mathrm{m}=2147483647.0\);
Var
I : integer;
temp, nseed : comp;
Begin
temp := a * seed;
nseed := temp - m * Trunc (temp / m);
seed := round(nseed);
RandomR := seed/m;
End;
```

Program 4
Pascal Function Code: RANUNI (SAS)
Function RandomR(seed : longint) : double;
\{* Real version with double precision. *\}
\{* RANUNI (SAS) is a simple MLCGs. Period is *\} \{* $2.72 \times 10^{9}$.

Const
$a=397204094.0$;
$\mathrm{m}=2147483647.0$;
Var
I : integer;
temp, nseed : comp;
Begin
temp := a * seed;
nseed := temp - m * Trunc (temp / m) ;
seed := round (nseed);
RandomR := seed/m;
End;

## Program 5

Pascal Function code: UNIFORM (SPSS)

```
function RandomR(seed : longint) :double;
        {* Real version with double precision. *}
        {* UNIFORM(SPSS) is a simple MLCGs. Period
        {* is 2.72\times10'.
const
            a = 16807.0;
            m = 2147483647.0;
var
            I : integer;
            temp, nseed : comp;
begin
    temp := a * seed;
    nseed := temp - m * Trunc(temp / m);
    seed := round(nseed);
    RandomR := seed/m;
end;
```

Program 6
Generating Random Numbers: RANDOM

```
PROGRAM SDBMD1OX;
    {* BMDP - Real Version 1 - Double Precision * }
    {* Generating 1,000,000 random numbers *)
{$N+}
uses Crt;
const A1 = 171; A2 = 172; A3 = 170;
    M1 = 30269; M2 = 30307; M3 = 30323;
VAR seed1, seed2, seed3 : longint; { seed < 30,0000 }
    nseed1, nseed2, nseed3 : longint;
    temp1, temp2, temp3 : longint;
    K, I : longint;
    RandomB,xr : double;
    outfile : text;
BEGIN
    clrscr;
    assign (Outfile, 'RNUMbmd.out');
    rewrite (Outfile);
    seed1 := 1;
    seed2 := 1;
    seed3 := 1; { < seed < 30,000 }
    K := 1000000;
    writeln(Outfile,'BMDP - Real Version - Double
    Precision');
    writeln(Outfile,' Sequence seed1 seed2 seed3
            Random number');
    For I := 1 to K do
        BEGIN
        temp1 := A1 * seedl;
        temp2 := A2 * seed2;
        temp3 := A3 * seed3;
        nseed1 := temp1 - M1 * Trunc(temp1 / M1);
        nseed2 := temp2 - M2 * Trunc(temp2 / M2);
        nseed3 := temp3 - M3 * Trunc(temp3 / M3);
        xr := nseed1 / M1 + nseed2 / M2 + nseed3 / M3;
        RandomB := xr - trunc(xr);
        writeln(Outfile,I:14, SEED1:7, SEED2:7, SEED3:7,' ',
            RandomB);
        seed1 := nseed1; seed2 := nseed2; seed3 := nseed3;
        END;
    close(outfile)
END.
{$N-}
```


## Program 7

Generating Random Numbers: RANDU

```
PROGRAM SDIBM1OX;
{* RANDU(IBM) - Real Version - Long Integer *}
{* Generating 1,000,000 random numbers
{$N+}
uses Crt;
const
a = 65539;
    mx = 2147483647;
var
    seed, nseed, temp, I, K : longint;
    RandomR : double;
    outfile : text;
BEGIN
        clrscr;
        assign (Outfile, 'RNUMibm.out');
        rewrite (Outfile);
        seed := 1;
        K := 1000000;
        writeln(Outfile,'IBM - Real Version - Long integer');
        writeln(Outfile,'f(seed) = a*seed - m*Trunc(a*seed /
            m)');
        writeln(Outfile,'MLCG; a = 65539, m = 2147483648');
        writeln(outfile);
        writeln(Outfile,' Seq Seed Random Number');
        For I := 1 to K do
        BEGIN
            temp := a * Seed;
            if temp < 0 then
                temp := 1 + (temp + mx );
            nseed := temp;
            RandomR := nseed*0.4656613E-9;
                    { S / 2^31 = S * 0.4656613E-9 }
            writeln(I:12,' ',seed:12,' ',RandomR );
            writeln(Outfile,I:12,' ',seed:12,' ',RandomR );
            seed := round(nseed);
        END;
    close(outfile);
END.
{$N-}
```

Program 8
Generating Random Numbers: RNUN, RANUNI, UNIFORM

```
PROGRAM SDIMS1OX;
    {* RNUN(IMSL) - Real Version - Comp Precision *}
    (* Generating 1,000,000 random numbers
    {* Program for RANUNI(SAS) AND UNIFORM(SPSS) *}
{$N+}
uses Crt;
const
        a = 950706376.0;
            {RANUNI:a=397204094, UNIFORM:a=16807}
            m = 2147483647.0;
var
    seed, I, K : longint;
    RandomR : double;
    temp, nseed : comp;
    outfile : text;
BEGIN
    Clrscr;
    assign (Outfile, 'RNUMims.out');
        rewrite (Outfile);
        seed := 1;
        K := 1000000;
        writeln(outfile);
        writeln(Outfile,' Seq Seed Random
            Number');
        For I := 1 to K do
            BEGIN
            temp := a * seed;
            nseed := temp - m * Trunc(temp / m);
            Randomr := nseed/m;
            writeln(outfile,I:12,' ',seed:12,' ',RandomR);
            seed := round(nseed);
            END;
        close(outfile);
END.
{$N-}
```

Program 9

## Generating Normal Random Numbers: RANDOM

```
PROGRAM NRBMD6;
    {* RANDOM(BMDP) - Real Version - Comp Precision *}
    {* A fast normal random number generator - Leva 1992 *}
    {* Generating random numbers with a normal distribution *}
    {* - Compute Area under curve in intervals, Means,
    {* Standard deviation, Lowest and Highest Number
    {* Generating Numbers N=100, 1000, 10000, 100000,
    {* and 1000000
{$N+}
uses Crt;
Const S = 0.449871; T = -0.386595;
    A = 0.19600; B = 0.25472;
    R1 = 0.27597; R2 =0.27846;
Var U, V, VY, X, Y, Q, V2, U2, RANDN, PS1, PS2, PS3,
    MEAN, STDEV, LOW, HIGH : double;
    seedl, seed2, seed3, CI, K, S1, S2, S3, NI : longint;
function RandomR : double;
    {* Real Version *}
const
    A1 = 171; A2 = 172; A3 = 170;
    M1 = 30269; M2 = 30307; M3 = 30323;
var
    nseed1, nseed2, nseed3 : longint;
    temp1, temp2, temp3 : longint;
    xr : double;
begin
    temp1 := A1 * seed1;
    temp2 := A2 * seed2;
    temp3 := A3 * seed3;
    nseed1 := temp1 - M1 * Trunc(temp1 / M1);
    nseed2 := temp2 - M2 * Trunc(temp2 / M2);
    nseed3 := temp3 - M3 * Trunc(temp3 / M3);
    seed1 := nseed1; seed2 := nseed2; seed3 := nseed3;
    xr := seed1 / M1 + seed2 / M2 + seed3 / M3;
    RandomR := xr - trunc(xr);
end;
BEGIN
    NI := 10;
    clrscr;
    writeln('RANDOM(BMDP) - Normal Random Numbers');
    writeln('MLCG; al=171, a2=172, a3=170, ml=30269,
        m2=30307m, m3=30323');
    writeln('seed = 101');
```

For CI := 1 to 5 do BEGIN
seed1 := 101; seed2 := 101; seed3 $:=101$;
LOW := 100.0;
HIGH $:=-100.0$;
K : = 0;
NI := NI*10;
S1 := 0; S2 := 0; S3 := 0;
REPEAT
BEGIN
$\mathrm{U}:=$ RandpmR;
VY := RandomR;
$\mathrm{V}:=1.7156$ * (VY - 0.5) ;
$\mathrm{X}:=\mathrm{U}-\mathrm{S}$;
$\mathrm{Y}:=\mathrm{ABS}(\mathrm{V})-\mathrm{T}$;
$Q:=X^{*} X+Y^{*}\left(A^{*} Y-B^{*} X\right) ;$
(* Evaluate the quadratic form *)
V2 : = V*V;
U2 : = (-4*U*U) *LN(U);
IF ( $\mathrm{Q}<\mathrm{R} 1$ ) OR ( $(\mathrm{Q}<=\mathrm{R} 2)$ AND (V2 $<=\mathrm{U} 2)$ ) THEN BEGIN \{* Accept $P$ if inside inner ellipse *) RANDN $:=$ Round $(10000 * \mathrm{~V} / \mathrm{U}) / 10000$; MEAN := MEAN + RANDN; STDEV : = STDEV + RANDN*RANDN; If RANDN < LOW then LOW := RANDN; If RANDN > HIGH then HIGH := RANDN; If (RANDN $>=-1$ ) and (RANDN $<=1$ ) then S1 := S1 + 1; If (RANDN $>=-2$ ) and (RANDN $<=2$ ) then

S2 := S2 + 1;
If (RANDN $>=-3$ ) and (RANDN $<=3$ ) then S3 $:=53+1$; $\mathrm{K}:=\mathrm{K}+1$; END;
END;
UNT'IL ( $\mathrm{K}=\mathrm{NI}$ );
PS1 := S1/K; PS2 := S2/K; PS3 := S3/K;
MEAN := MEAN/K;
STDEV := SQRT ((STDEV - (MEAN*MEAN/K))/(K-1));
Writeln('K=', K:8);
Writeln('PS1=',PS1:10:5,', PS2=', PS2:10:5,',
PS3 $=$ ', PS3:10:5);
Write ('MEAN=', MEAN:10:5,', STDEV=', STDEV:10:5); Writeln(', LOW=', LOW:10:5,', HIGH=', HIGH:10:5); Writeln;
END;
END.
$\{\$ \mathrm{~N}-\}$

Program 10
Generating Normal Random Numbers: RNUN, RANUNI, UNIFORM

## PROGRAM NRIMS6;

\{* RNUN(IMSL) RANUNI (SAS) UNIFORM(SPSS) *\}
\{* Real Version with Comp Precision
\{* A fast normal random number generator - Leva 1992 *\}
\{* Generating random numbers with a normal distribution *\}
\{* - Compute Area under curve in intervals, Means, *\}
\{* Standard deviation, Lowest and Highest Number *\}
\{* Generating Numbers N=100, 1000, 10000, 100000, *\}
\{* and 1000000
\{\$N+ \}
uses Crt;
Const
$S=0.449871 ; T=-0.386595 ;$
$\mathrm{A}=0.19600 ; \quad \mathrm{B}=0.25472$;
R1 $=0.27597$; $\mathrm{R} 2=0.27846$;
Var U, V, VY, X, Y, Q, V2, U2, RANDN,
MEAN, STDEV, LOW, HIGH, PS1, PS2, PS3 : double;
seed : comp;
CI, NI, K, S1, S2, S3 : longint;
Function RandomR : double;
const
$a=950706376.0 ; \quad\{R A N U N I: a=397204094$, UNIFORM: $a=16807\}$
$\mathrm{m}=2147483647.0$;
var
temp : comp;
begin
temp := a * seed;
seed := temp - m * Trunc (temp / m);
RandomR := seed/m;
end;
BEGIN
NI := 10;
Clrser;
writeln('Normal Random Numbers - Leva 1992');
writeln;
FOR CI := 1 TO 5 DO
BEGIN
seed := 1.0;
LOW := 100.0;
HIGH := -100.0;
$\mathrm{K}:=0$;
NI := NI * 10;

$$
\begin{aligned}
& \{N=100,1000,10000,100000,1000000\} \\
& \text { S1 }:=0 ; S 2:=0 ; S 3:=0 ;
\end{aligned}
$$

REPEAT

## BEGIN

U := RandomR;
VY := RandomR;
$\mathrm{V}:=1.7156$ * (VY - 0.5) ;
$\mathrm{X}:=\mathrm{U}-\mathrm{S}$;
$\mathrm{Y}:=\mathrm{ABS}(\mathrm{V})-\mathrm{T}$;
$Q:=X^{*} X+Y^{*}\left(A^{*} Y-B^{*} X\right) ;$ \{* Evaluate the quadratic form *\}
V2 : = V*V;
$\mathrm{U} 2:=(-4 * \mathrm{U} * \mathrm{U}) * \mathrm{LN}(\mathrm{U})$;
If $(Q<R 1)$ or ( $(Q<=R 2)$ and (V2 $<=U 2)$ ) then BEGIN \{* Accept $P$ if inside inner ellipse *\} RANDN := V/U; MEAN : = MEAN + RANDN; STDEV := STDEV + RANDN*RANDN; If RANDN < LOW then LOW $:=$ RANDN; If RANDN > HIGH then HIGH $:=$ RANDN; If (RANDN $>=-1$ ) and (RANDN $<=1$ ) then S1 := S1 + 1; If (RANDN $>=-2$ ) and (RANDN $<=2$ ) then $S 2:=S 2+1 ;$ If (RANDN $>=-3$ ) and (RANDN $<=3$ ) then S3 $:=$ S3 +1 ; $\mathrm{K}:=\mathrm{K}+1$; END;
END;
$\operatorname{UNTIL}(\mathrm{K}=\mathrm{NI})$;
PS1 := S1/K; PS2 $:=\mathrm{S} 2 / \mathrm{K}$; PS3 $:=\mathrm{S} 3 / \mathrm{K}$;
MEAN := MEAN/K;
STDEV : = SQRT ((STDEV - (MEAN*MEAN/K))/(K-1));
Writeln('K=', K; 8) ;
Writeln('PS1=',PS1:10:5,', PS2=', PS2:10:5,',
PS3=', PS3:10:5);
Write ('MEAN=', MEAN: 10:5,', STDEV=', STDEV:10:5); Writeln(', LOW=',LOW:10:5,', HIGH=',HIGH:10:5); Writeln;
END;
END.

Program 11
Computing Serial Correlation: RANDOM
(Seed=101, $N=100,000, h=1$ to 40 )

PROGRAM CORRBMD1;
\{* Serial Correlation - RANDOM(BMDP)- Knuth 1981 *\}
\{\$N+\}
uses Crt;
const

$$
\begin{array}{lll}
\mathrm{A} 1=171 ; & \mathrm{A} 2=172 ; & \mathrm{A} 3=170 ; \\
\mathrm{M} 1=30269 ; & \mathrm{M} 2=30307 ; & \mathrm{M} 3=30323 ;
\end{array}
$$

var
seed, seed1, seed2, seed3, nseed1, nseed2, nseed3, seed1n, seed2n, seed3n, temp1, temp2, temp3, I, J, H, N, D : longint;
SUM1, SUM11, SUM12 : comp;
xr, RandomR, CORR : double;
RN : array [1..3] of longint; outfile : text;

## BEGIN

```
Clrscr;
assign (outfile, 'CORRBMD2.out');
rewrite(outfile);
N := 100000; { Total number of cases }
D := 10000; { Integer range }
writeln('Serial Correlation - RANDOM(BMDP) -
    seed=101');
writeln(outfile,'Serial Correlation - RANDOM(BMDP)');
WRITELN(OUTFILE,'N = ',N,', D = ',D);
WRITELN('N = ',N,', D = ',D);
For H := 2 to 41 do { Disjoint distance = H-1 }
BEGIN
seed1 := 101; seed2 := 101; seed3 := 101;
SUM1 := 0;
SUM11 := 0;
SUM12 := 0;
        For I := 1 to N do
            BEGIN
                For J := 1 to H do
                BEGIN
                temp1 := A1 * seed1;
                temp2 := A2 * seed2;
                temp3 := A3 * seed3;
                nseed1 := temp1 - M1 * Trunc(temp1 / M1);
                nseed2 := temp2 - M2 * Trunc(temp2 / M2);
                nseed3 := temp3 - M3 * Trunc(temp3 / M3);
```

```
xr := nseed1 / M1 + nseed2 / M2 + nseed3 / M3;
RandomR := xr - trunc(xr);
seed1 := nseed1; seed2 := nseed2;
seed3 := nseed3;
    IF J = 1 THEN
        BEGIN
        RN[1] := Round(RandomR * D);
        seed1n := nseed1; seed2n := nseed2;
            seed3n := nseed3;
        END;
            END;
        RN[2] := Round(RandomR * D);
        seed1 := seed1n; seed2 := seed2n; seed3 := seed3n;
        SUM1 := SUM1 + RN[1];
        SUM12 := SUM12 + RN[1]*RN[2];
        SUM11 := SUM11 + RN[1]*RN[1];
    Corr := (N*SUM12 - SUM1*SUM1)/(N*SUM11 - SUM1*SUM1);
    Writeln('H= ',H-1:4,', Corr = ',Corr:8:5);
    Writeln(outfile,'H= ',H-1:4,', Corr = ', corr:8:5);
```

END;
END;
close(outfile);
END.
\{ $\$ \mathrm{~N}-$ -

Program 12
Computing Serial Correlation: RNUN, RANUNI, UNIFORM
(Seed=101, $N=100,000, h=1$ to 40 )

```
PROGRAM CORRSPS1;
    {* Serial Correlation - Knuth 1981 }
    {* RNUN(IMSL), RANUNI (SAS) and UNIFORM(SPSS) * }
{$N+}
uses Crt;
const
    a = 16807.0; {RNUN: a=950906376, RANUNI:a=397204094 }
    m = 2147483647.0;
var
    seed, seed1, seed2, I, J, H, N, D : longint;
    temp, nseed, SUM1, SUM11, SUM12 : comp;
    RandomR, Corr : double;
    RN : array [1..3] of longint;
    outfile : text;
```

BEGIN

```
Clrscr;
assign (outfile, 'CORRSPS2.out');
rewrite(outfile);
N := 100000; { Total number of cases }
D := 10000; { Integer range }
writeln{'Serial Correlation - seed=101');
writeln(outfile,'Serial Correlation - seed=101');
WRITELN (OUTFILE, 'N = ',N,', D = ',D);
WRITELN('N = ',N,', D = ',D);
For H := 2 to 41 do { Disjoint distance = H-1 }
BEGIN
seed1 := 101;
SUM1 := 0;
SUM11 := 0;
SUM12 ;= 0;
    For I :=1 to N do
        BEGIN
            For J := 1 to H do
            BEGIN
                seed := seedl;
            temp := a * seedl;
            Nseed := temp - m * Trunc(temp / m);
            RandomR : = nseed/m;
            seedl := Round(nseed);
            IF J = 1 THEN
                BEGIN
                    RN[1] := Round(RandomR * D);
```

```
                    seed2 := round(nseed);
                    END;
                    END;
        RN[2] := Round(RanđomR * D);
        seed1 := seed2;
        SUM1 := SUM1 + RN[1];
        SUM12 := SUM12 + RN[1]*RN[2];
        SUM11 := SUM11 + RN[1]*RN[1];
    END;
    Corr := (N*SUM12 - SUM1*SUM1)/(N*SUM11 - SUM1*SUM1);
    Writeln('H= ',H-1:4,', Corr = ', corr:8:5);
    Writeln(outfile,'H= ',H-1,', Corr = ',Corr:10:5);
    END;
    close(outfile);
END.
{$N- )
```


## Program 13

Computing Chi-square Values

PROGRAM CHISQR;

\{\$N+\}
uses Crt;
VAR
I, N, D : longint;
A, Z, CHI SQR : double;
outfile : text;

## BEGIN

clrscr;
assign (outfile, 'chisqr2.out');
rewrite(outfile);
$\mathrm{N}:=0$;
z : = 1.645;
Writeln(outfile,'Chi-Square Value (95\% level),
$Z=1, Z: 6: 3) ;$
Writeln(outfile);
Writeln(' $N \quad$ Chi-square');
Writeln(outfile, $\quad$ N Chi-square');
For I $:=1$ to 40 do
BEGIN
$\mathrm{N}:=\mathrm{N}+1000$;
$\mathrm{D}:=\mathrm{N}-1$;
A : $=2 /(9 * D)$;
CHI_SQR $:=D^{*}(1-\mathrm{A}+\mathrm{Z} * \operatorname{SQRT}(\mathrm{~A})) *(1-\mathrm{A}+\mathrm{Z} * \mathrm{SQRT}(\mathrm{A}))$

* (1-A+Z* SQRT (A)) ;

Writeln(N:10,CHI_SQR:15:4);
Writeln(outfile, N:10, CHI_SQR:15:4);
END;
close(outfile);
END.
\{ $\$ \mathrm{~N}-$ \}

## APPENDIX B

TABLES

TABLE B-1
RANDOM(BMDP): SEEDS APART 1,000,000 $\left(a_{1}=171, a_{2}=172, a_{3}=170, m_{1}=30269, m_{2}=30307, m_{3}=30323\right.$ )

| Sequence | seed 1 | seed 2 | seed 3 | Random number |
| ---: | ---: | ---: | ---: | ---: |
|  |  |  |  |  |
| 1000000 | 17340 | 1291 | 6299 | $6.00502854167477 \mathrm{E}-0001$ |
| 2000000 | 29089 | 25526 | 19081 | $1.74286494156979 \mathrm{E}-0001$ |
| 3000000 | 19317 | 23598 | 20786 | $5.85751010756602 \mathrm{E}-0001$ |
| 4000000 | 4446 | 24024 | 7783 | $9.32228000716688 \mathrm{E}-0002$ |
| 5000000 | 15408 | 30029 | 23663 | $1.29380062117661 \mathrm{E}-0001$ |
| 6000000 | 29011 | 4903 | 29539 | $3.23564333147914 \mathrm{E}-0001$ |
| 7000000 | 23826 | 2595 | 22181 | $6.82027026156095 \mathrm{E}-0001$ |
| 8000000 | 3406 | 28256 | 13684 | $3.18398930398409 \mathrm{E}-0001$ |
| 9000000 | 14990 | 24944 | 11846 | $6.59563227708267 \mathrm{E}-0001$ |
| 10000000 | 25234 | 18382 | 1267 | $9.81306627104165 \mathrm{E}-0001$ |
| 11000000 | 8163 | 13104 | 29944 | $3.59368770092560 \mathrm{E}-0001$ |
| 12000000 | 13584 | 24645 | 28785 | $9.84891485553345 \mathrm{E}-0001$ |
| 13000000 | 18033 | 27471 | 26882 | $4.88299616335411 \mathrm{E}-0001$ |
| 14000000 | 29775 | 9681 | 3638 | $5.47121743593804 \mathrm{E}-0001$ |
| 15000000 | 28557 | 9902 | 23084 | $9.40667865003844 \mathrm{E}-0001$ |
| 16000000 | 3503 | 16361 | 3027 | $6.12811634947189 \mathrm{E}-0001$ |
| 17000000 | 17532 | 1761 | 25325 | $1.81149087134447 \mathrm{E}-0002$ |
| 18000000 | 6348 | 12658 | 1160 | $2.02748684446365 \mathrm{E}-0001$ |
| 19000000 | 21877 | 2422 | 11428 | $4.05033325295163 \mathrm{E}-0001$ |
| 20000000 | 24102 | 12229 | 22453 | $4.41582629838952 \mathrm{E}-0001$ |
| 21000000 | 29362 | 27322 | 27029 | $4.68233887369051 \mathrm{E}-0001$ |
| 22000000 | 18670 | 19177 | 8955 | $5.12116418702879 \mathrm{E}-0001$ |
| 23000000 | 8086 | 6169 | 28099 | $2.22831605594558 \mathrm{E}-0001$ |
| 24000000 | 16871 | 23002 | 12177 | $1.20318592213532 \mathrm{E}-0001$ |
| 25000000 | 27096 | 1394 | 450 | $5.08775879222400 \mathrm{E}-0001$ |
| 26000000 | 2974 | 15097 | 10707 | $5.07210093045086 \mathrm{E}-0001$ |
| 27000000 | 28321 | 1160 | 7926 | $1.39520148038228 \mathrm{E}-0002$ |
| 28000000 | 19474 | 1127 | 21203 | $2.81837681085270 \mathrm{E}-0001$ |
| 29000000 | 24475 | 7705 | 7395 | $4.54180041759183 \mathrm{E}-0001$ |
| 30000000 | 27591 | 19896 | 27369 | $2.24934059985844 \mathrm{E}-0001$ |
| 31000000 | 3464 | 4281 | 2894 | $8.97495664436041 \mathrm{E}-0002$ |
| 32000000 | 4652 | 25557 | 1743 | $9.50710999588449 \mathrm{E}-0002$ |
| 33000000 | 5828 | 27521 | 15394 | $4.16579368154410 \mathrm{E}-0001$ |
| 34000000 | 21668 | 19919 | 16145 | $9.69226872123706 \mathrm{E}-0001$ |
| 35000000 | 7079 | 19901 | 13192 | $8.93369544636804 \mathrm{E}-0001$ |
| 36000000 | 6396 | 23489 | 25611 | $2.23952351767929 \mathrm{E}-0002$ |
| 37000000 | 23759 | 5342 | 26563 | $4.60281141782605 \mathrm{E}-0001$ |
| 38000000 | 24742 | 16111 | 27786 | $9.86935756848791 \mathrm{E}-0001$ |
| 39000000 | 4007 | 11185 | 2506 | $1.64068123631818 \mathrm{E}-0001$ |
| 40000000 | 7024 | 23277 | 5449 | $3.32698638229973 \mathrm{E}-0001$ |
|  |  |  |  |  |
|  |  |  |  |  |

TABLE B-2
RNUN(IMSL): SEEDS APART 1,000,000 ( $a=950706376, m=2147483647$ )

| Sequence | Seed | Random Number |
| ---: | ---: | ---: |
| 1000000 | 775684152 | $1.24176400771447 \mathrm{E}-0001$ |
| 2000000 | 280916178 | $1.87475218524912 \mathrm{E}-0001$ |
| 3000000 | 1938127747 | $7.28586820293491 \mathrm{E}-0001$ |
| 4000000 | 1841967761 | $6.03972144706162 \mathrm{E}-0001$ |
| 5000000 | 1928484068 | $7.82076497926412 \mathrm{E}-0002$ |
| 6000000 | 1093751850 | $9.23647804615855 \mathrm{E}-0001$ |
| 7000000 | 150562804 | $1.47457269554705 \mathrm{E}-0001$ |
| 8000000 | 194941651 | $7.34317916787378 \mathrm{E}-0001$ |
| 9000000 | 1249026946 | $7.09177075749811 \mathrm{E}-0001$ |
| 10000000 | 1333226850 | $3.31789018740779 \mathrm{E}-0001$ |
| 11000000 | 1384812283 | $5.84853448246072 \mathrm{E}-0001$ |
| 12000000 | 100996335 | $6.64211094223061 \mathrm{E}-0001$ |
| 13000000 | 797591083 | $3.78851262097643 \mathrm{E}-0001$ |
| 14000000 | 2092379687 | $9.51027217298293 \mathrm{E}-0001$ |
| 15000000 | 1243277564 | $2.39568299259789 \mathrm{E}-0001$ |
| 16000000 | 123291309 | $3.49367422680076 \mathrm{E}-0001$ |
| 17000000 | 2143232220 | $1.36669048637463 \mathrm{E}-0001$ |
| 18000000 | 208300439 | $4.92506203470987 \mathrm{E}-0001$ |
| 19000000 | 1705386104 | $3.34695058099318 \mathrm{E}-0001$ |
| 20000000 | 687968084 | $7.72208594145350 \mathrm{E}-0001$ |
| 21000000 | 1109708560 | $1.11533938959024 \mathrm{E}-0002$ |
| 22000000 | 1902810142 | $7.47825884142809 \mathrm{E}-0001$ |
| 23000000 | 1685700422 | $3.27490922216089 \mathrm{E}-0003$ |
| 24000000 | 392707863 | $5.29815041241150 \mathrm{E}-0001$ |
| 25000000 | 2108799714 | $3.41495060520943 \mathrm{E}-0001$ |
| 26000000 | 928520834 | $5.89975548717182 \mathrm{E}-0001$ |
| 27000000 | 445435235 | $7.54899520312855 \mathrm{E}-0001$ |
| 28000000 | 556377682 | $8.54368784397081 \mathrm{E}-0001$ |
| 29000000 | 43638014 | $2.11371650552085 \mathrm{E}-0001$ |
| 30000000 | 1106671343 | $5.49726103222801 \mathrm{E}-0001$ |
| 31000000 | 452352762 | $3.25633103645236 \mathrm{E}-0001$ |
| 32000000 | 1855818541 | $4.66812333309470 \mathrm{E}-0001$ |
| 33000000 | 1946259088 | $4.56046414773933 \mathrm{E}-0001$ |
| 34000000 | 1109300664 | $5.18773249126400 \mathrm{E}-0001$ |
| 35000000 | 1780040499 | $8.24072636116330 \mathrm{E}-0002$ |
| 36000000 | 844454976 | $4.59997975481673 \mathrm{E}-0001$ |
| 37000000 | 1145077827 | $5.28196478042843 \mathrm{E}-0001$ |
| 38000000 | 343703309 | $2.88990436722054 \mathrm{E}-0001$ |
| 39000000 | 1814315277 | $1.01368355611976 \mathrm{E}-0001$ |
| 40000000 | 2066975250 | $9.98624232597008 \mathrm{E}-0001$ |
|  |  |  |

TABLE B-3
RANUNI(SAS): SEEDS APART $1,000,000$ ( $a=397204094, \mathrm{~m}=2147483647$ )

| Sequence | Seed | Random Number |
| ---: | ---: | ---: |
| 1000000 | 775684152 | $1.24176400771447 \mathrm{E}-0001$ |
| 2000000 | 280916178 | $1.87475218524912 \mathrm{E}-0001$ |
| 3000000 | 1938127747 | $7.28586820293491 \mathrm{E}-0001$ |
| 4000000 | 1841967761 | $6.03972144706162 \mathrm{E}-0001$ |
| 5000000 | 1928484068 | $7.82076497926412 \mathrm{E}-0002$ |
| 6000000 | 1093751850 | $9.23647804615855 \mathrm{E}-0001$ |
| 7000000 | 150562804 | $1.47457269554705 \mathrm{E}-0001$ |
| 8000000 | 194941651 | $7.34317916787378 \mathrm{E}-0001$ |
| 9000000 | 1249026946 | $7.09177075749811 \mathrm{E}-0001$ |
| 10000000 | 1333226850 | $3.31789018740779 \mathrm{E}-0001$ |
| 11000000 | 1384812283 | $5.84853448246072 \mathrm{E}-0001$ |
| 12000000 | 100996335 | $6.64211094223061 \mathrm{E}-0001$ |
| 13000000 | 797591083 | $3.78851262097643 \mathrm{E}-0001$ |
| 14000000 | 2092379687 | $9.51027217298293 \mathrm{E}-0001$ |
| 15000000 | 1243277564 | $2.39568299259789 \mathrm{E}-0001$ |
| 16000000 | 123291309 | $3.49367422680076 \mathrm{E}-0001$ |
| 17000000 | 2143232220 | $1.36669048637463 \mathrm{E}-0001$ |
| 1800000 | 208300439 | $4.92506203470987 \mathrm{E}-0001$ |
| 19000000 | 1705386104 | $3.34695058099318 \mathrm{E}-0001$ |
| 20000000 | 687968084 | $7.72208594145350 \mathrm{E}-0001$ |
| 21000000 | 1109708560 | $1.11533938959024 \mathrm{E}-0002$ |
| 22000000 | 1902810142 | $7.47825884142809 \mathrm{E}-0001$ |
| 23000000 | 1685700422 | $3.27490922216089 \mathrm{E}-0003$ |
| 24000000 | 392707863 | $5.29815041241150 \mathrm{E}-0001$ |
| 25000000 | 2108799714 | $3.41495060520943 \mathrm{E}-0001$ |
| 26000000 | 928520834 | $5.89975548717182 \mathrm{E}-0001$ |
| 27000000 | 445435235 | $7.54899520312855 \mathrm{E}-0001$ |
| 2800000 | 556377682 | $8.54368784397081 \mathrm{E}-0001$ |
| 29000000 | 43638014 | $2.11371650552085 \mathrm{E}-0001$ |
| 30000000 | 1106671343 | $5.49726103222801 \mathrm{E}-0001$ |
| 31000000 | 452352762 | $3.25633103645236 \mathrm{E}-0001$ |
| 32000000 | 1855818541 | $4.66812333309470 \mathrm{E}-0001$ |
| 33000000 | 1946259088 | $4.56046414773933 \mathrm{E}-0001$ |
| 34000000 | 1109300664 | $5.18773249126400 \mathrm{E}-0001$ |
| 35000000 | 1780040499 | $8.24072636116330 \mathrm{E}-0002$ |
| 36000000 | 844454976 | $4.59997975481673 \mathrm{E}-0001$ |
| 37000000 | 1145077827 | $5.28196478042843 \mathrm{E}-0001$ |
| 38000000 | 343703309 | $2.88990436722054 \mathrm{E}-0001$ |
| 39000000 | 1814315277 | $1.01368355611976 \mathrm{E}-0001$ |
| 40000000 | 2066975250 | $9.98624232597008 \mathrm{E}-0001$ |
|  |  |  |
|  |  |  |

TABLE B-4
UNIFORM(SPSS): SEEDS APART $1,000,000$
( $a=16807, m=2147483647$ )

| Sequence | Seed | Random Number |
| ---: | ---: | :---: |
| 1000000 | 1531817769 | $5.71498343521496 \mathrm{E}-0001$ |
| 2000000 | 1203219744 | $8.42016775553122 \mathrm{E}-0001$ |
| 3000000 | 1263488938 | $5.30983987511594 \mathrm{E}-0001$ |
| 4000000 | 267863241 | $3.96635092513931 \mathrm{E}-0001$ |
| 5000000 | 1120555121 | $8.78152486345802 \mathrm{E}-0001$ |
| 6000000 | 1022264422 | $6.18847345290169 \mathrm{E}-0001$ |
| 7000000 | 1747963753 | $2.09783069421436 \mathrm{E}-0001$ |
| 8000000 | 805556608 | $5.83473578367137 \mathrm{E}-0001$ |
| 9000000 | 182095227 | $1.44486880928505 \mathrm{E}-0001$ |
| 10000000 | 1274898129 | $8.23525704826939 \mathrm{E}-0001$ |
| 11000000 | 1894411067 | $3.60539484471334 \mathrm{E}-0001$ |
| 12000000 | 1587610707 | $2.27633199760520 \mathrm{E}-0001$ |
| 13000000 | 445582146 | $2.90410003294428 \mathrm{E}-0001$ |
| 14000000 | 733909336 | $8.45466475861830 \mathrm{E}-0001$ |
| 15000000 | 1758704649 | $2.72001342508943 \mathrm{E}-0001$ |
| 16000000 | 1389536132 | $2.28683459678983 \mathrm{E}-0002$ |
| 17000000 | 1505957545 | $1.79836188061087 \mathrm{E}-0001$ |
| 18000000 | 818616541 | $7.95331739725234 \mathrm{E}-0001$ |
| 19000000 | 345307668 | $5.05317879144716 \mathrm{E}-0001$ |
| 20000000 | 573959076 | $1.56685747279174 \mathrm{E}-0002$ |
| 21000000 | 393264705 | $8.34798029081336 \mathrm{E}-0001$ |
| 22000000 | 1015694466 | $1.99945671111320 \mathrm{E}-0001$ |
| 23000000 | 1810221214 | $4.59662009710289 \mathrm{E}-0001$ |
| 24000000 | 447687283 | $7.65989506973880 \mathrm{E}-0001$ |
| 25000000 | 1396162821 | $8.85783427341740 \mathrm{E}-0001$ |
| 26000000 | 1047538754 | $4.25102362607188 \mathrm{E}-0001$ |
| 27000000 | 1472253324 | $3.98157040773964 \mathrm{E}-0001$ |
| 28000000 | 1954243683 | $6.32686057888291 \mathrm{E}-0001$ |
| 29000000 | 1442779532 | $7.25377373269469 \mathrm{E}-0001$ |
| 30000000 | 1298623140 | $5.04222474761411 \mathrm{E}-0001$ |
| 31000000 | 791741049 | $4.57807318986304 \mathrm{E}-0001$ |
| 32000000 | 3159397 | $7.26607559121497 \mathrm{E}-0001$ |
| 33000000 | 326475029 | $1.14130935684839 \mathrm{E}-0001$ |
| 34000000 | 589768177 | $7.43530660282602 \mathrm{E}-0001$ |
| 35000000 | 466416627 | $3.48751637781389 \mathrm{E}-0001$ |
| 36000000 | 1549158706 | $2.88074609957670 \mathrm{E}-0001$ |
| 37000000 | 710609784 | $4.94568944673319 \mathrm{E}-0001$ |
| 38000000 | 958137890 | $7.40928586917430 \mathrm{E}-0001$ |
| 39000000 | 388431 | $4.00044378079495 \mathrm{E}-0002$ |
| 40000000 | 1445411968 | $3.27793561074787 \mathrm{E}-0001$ |
|  |  |  |
|  |  |  |

TABLE B-5
CHI-SQUARE VALUES: $N=100$ s AND $N=1,000$ s

| $N$ | Chi-square | $N$ | Chi-square |
| :---: | :---: | :---: | :---: |
| 100 | 123.2251 | 1000 | 1073.6490 |
| 200 | 232.9135 | 2000 | 2104.1375 |
| 300 | 340.3306 | 3000 | 3127.5268 |
| 400 | 446.5777 | 4000 | 4147.2432 |
| 500 | 552.0789 | 5000 | 5164.6128 |
| 600 | 657.0507 | 6000 | 6180.3157 |
| 700 | 761.6218 | 7000 | 7194.7557 |
| 800 | 865.8758 | 8000 | 8208.1958 |
| 900 | 969.8708 | 9000 | 9220.8190 |
| 1000 | 1073.6490 | 10000 | 10232.7581 |
| 1100 | 1177.2423 | 11000 | 11244.1138 |
| 1200 | 1280.6754 | 12000 | 12254.9638 |
| 1300 | 1383.9680 | 13000 | 13265.3704 |
| 1400 | 1487.1361 | 14000 | 14275.3839 |
| 1500 | 1590.1928 | 15000 | 15285.0456 |
| 1600 | 1693.1492 | 16000 | 16294.3902 |
| 1700 | 1796.0144 | 17000 | 17303.4471 |
| 1800 | 1898.7964 | 18000 | 18312.2412 |
| 1900 | 2001.5021 | 19000 | 19320.7943 |
| 2000 | 2104.1375 | 20000 | 20329.1251 |
| 2100 | 2206.7076 | 21000 | 21337.2501 |
| 2200 | 2309.2173 | 22000 | 22345.1838 |
| 2300 | 2411.6705 | 23000 | 23352.9391 |
| 2400 | 2514.0709 | 24000 | 24360.5276 |
| 2500 | 2616.4217 | 25000 | 25367.9596 |
| 2600 | 2718.7259 | 26000 | 26375.2443 |
| 2700 | 2820.9863 | 27000 | 27382.3903 |
| 2800 | 2923.2051 | 28000 | 28389.4050 |
| 2900 | 3025.3846 | 29000 | 29396.2956 |
| 3000 | 3127.5268 | 30000 | 30403.0684 |
| 3100 | 3229.6336 | 31000 | 31409.7292 |
| 3200 | 3331.7067 | 32000 | 32416.2834 |
| 3300 | 3433.7476 | 33000 | 33422.7360 |
| 3400 | 3535.7578 | 34000 | 34429.0915 |
| 3500 | 3637.7386 | 35000 | 35435.3542 |
| 3600 | 3739.6913 | 36000 | 36441.5280 |
| 3700 | 3841.6171 | 37000 | 37447.61 .67 |
| 3800 | 3943.5170 | 38000 | 38453.6237 |
| 3900 | 4045.3920 | 39000 | 39459.5521 |
| 4000 | 4147.2432 | 40000 | 40465.4050 |

Note: $95 \%$ level, $Z=1.645$

TABLE B-6
SERIAL CORRELATION: RANDU(IBM) APART DISTANCE ( $h$ ), SEED=1 AND 101, $N=100,000$

| $h$ | seed=1 | seed=101 |
| :---: | :---: | :---: |
| 1 | 0.00081 | -0.00580 |
| 2 | 0.00202 | -0.00063 |
| 3 | -0.00749 | -0.00694 |
| 4 | 0.00192 | -0.00008 |
| 5 | 0.00549 | -0.00589 |
| 6 | 0.00136 | -0.00046 |
| 7 | -0.00498 | -0.00017 |
| 8 | -0.00783 | -0.00022 |
| 9 | -0.00277 | 0.00128 |
| 10 | 0.00031 | 0.00114 |
| 11 | 0.00582 | -0.00194 |
| 12 | 0.00430 | -0.00128 |
| 13 | -0.00229 | -0.00263 |
| 1.4 | 0.00349 | 0.00275 |
| 15 | 0.00310 | 0.00225 |
| 16 | -0.00357 | -0.00270 |
| 1.7 | 0.00136 | -0.00087 |
| 18 | -0.00359 | -0.00325 |
| 19 | 0.00479 | 0.00453 |
| 20 | -0.00711 | -0.00256 |
| 21 | -0.00172 | -0.00056 |
| 22 | -0.00177 | -0.00241 |
| 23 | -0.00057 | -0.00088 |
| 24 | -0.00323 | -0.00368 |
| 25 | -0.00605 | -0.00203 |
| 26 | 0.00563 | 0.00610 |
| 27 | 0.00226 | 0.00143 |
| 28 | -0.00590 | 0.00210 |
| 29 | -0.00254 | 0.00440 |
| 30 | 0.00197 | 0.00116 |
| 31 | -0.00294 | -0.00036 |
| 32 | 0.00365 | -0.00795 |
| 33 | 0.00052 | -0.00183 |
| 34 | -0.00353 | -0.00547 |
| 35 | 0.00461 | 0.00037 |
| 36 | -0.00086 | -0.00061 |
| 37 | -0.00197 | 0.00293 |
| 38 | 0.00238 | -0.00410 |
| 39 | -0.00231 | -0.00212 |
| 40 | 0.00397 | -0.00627 |

TABLE B-7
SERIAL CORRELATION: RNUN(IMSL)
APART DISTANCE ( $h$ ), SEED=1 AND 101, $N=100,000$

| $h$ | seed=1 | seed=101 |
| :---: | :---: | :---: |
| 1 | 0.00657 | -0.00160 |
| 2 | 0.00663 | -0.00285 |
| 3 | -0.00115 | -0.00006 |
| 4 | -0.00568 | -0.00411 |
| 5 | -0.00038 | -0.00077 |
| 6 | 0.00401 | 0.00673 |
| 7 | -0.00258 | -0.00136 |
| 8 | -0.00335 | -0.00173 |
| 9 | 0.00132 | -0.00123 |
| 10 | -0.00305 | 0.00134 |
| 11 | -0.00242 | -0.00162 |
| 12 | 0.00223 | 0.00710 |
| 13 | 0.00058 | -0.00214 |
| 14 | -0.00645 | -0.00254 |
| 15 | 0.00228 | 0.00045 |
| 16 | -0.00063 | -0.00439 |
| 17 | 0.00127 | 0.00737 |
| 18 | -0.00137 | -0.00094 |
| 19 | -0.00443 | 0.00077 |
| 20 | 0.00053 | -0.00238 |
| 21 | 0.00693 | 0.00338 |
| 22 | 0.00676 | -0.00055 |
| 23 | -0.00333 | 0.00148 |
| 24 | -0.00271 | 0.00018 |
| 25 | -0.00503 | 0.00057 |
| 26 | -0.00133 | 0.00028 |
| 27 | 0.00163 | -0.00345 |
| 28 | 0.00210 | -0.00017 |
| 29 | -0.00242 | 0.00310 |
| 30 | 0.00152 | 0.00311 |
| 31 | -0.00630 | 0.00185 |
| 32 | 0.00122 | 0.00530 |
| 33 | 0.00337 | 0.00031 |
| 34 | 0.00163 | 0.00226 |
| 35 | -0.00433 | -0.00307 |
| 36 | -0.00546 | 0.00032 |
| 37 | -0.00536 | -0.00064 |
| 38 | -0.00035 | -0.00035 |
| 39 | -0.00077 | -0.00057 |
| 40 | -0.00005 | -0.00329 |

TABLE B-8
SERIAL CORRELATION: RANUNI(SAS)
APART DISTANCE ( $h$ ), SEED=1 AND 101, $N=100,000$

| $h$ | seed=1 | seed=101 |
| :---: | :---: | :---: |
| 1 | -0.00123 | -0.00558 |
| 2 | 0.00310 | 0.00553 |
| 3 | -0.00360 | -0.00189 |
| 4 | -0.00346 | -0.00207 |
| 5 | 0.00231 | -0.00360 |
| 6 | -0.00102 | -0.00088 |
| 7 | 0.00409 | 0.00054 |
| 8 | 0.00053 | -0.00435 |
| 9 | -0.00759 | 0.00387 |
| 10 | 0.00352 | -0.00084 |
| 11 | 0.00002 | -0.00442 |
| 12 | -0.00250 | -0.00231 |
| 13 | 0.00017 | -0.00112 |
| 14 | 0.00017 | -0.00248 |
| 15 | 0.00560 | -0.00083 |
| 16 | -0.00337 | 0.00487 |
| 17 | -0.00210 | -0.00203 |
| 18 | 0.00401 | 0.00385 |
| 19 | 0.00230 | -0.00026 |
| 20 | -0.00291 | 0.00418 |
| 21 | 0.00329 | 0.00237 |
| 22 | 0.00452 | 0.00140 |
| 23 | 0.00521 | 0.00336 |
| 24 | 0.00392 | 0.00321 |
| 25 | -0.00299 | -0.00020 |
| 26 | -0.00148 | 0.00001 |
| 27 | 0.00225 | 0.00328 |
| 28 | 0.00288 | 0.00371 |
| 29 | 0.00612 | 0.00235 |
| 30 | 0.00346 | 0.00015 |
| 31 | -0.00120 | -0.00587 |
| 32 | -0.00311 | 0.00028 |
| 33 | -0.00116 | -0.00195 |
| 34 | -0.00209 | -0.00381 |
| 35 | -0.00099 | -0.00080 |
| 36 | 0.00436 | -0.00372 |
| 37 | -0.00222 | -0.00587 |
| 38 | 0.00060 | 0.00951 |
| 39 | 0.00085 | 0.00222 |
| 40 | 0.00202 | 0.00164 |

TABLE B-9
SERIAL CORRELATION: UNIFORM(SPSS) APART DISTANCE ( $h$ ), SEED $=1$ AND 101, $N=100,000$

| $h$ | seed=1 | seed=101 |
| :---: | :---: | :---: |
| 1 | 0.00240 | -0.00235 |
| 2 | -0.00271 | 0.00078 |
| 3 | 0.00347 | 0.00289 |
| 4 | 0.00381 | -0.00348 |
| 5 | -0.00087 | 0.00376 |
| 6 | -0.00168 | -0.00350 |
| 7 | -0.00509 | -0.00132 |
| 8 | -0.00410 | -0.00457 |
| 9 | 0.00349 | -0.00134 |
| 10 | -0.00144 | -0.00008 |
| 11 | 0.00331 | -0.00008 |
| 12 | -0.00076 | 0.00171 |
| 13 | -0.00434 | -0.00233 |
| 14 | 0.00182 | -0.00322 |
| 15 | 0.00234 | 0.00215 |
| 16 | 0.00308 | 0.00982 |
| 17 | -0.00334 | -0.00095 |
| 18 | -0.00240 | -0.00590 |
| 19 | -0.00016 | 0.00301 |
| 20 | -0.00058 | -0.00019 |
| 21 | 0.00678 | -0.00258 |
| 22 | 0.00177 | 0.00359 |
| 23 | 0.00075 | -0.00414 |
| 24 | -0.00230 | -0.00228 |
| 25 | 0.00355 | -0.00774 |
| 26 | -0.00109 | -0.00467 |
| 27 | -0.00081 | 0.00368 |
| 28 | -0.00190 | 0.00165 |
| 29 | -0.00044 | -0.00080 |
| 30 | -0.00058 | -0.00159 |
| 31 | 0.00630 | 0.00204 |
| 32 | 0.00507 | -0.00023 |
| 33 | -0.00155 | -0.00278 |
| 34 | 0.00230 | -0.00197 |
| 35 | -0.00577 | -0.00088 |
| 36 | -0.00359 | -0.00365 |
| 37 | 0.00632 | -0.00322 |
| 38 | -0.00343 | 0.00090 |
| 39 | -0.00052 | 0.00822 |
| 40 | 0.00256 | -0.00313 |

TABLE B-10
SERIAL CORRELATION: RANDOM(BMDP) APART DISTANCE ( $h$ ), SEED=1 AND 101, $N=100,000$

| $h$ | seed=1 | seed=101 |
| :---: | :---: | :---: |
| 1 | 0.00399 | 0.00645 |
| 2 | -0.00278 | -0.00060 |
| 3 | -0.00261 | -0.00107 |
| 4 | 0.00239 | -0.00066 |
| 5 | 0.00204 | 0.00007 |
| 6 | 0.00496 | -0.00227 |
| 7 | -0.00191 | 0.00657 |
| 8 | 0.00583 | -0.00147 |
| 9 | -0.00460 | 0.00050 |
| 10 | 0.00222 | 0.00274 |
| 11 | 0.00302 | -0.00054 |
| 12 | 0.00216 | 0.00270 |
| 13 | -0.00139 | -0.00151 |
| 14 | -0.00241 | 0.00398 |
| 15 | -0.00176 | 0.00133 |
| 16 | -0.00476 | 0.00069 |
| 17 | -0.00382 | -0.00063 |
| 18 | -0.00402 | 0.00049 |
| 19 | -0.00156 | -0.00008 |
| 20 | 0.00161 | -0.00019 |
| 21 | 0.00071 | -0.00169 |
| 22 | -0.00250 | 0.00229 |
| 23 | 0.00122 | -0.00235 |
| 24 | -0.00188 | 0.00862 |
| 25 | -0.00024 | 0.00174 |
| 26 | 0.00111 | 0.00501 |
| 27 | -0.00553 | -0.00300 |
| 28 | 0.00515 | 0.00362 |
| 29 | -0.00399 | -0.00022 |
| 30 | -0.00055 | 0.00239 |
| 31 | -0.00378 | -0.00167 |
| 32 | -0.00053 | 0.00128 |
| 33 | 0.00611 | 0.00331 |
| 34 | 0.00110 | 0.00524 |
| 35 | 0.00557 | -0.00206 |
| 36 | 0.00014 | -0.00115 |
| 37 | -0.00068 | -0.00152 |
| 38 | 0.00607 | 0.00141 |
| 39 | -0.00408 | -0.00128 |
| 40 | -0.00127 | 0.00291 |

## APPENDIX C

FIGURES

Fig. C-1. Find Period of Random Number Sequence: RANDU(IBM) (MLCG; $a=65539, m=2147483648$ )

| Sequence | Seed | Random Number |
| ---: | ---: | ---: |
|  | 1 | 1 |
| 2 | 65539 | $1.05189759407000 \mathrm{E}-0005$ |
| 3 | 393225 | $8.23987204692500 \mathrm{E}-0004$ |
| 4 | 1769499 | $3.295936245899700 \mathrm{E}-0004$ |
| 5 | 7077969 | $1.23597326331999 \mathrm{E}-0003$ |
| 6 | 26542323 | $4.44949695861021 \mathrm{E}-0002$ |
| 7 | 95552217 | $1.55732223817813 \mathrm{E}-0001$ |
| 8 | 334432395 | $5.33938616631962 \mathrm{E}-0001$ |
| 9 | 1146624417 | $8.02041658175029 \mathrm{E}-0001$ |
| 10 | 1722371299 | $6.80239936251330 \mathrm{E}-0003$ |
| 11 | 14608041 | $8.22439690651201 \mathrm{E}-0001$ |
| 12 | 1766175739 | $8.73416440618895 \mathrm{E}-0001$ |
| 13 | 1875647473 | $8.38541509621830 \mathrm{E}-0001$ |
| 14 | 1800754131 | $1.70501173930195 \mathrm{E}-0001$ |
| 15 | 366148473 | $4.76133647779654 \mathrm{E}-0001$ |
| 16 | 1022489195 | $3.22291294049744 \mathrm{E}-0001$ |
| 17 | 692115265 | $6.48545016050853 \mathrm{E}-0001$ |
| 18 | 1392739779 | $9.90648449857416 \mathrm{E}-0001$ |
| 19 | 2127401289 | $1.0698554686820 \mathrm{E}-0001$ |
| 20 | 229749723 | $7.26077524711980 \mathrm{E}-0001$ |
| $\cdot$ | $\cdot$ | . |
| $\cdot$ | $\cdot$ | . |
| $\cdot$ | $\cdot$ | . |
| 1610612730 | 1559961379 | $4.22496498436606 \mathrm{E}-0001$ |
| 1610612731 | 907304297 | $9.97256333647354 \mathrm{E}-0001$ |
| 1610612732 | 2141591611 | $1.81069461441826 \mathrm{E}-0001$ |
| 1610612733 | 388843697 | $1.11109983876147 \mathrm{E}-0001$ |
| 1610612734 | 238606867 | $3.70347775368701 \mathrm{E}-0002$ |
| 1610612735 | 79531577 | $2.22218837592319 \mathrm{E}-0001$ |
| 1610612736 | 477211307 | $4.6566130000000 \mathrm{E}-0010$ |
| 1610612737 |  | $3.05189759407000 \mathrm{E}-0005$ |
| 1610612738 | 65539 | $1.83109664692500 \mathrm{E}-0004$ |
| 1610612739 | 393225 | $8.23987204688700 \mathrm{E}-0004$ |
| 1610612740 | 1769499 | $3.29593624589970 \mathrm{E}-0003$ |
| 1610612741 | 7077969 | $1.23597326331999 \mathrm{E}-0002$ |
| 1610612742 | 26542323 | $4.44949695861021 \mathrm{E}-0002$ |
| 1610612743 | 95552217 | $1.55732223817813 \mathrm{E}-0001$ |

Fig. C-2. Find Period of Random Number Sequence: RNUN (IMSL) (MLCG; $a=950706376, m=2147483647$ )

| Sequence | Seed | Random Number |
| :---: | :---: | :---: |
| 1 | 1 | $4.42707155105987 \mathrm{E}-0001$ |
| 2 | 950706376 | $6.00829585735141 \mathrm{E}-0002$ |
| 3 | 129027171 | $8.04783729745440 \mathrm{E}-0001$ |
| 4 | 1728259899 | $1.70050721229078 \mathrm{E}-0001$ |
| 5 | 365181143 | $9.15882680991610 \mathrm{E}-0001$ |
| 6 | 1966843080 | $4.86697532463212 \mathrm{E}-0001$ |
| 7 | 1045174992 | $2.96242899865025 \mathrm{E}-0001$ |
| 8 | 636176783 | $7.46408942037452 \mathrm{E}-0001$ |
| 9 | 1602900997 | $2.98420475934828 \mathrm{E}-0001$ |
| 10 | 640853092 | $2.00195465795787 \mathrm{E}-0001$ |
| 11 | 429916489 | $7.78344427132208 \mathrm{E}-0001$ |
| 12 | 1671481929 | $5.98657634853692 \mathrm{E}-0001$ |
| 13 | 1285607481 | $4.96484454021083 \mathrm{E}-0001$ |
| 14 | 1066192246 | $2.27228291438533 \mathrm{E}-0002$ |
| 15 | 48796904 | 5.47819965774110E-0001 |
| 16 | 1176434418 | 3.61547745001292E-0001 |
| 17 | 776417870 | $4.01150183007657 \mathrm{E}-0001$ |
| 18 | 861463458 | $7.18946064225839 \mathrm{E}-0001$ |
| 19 | 1543924916 | $2.59610212994558 \mathrm{E}-0001$ |
| 20 | 557508687 | $7.68644253149929 \mathrm{E}-0001$ |
| . | . |  |
| - |  |  |
| , |  |  |
| 2147483641 | 269649070 | $6.56675191901939 \mathrm{E}-0001$ |
| 2147483642 | 1410199236 | $9.02196606575603 \mathrm{E}-0001$ |
| 2147483643 | 1937452459 | $2.76989248710214 \mathrm{E}-0001$ |
| 2147483644 | 594829882 | 8.32250046000001E-0001 |
| 2147483645 | 1787243364 | $1.58494286778613 \mathrm{E}-0001$ |
| 2147483646 | 340363889 | $4.65661287524580 \mathrm{E}-0010$ |
| 2147483647 | 1 | $4.42707155105987 \mathrm{E}-0001$ |
| 2147483648 | 950706376 | $6.00829585735141 \mathrm{E}-0002$ |
| 2147483649 | 129027171 | $8.04783729745440 \mathrm{E}-0001$ |
| 2147483650 | 1728259899 | $1.70050721229078 \mathrm{E}-0001$ |
| 2147483651 | 365181143 | 9.15882680991610E-0001 |
| 2147483652 | 1966843080 | $4.86697532463212 \mathrm{E}-0001$ |
| 2147483653 | 1045174992 | $2.96242899865025 \mathrm{E}-0001$ |
| 2147483654 | 636176783 | $7.46408942037452 \mathrm{E}-0001$ |
| 2147483655 | 1602900997 | $2.98420475934828 \mathrm{E}-0001$ |

Fig. C-3. Find Period of Random Number Sequence: RANUNI(SAS) (MLCG; $a=397204094, m=2147483647$ )

Sequence

2147483640
2147483641
2147483642
2147483643
2147483644
2147483645
2147483646
2147483647
2147483648
2147483649
2147483650
2147483651
2147483652
2147483653
2147483654
2147483655

Seed
$1 \quad 1.84962569822074 \mathrm{E}-0001$
$397204094 \quad 9.70088715651114 \mathrm{E}-0001$
2083249653 3.99824306089349E-0001
$858616159 \quad 2.59398645376507 \mathrm{E}-0001$
$557054349 \quad 9.21602577865870 \mathrm{E}-0001$
$1979126465 \quad 9.69277349752037 \mathrm{E}-0001$
$2081507258 \quad 5.42979173149438 \mathrm{E}-0001$
$1166038895 \quad 5.31691722819438 \mathrm{E}-0001$
$1141799280 \quad 4.97940262080142 \mathrm{E}-0002$
$106931857 \quad 6.65665516008467 \mathrm{E}-0002$
$142950581 \quad 8.19318570578153 \mathrm{E}-0001$
$1759473232 \quad 5.23870521468981 \mathrm{E}-0001$
$11250033788.53394310853162 \mathrm{E}-0001$
1832650327 6.71845767959880E-0002
$144277780 \quad 9.57023857607052 \mathrm{E}-0001$
$20551930842.97193964150359 \mathrm{E}-0001$
$6382191782.72611789066629 \mathrm{E}-0001$
$5854293596.89929630928640 \mathrm{E}-0001$
$1481612600 \quad 9.76764862414806 \mathrm{E}-0001$
$20975865692.26507518545961 \mathrm{E}-0001$

| Seed | Random Number |
| ---: | :---: |
|  |  |
| 1 | $1.84962569822074 \mathrm{E}-0001$ |
| 397204094 | $9.70088715651114 \mathrm{E}-0001$ |
| 2083249653 | $3.99824306089349 \mathrm{E}-0001$ |
| 858616159 | $2.59398645376507 \mathrm{E}-0001$ |
| 557054349 | $9.21602577865870 \mathrm{E}-0001$ |
| 1979126465 | $9.69277349752037 \mathrm{E}-0001$ |
| 2081507258 | $5.42979173149438 \mathrm{E}-0001$ |
| 1166038895 | $5.31691722819438 \mathrm{E}-0001$ |
| 1141799280 | $4.97940262080142 \mathrm{E}-0002$ |
| 106931857 | $6.65665516008467 \mathrm{E}-0002$ |
| 142950581 | $8.19318570578153 \mathrm{E}-0001$ |
| 1759473232 | $5.23870521468981 \mathrm{E}-0001$ |
| 1125003378 | $8.53394310853162 \mathrm{E}-0001$ |
| 1832650327 | $6.71845767959880 \mathrm{E}-0002$ |
| 144277780 | $9.57023857607052 \mathrm{E}-0001$ |
| 2055193084 | $2.97193964150359 \mathrm{E}-0001$ |
| 638219178 | $2.72611789066629 \mathrm{E}-0001$ |
| 585429359 | $6.89929630928640 \mathrm{E}-0001$ |
| 1481612600 | $9.76764862414806 \mathrm{E}-0001$ |
| 2097586569 | $2.26507518545961 \mathrm{E}-0001$ |
| . | . |
| . | . |
|  | . |
| 1241614661 | $4.27323462640552 \mathrm{E}-0001$ |
| 917670148 | $8.23083145927211 \mathrm{E}-0001$ |
| 1767557596 | $2.64687584370695 \mathrm{E}-0001$ |
| 568412259 | $1.43010412409441 \mathrm{E}-0001$ |
| 307112522 | $2.93658292523426 \mathrm{E}-0001$ |
| 630626381 | $2.73544537030880 \mathrm{E}-0002$ |
| 58743242 | $4.65661287524580 \mathrm{E}-0010$ |
| 397204094 | $1.84962569822074 \mathrm{E}-0001$ |
| 2083249653 | $9.70088715651114 \mathrm{E}-0001$ |
| 858616159 | $2.59824306089349 \mathrm{E}-0001$ |
| 557054349 | $9.21602545376507 \mathrm{E}-0001$ |
| 1979126465 | $9.6927734975870 \mathrm{E}-0001$ |
| 2081507258 | $5.42979173149438 \mathrm{E}-0001$ |
| 1166038895 | $5.31691722819438 \mathrm{E}-0001$ |
| 1141799280 | $4.97940262080142 \mathrm{E}-0001$ |
|  | . |

1241614661
$4.27323462640552 \mathrm{E}-0001$
$8.23083145927211 \mathrm{E}-0001$
$17675575962.64687584370695 \mathrm{E}-0001$
$5684122591.43010412409441 \mathrm{E}-0001$
$307112522 \quad 2.93658292523426 \mathrm{E}-0001$
$6306263812.73544537030880 \mathrm{E}-0002$
$\begin{array}{ll}1 & 4.65661287524580 \mathrm{E}-0010 \\ 1.84962569822074 \mathrm{E}-0001\end{array}$
$397204094 \quad 9.70088715651114 \mathrm{E}-0001$
0832953 3.99824306089349E-0001
557054349 9. $21602577865870 \mathrm{E}-001$
1979126465 9.69277349752037E-0001
$2081507258 \quad 5.42979173149438 \mathrm{E}-0001$
$11417992804.97940262080142 \mathrm{E}-0002$

Fig. C-4. Find Period of Random Number Sequence: UNIFORM (SPSS) (MLCG; $a=16807, m=2147483647$ )

| Sequence | Seed | Random Number |
| :---: | :---: | :---: |
| 1 | 1 | 7.82636925942561E-0006 |
| 2 | 16807 | $1.31537788143166 \mathrm{E}-0001$ |
| 3 | 282475249 | $7.55605322195033 \mathrm{E}-0001$ |
| 4 | 1622650073 | 4.58650131923449E-0001 |
| 5 | 984943658 | $5.32767237412169 \mathrm{E}-0001$ |
| 6 | 1144108930 | $2.18959186328090 \mathrm{E}-0001$ |
| 7 | 470211272 | $4.70446162144861 \mathrm{E}-0002$ |
| 8 | 101027544 | $6.78864716868319 \mathrm{E}-0001$ |
| 9 | 1457850878 | $6.79296405836612 \mathrm{E}-0001$ |
| 10 | 1458777923 | 9.34692895940828E-0001 |
| 11 | 2007237709 | $3.83502077489859 \mathrm{E}-0001$ |
| 12 | 823564440 | 5.19416372067955E-0001 |
| 13 | 1115438165 | $8.30965346112365 \mathrm{E}-0001$ |
| 14 | 1784484492 | 3.45721105274614E-0002 |
| 15 | 74243042 | $5.34616350445252 \mathrm{E}-0002$ |
| 16 | 114807987 | $5.29700193335163 \mathrm{E}-0001$ |
| 17 | 1137522503 | $6.71149384077242 \mathrm{E}-0001$ |
| 18 | 1441282327 | $7.69818621114743 \mathrm{E}-0003$ |
| 19 | 16531729 | $3.83415650754895 \mathrm{E}-0001$ |
| 20 | 823378840 | $6.68422375185612 \mathrm{E}-0002$ |
| . | . |  |
|  |  |  |
|  |  |  |
| 2147483641 | 1483866096 | $2.83998838292434 \mathrm{E}-0001$ |
| 2147483642 | 609882861 | $1.68475180942786 \mathrm{E}-0001$ |
| 2147483643 | 361797696 | $5.62366105412303 \mathrm{E}-0001$ |
| 2147483644 | 1207672015 | 6.87133664585246E-0001 |
| 2147483645 | 1475608308 | $6.55500684238738 \mathrm{E}-0001$ |
| 2147483646 | 1407677000 | $4.65661287524580 \mathrm{E}-0010$ |
| 2147483647 | 1 | $7.82636925942561 \mathrm{E}-0006$ |
| 2147483648 | 16807 | $1.31537788143166 \mathrm{E}-0001$ |
| 2147483649 | 282475249 | $7.55605322195033 \mathrm{E}-0001$ |
| 2147483650 | 1622650073 | $4.58650131923449 \mathrm{E}-0001$ |
| 2147483651 | 984943658 | $5.32767237412169 \mathrm{E}-0001$ |
| 2147483652 | 1144108930 | $2.18959186328090 \mathrm{E}-0001$ |
| 2147483653 | 470211272 | $4.70446162144861 \mathrm{E}-0002$ |
| 2147483654 | 101027544 | $6.78864716868319 \mathrm{E}-0001$ |
| 2147483655 | 1457850878 | $6.79296405836612 \mathrm{E}-0001$ |

## BIBLIOGRAPHY

BMDP 1983. Statistical Software. Berkeley: University of California Press, California.
BMDP 1992. BMDP Statistical software Manual, volume 1. Berkeley: University of California Press, California.

Borland 1992. Turbo Pascal 7.0. Scott Valley: Borland International Inc., California.
Box, G. E. P. and Muller, M. E. 1958. A note on the generation of random normal deviates. Annals of Mathematical Statistics 29: 610-613.

Bratley, P., Fox, B. L. and Schrage, L. E. 1987. A guide to simulation, 2nd edition. New York: Springer-Verlag, New York.

Campagner, A. 1992. Fast and reliable random-number generation. Nelson, B. L., Kelton, W. D., and Clark G. M. (editors). Proceedings of the 1992 Winter Simulation Conference. Washington, DC: IEEE Press, 438-442.

Carta, D. G. 1990. Two fast implementations of the "minimal standard" random number generator. Communications of the ACM 33, 1:87-88.

Coveyou, R, R., and McPherson, R. D. 1967. Fourier analysis of uniform random number generators. Journal of the Association for Computing Machinery 14: 100-119.

Deng, L., and Chu, Y. 1991. Combining random number generators. Nelson, B. L., Kelton, W. D., and Clark G. M. (editors), Proceedings of the 1991 Winter Simulation Conference. Washington DC: IEEE Press, 1043-1046.

Dudewicz E. J. and Rally T. G. 1981. The Handbook of Random Number Generation and Testing with TESTRAND Computer Code. Columbus: American Science Press, Ohio.

Fishman, G. S. and Moore, L. S., III. 1982. A statistical evaluation of multiplicative congruential random number generators with modulus $2^{31}-1$. Journal of the American Statistical Association 77: 129-136.

Fishman, G. S. and Moore, L. S., III. 1986. An exhaustive analysis multiplicative congruential random number generators with modulus $2^{31}-1$. SIAM Journal on Scientific and Statistical Computing 7, 1: 24-45.

Haas, A. 1987. The multiple prime random number generator. ACM Transactions on Mathematical Software 13, 14: 368-381.

Hamilton, K. G. 1993. Pseudorandom number generators for personal computers. Computer Physics Communications 75: 105-117.

IMSL 1991. Library Users Manual, Stat/Library, version 2.0. Houston: IMSL Inc., Texas.

James F. 1990. A review of pseudorandom number generators. Computer Physics Communicalions 60: 329-344.

Kleijnen, J. P. and Groenendaal W. 1992. Simulation: A statistical perspective. West Sussex: John Wiley \& Son, England.

Knuth, D. E. 1981. The Art of Computer Programming, volume 2:Seminumerical Algorithm. 2nd edition. Reading: Addison-Wesley, Massachusetts.

Leva, J. L. 1992a. A fast normal random number generator. ACM Transactions on Mathematical Software 18, 4: 449-453.

Leva, J. L. 1992b. Algorithm 712: A normal random number generator, ACM Transactions on Mathematical Software 18, 4: 454-455.

L'Ecuyer, P. 1988. Efficient and portable combined random number generators. Communications of the ACM 31, 6: 742-749.

L'Ecuyer, P. 1990. Random numbers for simulation. Communications of the ACM 33, 10: 85-97.

L'Ecuyer, P. 1992. Testing random number generators. Nelson, B. L., Kelton, W. D., and Clark G. M. (editors), Proceedings of the 1992 Winter Simulation Conference. Washington, DC: IEEE Press, 305-313.

L'Ecuyer, P., Blouin, F. and Couture, R. 1993. A search for good multiple recursive random number generators. ACM Transactions on Modeling and Computer Simulation, 3, 2: 87-98.

Lewis, P. A. W., Goodman, A. S. and Miller, J. M. 1969. A pseudo-random number generator for the System/360. IBM Systems Journal 8, 2: 136-146.

Marsaglia, G. 1968. Random numbers fall mainly in the planes. Proceedings of the National Academy of Sciences of the United States of America 61. Washington, DC:

GPO., 25-28.
Marsaglia, G. 1985. A current view of random number generators. Billard, L. (editor). Computer Science and Statistics: Proceedings of Sixteenth Symposium on the Interface. Amsterdam: Elsevier Science Publishers, North-Holland, 3-10.

Marsaglia, G., Narasimhan, B. and Zaman, A. 1990. A random number generator for PC's. Computer Physics Communications 60: 345-349.

Marsaglia, G. and Zaman, A. 1991. A new class of random number generators. Annals of Applied Probability, volume 1, 3: 462-480

Marsaglia, G., Zaman A. and Tsang, W. W. 1990. Toward a universal random number generator. Statistics \& Probability Letters, 9: 35-39.

Modianos, D. T., Scott, R. C. and Cornwell, L. W. 1984. Random number generation on microcomputers. Interfaces 14, 4: 81-87.

Niederreiter, H. 1992. Random number generation and quasi-Monte Carlo methods, Philadelphia: SIAM, Pensilvania.

Park, S. K. and Miller, K. W. 1988. Random number generators: Good ones are hard to find. Communications of the ACM 31, 10: 1192-1201.

Payne, J. A. 1982. Introduction to Simulation: Programming Techniques and Methods of Analysis. New York: McGraw-Hill, New York.

RAND Corporation 1955. A Million Random Digits with 100000 Normal Deviates. New York: The Free press, New York.

Ripley, B. D. 1983. The lattice structure of pseudorandom number generators. Proceedings of the Royal Society of London, A 389: 197-204.

Ripley, B. D. 1987. Stochastic simulation. New York: John Wiley \& Son, Inc., New York.

SAS 1990. SAS Language: Reference, volume 6. Cary: SAS Institute Inc., North Carolina.

Selby, S. M. 1975. SCRC Standard Mathematical Tables. 23rd edition. Boca Raton: CRC Press, Florida.

Schrage, L. 1979. A more portable FORTRAN random number generator, $A C M$ Transactions on Mathematical Software 5, 2: 132-138.

Sobol, I. M. 1974. The Monte Carlo method. Chicago: University of Chicago Press, Illinois.

SPSS 1990. Statistical Algorithm. 2nd edition. Chicago: SPSS inc., Illinois.
SPSS 1994. SPSS for Windows 6.1. Chicago: SPSS inc., Illinois.
Tausworthe, R. C. 1965. Random numbers generated by linear recurrence modulo two. Mathematics of Computation: 201-209

Tezuka, S., and L'Ecuyer, P. 1992. Analysis of add-with-carry and subtract-with-borrow generators. Swain, J. J., Goldsman, D., Crain, R. C., and Wilson, J. R. (editors). Proceedings of the 1992 Winter Simulation Conference. Washington, DC: IEEE Press, 443-447.

Tezuka, S., L'Ecuyer, P. and Coutre, R. 1993. On the lattice structure of the add-withcarry and subtract-with-borrow random number generators. ACM Transaction on Modeling and Computer Simulation 3, 4: 315-331.
von Neumann, J. 1952. Various techniques used in connection with random digits, U.S. National Bureau Standard Applied Mathematics Series 12. Washington, DC: GPO., 36-38.

Wichmann, B. A. and Hill, I. D. 1982. An efficient and portable pseudorandom number generator. Applied Statistics 31: 188-190.

