A PROGRAM TO EVALUATE UNIFORM RANDOM NUMBER GENERATORS

A Thesis Presented to the Faculty

of the

Department of Industrial and Systems Engineering

University of Houston

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

in Operations Research

---

by

Andrew J. Sacks

August 1974

ACKNOWLEDGMENTS

I wish to express my appreciation to the members of my committee, Dr. Donaghey, Dr. Rhodes and Dr. Motard, for their assistance. I also wish to express my appreciation to Gulf Oil Corporation for permitting me to avail myself of their facilities while completing requirements for the Master of Science degree. Finally, to my wife, Barbara, and son, Jonathan, a sincere thanks for their patience and understanding.

ABSTRACT

This thesis presents a program to evaluate uniform pseudo-random number generators. It presents various methods of generation and the tests available to test the adequacy of the methods. The program is employed to test a number of frequently used generators and the results are reported. The strengths and weaknesses of the programmed tests are also discussed.

TABLE OF CONTENTS

CHAPTER 1

RANDOM NUMBER GENERATORS

A. Introduction

There have been many words written and spoken on the relative merits and shortcomings of various types of uniform random number generators. As soon as a new algorithm or procedure is presented as being totally random and unbiased a critic of the new method discovers an area where the performance of the procedure is less than ideal. This type of banter has been going on for years and with the emergence of the digital computer and the availability of a high-speed means of generating these numbers the arguments are becoming more and more frequently found than ever in the journals of computer and statistical methodology. One point, however, is not argued; that is the statistical importance of random number generators in a wide variety of applications.

Random numbers are useful in many areas. For example:

a. Simulation - when a computer is used to simulate natural phenomena, random numbers are required to make things realistic. The term simulation is rather broad and covers studies from nuclear physics and space technology to queuing theory (for example, people entering a bank or cafeteria at random intervals expecting services) and computer software simulation (analysis of various types of software, hardware, peripherals and job streams to optimize throughput).

b. Sampling - in many statistical analyses it is often impractical to examine all possible cases due to the large number of possibilities. A well chosen random sample, will, however, allow the statistician to draw meaningful conclusions from the data and gain insight into the problem from a substantially smaller subset of observations.

c. Numerical analysis - many techniques for solving complicated numerical problems have been devised using random numbers.

d. Computer programming - random values make a good source of data for testing the effectiveness of computer algorithms.

e. Decision making - random numbers are becoming increasingly more important in association with corporate and industrial decisions. Techniques such as decision theory and risk analysis employ random numbers in a simulation type application to aid managers in evaluating various alternatives.

f. Recreation - rolling dice, shuffling cards, playing roulette are all every day occurrences which involve random number theory. This commonplace use of random numbers has had the name "Monte Carlo Method" devised as the general term used to describe an algorithm that employs random numbers.

Although we have been and will probably continue to talk about random numbers, there really is no such entity as a random number. We cannot, for example, say 21 is a random number or 21 is not a random number. What we are actually saying is that we really are speaking about a sequence of independent random numbers with a previously specified distribution and that each number observed by us was obtained by chance, having nothing to do with other numbers in the sequence.

Uniformly distributed variables in the range zero to one (denoted (0, 1)) play an important role in the generation of random variables drawn from other probability distributions such as the exponential, normal, Poisson and binomial distributions. In fact, random variables from these distributions are often derived by transforming one or more uniform (0, 1) random variables. For example, $-\ln(r)$ where r is a random variable from a uniform (0, 1) yields an exponentially distributed random variable with mean 1. For this reason, we will concern ourselves primarily with the uniform distribution and the distributions referred to should be understood to be uniform unless some other distribution is explicitly stated.

A uniform distribution is one in which each possible number is equally probable. In other words each of the ten digits 0 through 9 will occur about 1/10 of the time in a (uniform) sequence of random digits. Each pair of two successive digits (00 through 99) should occur about 1/100 of the time and so on. Yet if we take a truly random sequence of 1000 digits, it will not always have exactly 100 zeros, 100 ones, etc. In fact, the probability of this actually occurring is quite small; however, a sequence of such sequences will have this character on the average.

Another quality of a random number generator besides the frequency of the appearance of the digits, is the actual sequence of the digits.

A sequence such as

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

or

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

might have each digit appear with the same frequency but indeed there is a suspicious ordering to the digits which would cast doubt on the validity of the generator.

There are several ways to formulate a good abstract definition of a random sequence but perhaps we should begin with an intuitive approach to the concept.

One of the first published accounts of random digits appeared in 1927 as a table of over 40,000 random digits taken at random from census reports. This table was compiled by L. H. C. Tippett (17). Since then a number of machines were built for mechanically generating random numbers. M. G. Kendall and B. Babington-Smith (7, 8) describe such a machine in their papers in the Journal of the Royal Statistical Society. The machine essentially consisted of a spinning disk divided into ten equal sections, each having a digit from 0 to 9 on it. The disk was caused to spin at a relatively high speed in a dark room. The operator would at unequal intervals press a button to flash a light on for an instant which would illuminate the spinning disk, making it appear motionless and noticing the digit at which a previously mounted pointer was indicating. Statistical analyses of the sequences generated by the above method proved it to be a reasonable means of random number generation.

Shortly after computers were introduced people began to search for efficient ways to obtain random numbers in computer programs. One obvious way is to read in tables of already known random sequences. This method, however, is of limited utility because of memory available, input time requirement, the table may be too short for the application and it certainly seems like it would be a nuisance to prepare and maintain the table. Obviously, the computer should be programmed to generate its own random number sequences.

B. Random Numbers by Computer

The idea of using the computer and its inherent high speed for generating random numbers was suggested in the 1940's by John von Neumann. His method was called the "middle square" method and consisted of using the previously generated random number, squaring it and extracting the middle digits as the new random number. So, for instance, if we were generating 10-digit numbers and the previous value was 5,772,156,649 the new random number would be 7923805949 (the middle ten digits of 33317792380594909291).

One obvious objection to this technique is how can a sequence generated in such a way be random since each number is completely determined by its predecessor. The answer, of course, is that the sequence is not random, but it appears to be. For this reason such deterministically generated sequences are called pseudo-random or quasi-random sequences. Although they are derived from a fully specified formula by a digital computer, their statistical properties coincide with the statistical properties of numbers generated by an "idealized chance device" that selects numbers from the unit interval independently and with all numbers equally likely. Provided that these pseudo-random numbers can pass the set of statistical tests (frequency tests, auto-correlation tests, lagged product tests, runs tests, gap test and others all of which will be fully detailed in this paper) implied by an idealized chance device, then these pseudo-random numbers can be treated as "true" random numbers even though they are not.

Naylor (15) proposes several criteria that should be satisfied by an ideal pseudo-random number generator. An ideal pseudo-random number generator should yield sequences of numbers that are 1) uniformly distributed, 2) statistically independent, 3) reproducible and 4) nonrepeating for any desired length. Furthermore, such a generator should also be capable of 5) generating random numbers at high rates of speed, yet of 6) requiring a minimum amount of computer memory capacity. He also states that congruential methods of random number generation come closer to satisfying all of these criteria than any other known method.

Congruential Methods

The congruential methods for generating pseudo-random numbers are based on the mathematical concept of congruence which basically states that two numbers a and b are congruent modulo m if their difference is an integral multiple of m. The congruence relation is expressed by the notation $a \equiv b \pmod{m}$ and is read "a is congruent to b modulo m." In other words a-b is exactly divisible by m which also implies that a/m and b/m have the same remainders.

The following recursive formula is basic to all congruential methods.

$$X_{n+1} = (aX_n + c) \bmod m \qquad n > 0 \qquad \text{(I-1)}$$

where X_o, a, c, and m are all non-negative integers. Expanding (I-1) for i = 0, 1, 2... we obtain

$$X_1 = (aX_o + c) \bmod m$$

$$X_2 = (a(aX_o + c) \bmod m$$

$$= a^2X_o + (a + 1)\, c \bmod m$$

$$X_3 = a^3X_o + (a^2+a+1)\, c \bmod m$$

$$= a^3X_o + \frac{(a^3 - 1)}{(a - 1)}\, c \bmod m$$

$$X_i = a^iX_o + \frac{(a^i - 1)}{(a - 1)}\, c \bmod m \qquad \text{(I-2)}$$

Given an initial starting value X_o, a constant multiplier a, and an additive constant c, then (I-2) yields a congruence relationship (modulo m) for values of i, i=0, 1, 2... .

This sequence is called a linear congruence sequence. For example, the sequence obtained when X_o = a=c=7, m=10, is 7, 6, 9, 0, 7, 6, 9, 0, ... As this example shows, the sequence is not always "random" for all choices of the initial parameters; in fact the choice of these values is extremely critical in producing useful random sequences.

The above example illustrates another characteristic of congruential sequences: they always get into a loop or begin repeating themselves. This repeating cycle is called the period; the sequence above having a period of 4. Obviously a useful sequence will have a relatively long period.

Two types of congruential methods have been derived from Equation (I-2). When $c = 0$ the term multiplicative congruential generator is often used and when $c \neq 0$ often the term mixed congruential method is employed. The case $c = 0$ generally proceeds a little faster than when $c \neq 0$. However the restriction $c = 0$ cuts down the length of the period of the sequence although careful selection of the other parameters still allows for a reasonably long period.

Far and away, the most widely used method of generating random numbers by computer employs in one variation or another a congruential method. In almost all of these cases the method used is a multiplicative congruential method or a mixed congruential method. These two methods and two variations of them are discussed below.

<u>The Multiplicative Method</u>

The multiplicative congruential method computes a sequence $\{X_i\}$ of non-negative integers less than m by means of the congruence relation

$$X_{i+1} = aX_i \mod m \quad \text{(I-3)}.$$

This method is a special case of the congruence relation (I-1) where $c = 0$. This method has been found to behave quite well from a statistical point of view. Given a prudent choice of the multiplier a and starting value (or seed) X_o, it is possible to generate sequences of numbers that are uncorrelated and uniformly distributed. Additionally, careful choice of the two above-mentioned parameters guarantees a maximum period for sequences generated by this method. Knuth (9) states the following theorem concerning multiplicative congruential generators and cites the proof by R. D. Carmichael, Bulletin of the American Math. Society, Vol, 16, (1910), pp 232-38.

Theorem

The maximum period possible when $c = 0$ (a multiplicative congruential method) is achieved if

1) X_o is relatively prime to m
2) a is a primative element modulo m.

To clarify the above theorem it is also necessary to note that when a is relatively prime to m, the smallest integer λ for which $a^{\lambda} \equiv 1 \pmod{m}$ is called "the order of a, modulo m". Any value of a which gives the maximum possible order modulo m is called a "primative element, modulo m."

If we let $\lambda(m)$ denote the order of a primative element, i.e., the maximum possible order, modulo m, we can show (Knuth) that $\lambda(p^e)$, where p is a prime number and e a positive integer, $p^e > 2$, is a divisor of $p^{e-1}(p-1)$. The precise value of $\lambda(m)$ for all cases can then be given as

$$\begin{aligned}\lambda(2) &= 1\\ \lambda(4) &= 2\\ \lambda(2^e) &= 2^{e-2}, \; e \geq 3\\ \lambda(p^e) &= p^{e-1}(p-1), \; p \geq 2.\end{aligned}$$

This theory simplifies somewhat when we note that as applicable to computer generated random numbers p represents 2 or 10 depending on whether the generator is to be run on a binary or decimal computer and e represents the number of bits per word available for computation. For the IBM 360 then $p = 2$, $e = 31$.

Knuth then presents another theorem as follows:

<u>Theorem</u>

a is a primative element modulo p^e if and only if

i) $p^e = 2$, a is odd;

$p^e = 4$, a mod 4 = 3; or

$p^e = 8$, a mod 8 = 3, 5, 7; or

$p = 2$, $e \geq 4$, a mod 8 = 3, 5;

or

ii) p is odd, $e = 1$, $a \not\equiv 0 \pmod p$ and

$a^{(p-1)/q} \not\equiv 1 \pmod p$ for any prime divisor

q of p-1;

or

iii) p is odd, $e > 1$, a satisfies ii) and

$a^{p-1} \not\equiv 1 \pmod{p^2}$

Once again this theory simplifies in the common case where $m = 2^e$, $e \geq 4$ and our sole requirement is that a = 3 or 5 (modulo 8).

The Mixed Method

The mixed congruential method computes a sequence of $\{X_i\}$ of non-negative integers less than m by means of the congruence relation given by

$$X_{i+1} = aX_i + c \pmod{m}.$$

This method differs from the multiplicative procedure in that c is not assumed to be zero. The advantage of this method is that it is possible to obtain sequences with a period that covers the full set of m different numbers (the multiplicative method has a maximum period of m-1). From a computational and speed standpoint this method requires an extra addition operation compared to the multiplicative method.

The theorem concerning the maximum period for a linear congruential sequence is as follows:

Theorem

The linear congruential sequence has a period of length m if and only if

i) c is relatively prime to m

ii) $a \equiv 1 \pmod{p}$, p is a prime factor of m

iii) $a \equiv 1 \pmod{4}$ if 4 is a factor of m.

The practical considerations of this condition, when dealing with binary computers, is relatively straightforward. To achieve a full period $h = m = 2^e$ the parameter c must be odd and a must satisfy the congruence relationship

$$a \equiv 1 \pmod{4}$$

which can always be achieved by setting $a = 2^k + 1$ for $k \geq 2$. Any positive integer can be selected for the starting value X_o. However the above-mentioned conditions are not in themselves sufficient for assuming that sequences generated by the mixed congruential method will be statistically satisfactory. Naylor states that only by empirical testing can we have confidence in the statistical properties of sequences that are produced.

The Combination Methods

Within a few years after their discovery, congruential methods came under attack in the literature on the grounds that the sequences generated were not statistically satisfactory especially with respect to autocorrelation (a measure of the tendency of numbers in the sequence to show a linear functional relationship to other numbers in the sequence a given but constant distance away). As a result of these findings, several new versions of congruential methods were suggested in the literature.

MacLauren and Marsaglia (11) suggested two combination methods. The first method, to be used on computers with buffered input, reads in a tape of p previously stored random numbers. Then, a congruential generator computes an index that determines which random number is selected from the inputted sequence. This process is continued until all the inputted numbers are exhausted at which time a new tape is read and the process continues. The second method suggested by these two men uses two random number generators. To begin, a table of 128 locations in core was filled with numbers generated by the first generator. To output random numbers the second generator computed an index to determine the location in the table of the random number to be used. The location in the table was refilled with a new number generated from the first congruential generator. MacLauren and Marsaglia recognize that the time required using this method is about twice the time required in a non-combination method but they strongly feel the time penalty is worth suffering to obtain a sequence of numbers with better statistical properties. In fact an article written three years later by Marsaglia and Bray (14) presents a combination method that involves three congruential generators. They state "short and fast programs will result even if three generators are mixed. One to fill, say 128 storage locations, one to choose a location from the 128 and a third thrown in just to appease the gods of chance. Why be half (or two-thirds safe)?"

Another method of generating pseudo-random numbers based on the combination of two congruential generators has been proposed by Westlake (18). It retains two of the desirable features of congruential generators, namely, the long cycle and the ease of implementation with a digital computer. Unlike the combination method of MacLauren and Marsaglia, Westlake's method does not require the retention in memory of a table of generated numbers. Westlake, instead, uses the two generators and does bit-wise addition followed by division. To further insure randomness, Westlake also adopted the procedure of modifying one random number by permuting its bits in a random manner determined by the other random number. Like the generators of MacLauren and Marsaglia, this procedure yielded completely satisfactory results on a fairly stringent series of statistical tests.

The combination methods are so prevalent as to be too numerous to describe. More recently, however, the enthusiasm has been dampened somewhat by the paper by Coveyou and MacPherson (1). They conclude, through Fourier analysis, that any multiplicative generator is statistically satisfactory if its multiplier meets certain requirements. On the other hand Marsaglia (13) still maintains that every multiplicative generator has a defect which makes it unsuitable for certain Monte Carlo problems namely - points produced in the n-cube fall in a relatively small number of parallel hyper-planes.

Other Methods

Of course, linear congruential methods of pseudo-random number generating are not the only methods ever suggested for computer use. There are a number of other methods which should be briefly discussed.

One of the common fallacies encountered in connection with random number generation is that a good generator can be modified slightly to yield an even better generator. Actually this is not so and in fact the new generator is oftentimes less random than the original one. Knuth expresses this idea as a moral to an episode where he thought he was creating a fantastically good random number generator using a rather complicated and peculiar algorithm, without actually examining the theory behind the algorithm. The algorithm when implemented proved deficient in many areas and led Knuth to state "random numbers should not be generated with a method chosen at random."

Two additional methods of generating random numbers are quadratic congruential methods of the form

$$X_{n+1} = (dX_n^2 + aX_n + c) \mod m \qquad \text{(I-4)}$$

and

$$X_{n+1} = X_n(X_n + 1) \mod 2^e, \; X_o \mod 4 = 2. \qquad \text{(I-5)}$$

In the case of (I-4) the sequence has a period of length m provided parameters a, c and d are properly chosen. Case (I-5) involves less computation time than (I-4), in fact just slightly more time than the linear congruential form of (I-1).

One other nonlinear congruential method of generating random numbers involves using the Fibonacci sequence, $X_{n+1} = X_n + X_{n-1}$. This sequence, which in itself is important in describing many natural phenomena can be modified by division modulo m to produce a random sequence of a relatively long period. However, recent studies of such sequences have proved them not to be satisfactorily random. A slight modification to the Fibonacci sequence to the form

$$X_{n+1} = (X_n + X_{n-k}) \mod m$$

when k is comparatively large has been shown to produce acceptable sequences of random numbers (k = 16).

There are therefore different and varied methods of generating random numbers by computer. Without knowledge of the particular application however it would be indeed difficult to recommend one over any other. Chapter 2 deals with various statistical tests to aid in selecting a satisfactory generator.

CHAPTER 2

STATISTICAL TESTS FOR EVALUATING RANDOM NUMBER GENERATORS

The statistical properties of pseudo-random numbers generated by methods such as those described in Chapter 1 should of course coincide with the statistical properties of numbers generated by an idealized chance device that selects numbers from the unit interval (0, 1) independently and with all numbers equally likely. Obviously, as we have previously mentioned, the numbers generated by computer are not random because they are completely determined by a number of initial parameters and have their precision limited to the accuracy of the computer. However, we will agree that as long as our pseudo-random numbers can pass a rigid set of statistical tests that the idealized generator would theoretically also pass, the pseudo-random numbers will be treated as "truly" random numbers.

Because random number generators are frequently used in the simulation of nondeterministic or stochastic systems the importance of the statistical agreement described above becomes evident. For example, if the probability of the occurrence of a physical event at a given point in time is .60, then if the generated random number assigned to that event at that point in time is less than or equal to .60 the event is assumed to have occurred. A generated random number between .60 and 1. would imply the event at this point in time did not occur. Generally, in this manner the entire course of events of a given case are run or simulated and the final outcome along with relevant intermediate results are reported. Obviously a poor or biased random number generator would tend to cast suspicion as to the accuracy of the simulation. A number of statistical tests are available to examine pseudo-random sequences and which allow the analyst or statistician to make statements concerning the apparent randomness or lack of it in a given sequence. There is literally no end to the number of tests that can and have been conceived and, in fact, for specific applications oftentimes a specific test need be developed to protect against biased introduced by the peculiarities of the application itself. In general there are a number of more common tests and these are described below.

A. Moments

An obvious and desirable characteristic of a pseudo-random number generator on the unit interval is agreement between the observed moments and the known theoretical ones. The first moment, or average, is calculated as

$$\overline{X} = \frac{1}{N} \sum_{i=1}^{N} X_i. \qquad \text{(II-1)}$$

The expected value for this quantity would be 1/2. The second moment, denoted $\overline{X^2}$, is expressed as

$$\overline{X^2} = \frac{1}{N} \sum_{i=1}^{N} X_i^2 \qquad \text{(II-2)}$$

and its theoretical value is 1/3. The third moment, or $\overline{X^3}$ is expressed as

$$\overline{X^3} = \frac{1}{N} \sum_{i=1}^{N} X_i^3 \qquad \text{(II-3)}$$

and it has a theoretical value of 1/4. Another quantity directly related to moments, and in the case of a 0 mean distribution identical to the second moment is the variance, denoted S^2, and calculated as

$$S^2 = \overline{X^2} - \overline{X}^2 = \frac{1}{N} \sum_{i=1}^{N} (X_i - \overline{X})^2 \qquad \text{(II-4)}$$

The variance of a uniform distribution on the unit interval (0, 1) is 1/12.

B. Chi-Square (χ^2) Tests

The χ^2 test is perhaps the best known of all statistical tests and it is a basic method which is often used in connection with many other tests. To apply this test we divide the range (0 to 1) of the N samples into r classes and determine the number of samples, V_i, which fall into each class. From the assumed theoretical distribution we compute p_i, the probability of being in the ith class. Then Np_i is the expected number in the ith class and a statistic χ^2 is defined as

$$\chi^2 = \sum_{i=1}^{r} \frac{(V_i - Np_i)^2}{Np_i}$$

and represents a measure of dispersion between the data and the assumed distribution. A comparison can then be made with the computed value of χ^2 and a known value of χ^2 such that if the calculated value is larger than the known value from a table a very small probability can be attached to the conclusion that the observed observations were actually drawn from the assumed distribution. Also, if we have k-independent sets of N observations we can perform similar tests on the k calculated values of χ^2.

The selection of class width is somewhat arbitrary. Generally speaking class width should be chosen so that all Np_i are at least 5 and probably should be nearer to at least 10. The lengths of the class intervals need not all be the same but except for the endpoints of some distributions where larger class widths are needed to satisfy the requirement of $Np_i > 5$, there is not much to be gained by unequal interval sizes. Mann and Wald (10) suggest using k intervals where

$$k = 4 \sqrt[5]{2(n-1)^2/c^2}$$

and c is related to the size of the critical region (the probability associated with the critical region under the null hypothesis or significance level). Some values of c for different significance levels are shown below in Table II-1.

TABLE II-1
Mann-Wald Values of c for Some
Significance Levels

Significance Level	c
.001	3.09
.01	2.327
.025	1.960
.05	1.645
.10	1.282
.15	1.037
.20	.842

Another consideration might lead to a different number of classes. For example, if many times in the simulation model using the pseudo-random number generator under scrutiny a choice is made between n equally likely alternatives it might be expeditious to choose $k = n$. As Gorenstein (4) points out, in the final analysis it is up to the user to design tests to suit the needs of his simulation. There can be no general method that will guarantee good results.

C. The Kolmogorov-Smirnov Test

The χ^2 test applies to the situation where observations fall or are arbitrarily placed in a finite number of categories. It is commonplace however to consider random quantities which may assume infinitely many values, e.g., random variables on the (0, 1) interval, and for some reason be unwilling to set up arbitrary classes. By examining the cumulative distribution function we can eliminate the need of setting up arbitrary class sizes and use the Kolmogorov-Smirnov Test (K-S test).

The cumulative distribution function, denoted F(X), where

$$F(X) = \Pr\{x \leq X\}$$

indicates the probability that a random variable x is less than or equal to some given value X. In the case of a uniformly distributed variable on the unit interval (0, 1), $\Pr\{x \leq X\} = X$. For example $\Pr\{x \leq 2/3\} = 2/3$. If we made N independent observations or samplings of the random variable x, obtaining the values $x_1, x_2, \ldots x_N$ we can form the empirical cumulative distribution function

$$F_N(X) = \frac{\text{number of } x_i \leq X}{N} .$$

As N increases $F_N(X)$ should be a better and better approximation of F(X).

The K-S test may be used when F(X) has no jumps. It measures the concordance between F(X) and $F_N(X)$. A poor random number generator will yield an empirical distribution function which will not approximate F(X) very well.

To apply the K-S test to the unit interval (0, 1) where we have a sequence of N random observations we form the following two statistics:

$$K_N^+ = \sqrt{N} \max_{0 < x < 1} (F_N(X) - F(X))$$

$$K_N^- = \sqrt{N} \max_{0 < x < 1} (F(X) - F_N(X)). \qquad \text{II-6}$$

Here K_N^+ measures the greatest amount of deviation when F_N is greater than F, and K_N^- represents the maximum amount deviation when F_N is less than F.

As in the χ^2-test, we may now look up critical values of K_N^+, K_N^- in a table to determine if they are significantly high or low and thus decide if our sampled distribution does, in fact, resemble the hypothesized distribution.

Although the Kolmogorov-Smirnov test is a more statistically accurate test than the χ^2-test, there are a number of disadvantages associated with it. Some of the main disadvantages are 1) all N observations must be available during the test 2) the observations, although obtained in a random order, must be sorted in ascending order and 3) there are a considerably greater number of calculations involved in the K-S test as compared to the χ^2-test.

Runs Tests

D. The expected random oscillatory nature of sequences of pseudo-random numbers can be tested by "runs tests". Two standard types of runs tests are runs up and down and runs above and below the mean.

Runs up and down - Let $x_1, x_2, \ldots x_N$ be a sequence of N unequal numbers. Consider a sequence of N-1 signs, a_i, where a_i is the sign of $x_{i+1}-x_i$. A sequence of p consecutive plus signs not immediately followed or preceded by a plus sign is called a "run up of length p". An analogous sequence of minus signs is called a "run down of length p".

For example the sequence

1 5 19 15 13 12 18 2 4 9 11

gives ++ - - - + - + + +

which has a run up of 2 followed by a run down of 3, up of 1, down of 1, and up of 3.

If we let

r = number of runs in the sequence

r_p = number of runs of length p in the sequence

r_p^+ = number of runs of length p or more in sequence

then

$$E(r) = 1/3\ (2N-1);\ Var\ (r) = (16N-29)/90, \tag{II-7}$$

$$E(r_p) = 2N(p^2+3p+1)-2(p^3+3p^2-P-4) \quad /(p+3)! \quad \text{for } p \leq N-2, \tag{II-8}$$

$$E(r_p^+) = 2N(p+1) - 2(p^2+p-1) \quad /(p+2)! \quad \text{for } p \leq N-1 \tag{II-9}$$

and

$$\frac{r-E(r)}{\sqrt{Var\ (r)}} \tag{II-10}$$

is asymptotically normally distributed as $N \rightarrow \infty$ with mean 0 and variance 1. We, therefore, can easily test the hypothesis $H_o: r=E(r)$ by calculating (II-10) and comparing it to the value in the appropriate table of the normal distribution. Likewise, the χ^2 goodness of fit test may be used to check whether a pseudo-random number generator is acceptable based on the distribution of length of runs. A common characteristic of nonrandom sequences of numbers is an excess of long runs.

Runs above and below mean - The expected number of runs above and below the mean is

$$E(r^{(m)}) = \frac{N}{2} + 1 \tag{II-11}$$

where $r^{(m)}$ is the number of runs above and below the mean. These runs are counted by constructing a sequence of N signs with the plus or minus depending on whether X_i is greater or less than the mean of the distribution (1/2 in our case). The expected total number of runs of length p is $(N-p+3)\,2^{-k-1}$. A χ^2-test may be used to check whether a given pseudo-random number generator is acceptable.

E. Serial Tests

1) Pairs Test

For a locally random series no number shall tend to be followed by any other number. If we, therefore, construct a table with the rows representing a frequency distribution of the first number of a pair of uniform random values and the columns representing a frequency distribution of the second number of the pair we would expect the frequencies to be approximately equal in all cells after N pairs had been examined. To test this hypothesis we could apply the χ^2 test to these cells of the table with the theoretical or expected number of observations in each cell equal to N/number of cells. Clearly it would be possible to extend this test to triples, quadruples, etc.; however, the size of the table increases rapidly and to insure an expected theoretical value of at least five or ten the total number of observations needed begins to get quite large. Also, the calculations required to compute χ^2 begin to use substantial computer time.

It would be appropriate to note here that it would be a mistake to perform the serial test on the pairs (x_1, x_2), (x_2, x_3) ... (x_{2N-1}, x_{2N})

because the chi-square test requires independence of the observations. Rather, for this particular test we should use the pairs (x_1, x_2), $(x_3, x_4) \ldots (x_{2N-1}, x_{2N})$ which yields approximately half as many observations as the incorrect former method.

2) Autocorrelation

The autocorrelation function is a measure which is widely used in the study of stochastic processes. If we let x_i, $i = 1, 2, \ldots$ be a sequence from the unit interval (0, 1), then we may define the autocorrelation function of a sample of length N from this sequence as

$$R(t) = 1/12 \sum_{i=1}^{N-t} (x_i - 1/2)(x_{i+t} - 1/2) \qquad \text{(II-12)}$$

where t is commonly referred to as the length of the lag and R(t) as the autocorrelation at lag t.

The correlation coefficient always lies between -1 and +1. When it is zero or near zero, it indicates that the sequences $\{x_i\}$ and $\{x_{i+t}\}$ are (statistically speaking) independent of each other. When the correlation coefficient is near ± 1 (it indicates a high degree of linear dependence between the two sequences. A value of ± 1 would indicate total dependence and, in fact,

$$x_{i+t} = ax_i + b$$

for some constants and and b.

A satisfactory random sequence would have autocorrelations near zero for all lags tested.

F. Gap Test

All of the preceding tests have been conceived for randomness of numbers where each number consisted of some fixed or finite number of digits. The gap test for digits is concerned with the randomness of the digits in a sequence of numbers. For any given digit d, we can examine the lengths of the gaps of "non-d" digits between any two "d-digits". In other words, a gap of length k occurs when k "non-d" digits occur between two "d-digits". Two consecutive d's produce a zero gap.

The theoretical probability of obtaining a gap of length k is

$$\Pr\{\text{gap} = k\} = (.9)^k \, (.1) \qquad \text{(II-13)}$$

For a given sequence of digits, tallies are made of the number of gaps occurring for each length. A chi-square goodness of fit test can be used to compare expected and theoretical number of gaps.

A second type of gap test does not examine digits but examines the actual random number. In this case, a gap is the number of consecutive observations in the sequence that do not fall between a specified a and b. Generally, a tally is kept as to gaps of lengths 0, 1, 2 ... t-1, and the number of gaps of length t or greater.

In the case of examining pseudo-random numbers between zero and one, we would have the following relationship

$$0 \leq a < b \leq 1$$

and the following probabilities associated with the gap lengths

$$\Pr\{gap = 0\} = b-a$$

$$\Pr\{gap = 1\} = (b-a)\,(1-(b-a))$$

$$\Pr\{gap = 2\} = (b-a)\,(1-(b-a))^2$$

$$\vdots$$

$$\Pr\{gap = t-1\} = (b-a)\,(1-(b-a))^{t-1}$$

$$\Pr\{gap \geq = t\} = (1-(b-a))^t$$

Chi-square tests can also be applied here as in the digit gap test.

G. Maximum Test

For a set of N independent uniform random numbers, $x_1, x_2, \ldots x_N$, in the unit interval (0, 1), we can define a random variable $W = \text{Max}\,(x_1, x_2, \ldots x_N)$ and the distribution of W is given by

$$F\,(W) = \max\,(x_1, x_2, \ldots x_N)^N. \qquad \text{(II-14)}$$

Since $\Pr\{W < a\} = F(a) = a^N$ for $0 < a < 1$, F (w) as defined in (II-14) is distributed over the unit interval with a cumulative distribution function $F(W \leq w) = W^N$. By sampling several sets of N independent random numbers we can use the χ^2-test on the distribution of W.

H. Minimum Test

This test is the same as the maximum test of N except that the minimum of $(x_1, x_2, \ldots x_N)$ are taken and the corresponding distribution function used.

At this point it might be wise to close by answering the question as to why are so many tests necessary. It seems like more time is spent testing the numbers than in using them. This is probably not true but the importance of knowing the shortcomings of a particular random number generator cannot be understated. This is because the simulation, risk analysis or other models using the particular random number generator are highly dependent on the accuracy and unbiasedness of the generator for their value as viable tools. If confidence cannot be established in the random number generator of these models, there is little likelihood of people believing in the models that employ these generators. With confidence established in the random number generator the question of confidence in the actual model is at least reduced to the assumptions and relationships developed therein.

CHAPTER 3

A PROGRAM TO EVALUATE UNIFORM RANDOM NUMBER GENERATORS

A. Introduction

The computer program described in this section performs eight standard statistical tests on a vector of random numbers. The vector can be input to the program or the subroutine which generates the vector can be linked to the main program and the random numbers generated at execution time. The eight tests available are:

1. Gap test
2. Runs test
3. Pairs test
4. Chi-square test
5. Moments
6. Runs above/below mean
7. Autocorrelations
8. Kolmogorov-Smirnov test.

The program is written in FORTRAN IV and was compiled using the IBM FORTRAN G compiler on a System 360 Model 65. The program runs in approximately 120 K bytes of core and uses a temporary disk file to store the vector of random values. The maximum length of this vector is essentially unlimited as the program reads the random numbers into core from the temporary file in blocks of 10,000.

B. System Control Cards

The user documentation contained herein assumes that the program is available in a load module form, i.e. it has been compiled and link edited. To execute the program, therefore, the function RAN need only be compiled and linked to the main program. Although RAN will be called only if the user specifies that random numbers will be generated during execution, its module must always exist, therefore the load module contains a dummy function RAN. This function is as follows:

```
FUNCTION RAN (NX)
RAN = 0.0
RETURN
END
```

If the user is to read his random vector from an already created file the above dummy function will allow the load module to execute. If the user wishes to generate his vector of random values during execution he must compile his random number generator as a function named RAN (NSEED) and link edit this function into the load module in place of the existent dummy function. The mechanics of this procedure will of course vary from computer to computer. For an IBM/360 computer with the program load module located in an accessible library the following sequence of instructions will suffice. FORTGCLG is a catalogued procedure to execute a FORTRAN compile, link edit and go. It is comprised of three steps - FORT, LKED, and GO. TABLE III. 1 shows this procedure.

TABLE III. 1

FORTGCLG - A Procedure to Execute FORTRAN Compile, Link Edit and Go

```
 *** PROCEDURE FORTGCLG ***                              *** PROCEDURE FORTGCLG ***
//FORTGCLG   PROC   FORMS='A,,0001',CPRM=SOURCE,LPRM=LIST,
//    LIBR='UTCC.DUMMYLIB'
//****    FORTRAN G COMPILE LINK EDIT AND GO    ***
//FORT      EXEC PGM=IEYFORT,REGION=104K,TIME=10,PARM='&CPRM'
//SYSPRINT DD SYSOUT=(&FORMS),DCB=(LRECL=120,BLKSIZE=3120,RECFM=FBSA),
//    SPACE=(3120,(40,40))
//SYSLIN DD DSN=&&LOADSET,SPACE=(3120,(12,12)),DCB=BLKSIZE=3120,
//   UNIT=SYSDA,DISP=(MOD,PASS,DELETE)
//SYSIN    DD DSN=&&SOURCE,DISP=(OLD,DELETE,DELETE)
//LKED     EXEC PGM=IEWLF880,REGION=114K,COND=(4,LT,FORT),
//         TIME=2,PARM='XREF,LIST,LET,&LPRM,SIZE=(114K,24K)'
//SYSPRINT DD SYSOUT=(&FORMS),DCB=BLKSIZE=605,SPACE=(605,(17,34))
//SYSLIN   DD DSN=&&LOADSET,DISP=(OLD,DELETE,DELETE)
//            DD  DDNAME=SYSIN
//SYSLMOD   DD UNIT=SYSDA,DSN=&&GODATA(RUN),DISP=(,PASS,DELETE),
//   SPACE=(TRK,(19,10,1))
//SYSLIB     DD  DSNAME=SYS1.FORTLIB,DISP=SHR
//            DD DSN=&LIBR,DISP=SHR
//       DD  DSNAME=TEHO.LOADLIB,DISP=SHR
//            DD DSN=SYS1.GULFMOD,DISP=SHR
//SYSUT1  DD UNIT=SYSDA,SPACE=(TRK,(19,10))
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//SYSUDUMP DD SYSOUT=(&FORMS),SPACE=(TRK,(1,19))
//FT05F001 DD  DDNAME=SYSIN
//FT06F001 DD SYSOUT=(&FORMS),SPACE=(TRK,(1,19)),
//   DCB=(RECFM=VBA,LRECL=137,BLKSIZE=1100)
//FT07F001 DD SYSOUT=B,SPACE=(TRK,(1,19)),
//   DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
```

```
ccl        7
  //          EXEC FORTGCLG,REGION.GO=120K,LIBR='libr'
  //FORT.SYSIN DD *
              FUNCTION RAN (NSEED)
                  FORTRAN Source
                  code for function RAN
              RETURN
               END
  /*
  //LKED.SYSIN  DD  *
    INCLUDE SYSLIB (progname)
    ENTRY MAIN
  /*
  //GO.FT02F001  DD  UNIT=SYSDA,SPACE=(TRK,(50,10)),DISP=NEW,
  //    DCB=(RECFM=FB,LRECL=160,BLKSIZE=3200)
  //GO.SYSIN  DD  *
             Program control
                 cards
  /*
```

where in the above deck listing:

libr denotes the library where the program load module can be found,

progname denotes the name of the program.

In the Gulf Houston Datacenter, libr would be MSDC.LOADLIB and progname would be MSHØ0074.

To run the Uniform Random Number Evaluation Program where the vector of random numbers is already on a file the following sequence of cards would be used. Once again we assume the program resides on an accessible library.

```
ccl        6
  //        EXEC PGM=progname, REGION=120K
  //STEPLIB  DD  DSN=libr, DISP=SHR
  //FTnnF001  DD  UNIT=SYSDA, DSN=filename, DISP=OLD
  //FT06F001  DD  SYSOUT=A
  //FT05F001  DD  *
            Program control
                 cards
  /*
```

where in the above deck listing

progname denotes the name of the program

libr denotes the library where the program load module can be found

nn denotes the unit number of the file where the vector of random numbers is located

filename denotes the name given to this file

C. Program Control Cards

The program will evaluate, sequentially, an unlimited number of pseudo-random numbers. Each vector to be analyzed requires a single header card and then, depending on which tests are to be performed and whether or not the pseudo-random numbers have already been generated, a variable number of additional control cards.

1. Header Card

This card specifies the length of the vector to be analyzed, provides a seed if the pseudo-random numbers are to be generated during execution or the file number where the previously generated vector resides, specifies which tests are to be executed, and provides the user the option to print the vector of pseudo-random numbers. The format for the header card is as follows:

Card Column	Label	Description
1-10 (right justified)	NT	Number of random values in the vector if it is located on an already existing file or the number of values to be generated during execution using the supplied function RAN.
11-20	NSEED	Initial seed for random number generator if vector is to be generated during execution. May be left blank if vector already exists on a file.
21-22	IND(1)	Number of gap tests to be performed max=10
24	IND(2)	=0 do not perform runs test =1 perform runs test

26	IND(3)	=0 do not perform pairs test
		=1 perform pairs test
28	IND(4)	=0 do not perform chi-square test
		=1 perform chi-square test
30	IND(5)	=0 do not calculate moments
		=1 calculate moments
32	IND(6)	=0 do not perform runs above/below mean test
		=1 perform runs above/below mean test
34	IND(7)	=0 do not calculate autocorrelations
		=1 calculate autocorrelations
36	IND(8)	=0 do not perform Kolmogorov-Smirnov test
		=1 perform Kolmogorov-Smirnov test
37-40		not presently used
41-42	IFILE	Unit number of file where previously generated vector of pseudo-random numbers resides. Should be 0 or blank if numbers are to be executed at run time.
44	NPRNT	= 0 do not print pseudo-random vector
		= 1 print pseudo-random vector
45-80	TITLE(1)-TITLE(9)	A user-supplied title which will be printed on the first page of output

2. Variable Format Card

The variable format card is only read if the vector of pseudo-random numbers is to be read from an existing data file. If such is the case, this card is the FORTRAN FORMAT statement, without the statement number and word FORMAT that was used to write the previously generated vector to its data file. For example, if the file generated contains 50,000 random numbers in records of length 20 the statement that wrote these to the data file might have been

```
cc3     7                       21
   900  FORMAT(20F10.5)
```

In this instance, the variable format card would be

```
cc1           9
   (20F10.5)
```

No card should be inserted in this position if the pseudo-random vector is generated at execution time.

3. Test Parameter Cards

Certain of the statistical tests available require a control card to provide user specified parameters needed for the test. These tests are

a) the gap test

b) the pairs test

c) the chi-square test

Each time one of these tests is to be performed its parameter card is read. For the tests specified in the header card, the respective parameter cards must therefore be present. The format of these cards for each of the above tests is as follows:

Gap Test

The gap test performed by the program is the second one mentioned in Chapter 2, Section F. It tallies the number of consecutive observations in the sequence that do not fall between a user specified interval from a to b. The program has the capability of performing up to 10 simultaneous gap tests as indicated by IND(1). This number of cards (IND(1)) is needed. The format is as follows:

Gap Test Parameter Card

Card Column	Label	Description
1-5	CA	Lower end of gap interval
6-10	CB	Higher end of interval
14-15	MXGAP	Maximum number of cells to record gap length; 0, 1, 2, . . . MXGAP-1. MXGAP $\leq$ 10

CA must be less than CB. Also, to insure a meaningful chi-square test on the distribution of gap lengths

$$NT*(CB-CA)^2(1-(CB-CA))^{MXGAP} \geq 5.$$

Pairs Test

The pairs test tallies adjacent pairs of pseudo-random numbers in a two-dimensional frequency table where the horizontal and vertical axes are divided into a user specified number of categories, say NCP. Thus there are NCP^2 total cells. The user can specify NCP or the program will calculate a desirable value using the Mann-Wald criterion described earlier. In either instance, however, a Pairs Test Parameter Card must be present if the Pairs Test has been specified. Its format is as follows:

Pairs Test Parameter Card

Card Column	Label	Description
1-3	NCP	Number of cells to appear on horizontal and vertical axes. Maximum 50. =0 program will calculate NCP using Mann-Wald criterion.

Chi-Square Test

The user must provide the number of cells for the tallying of the frequency distribution for the Chi-square test. As with the pairs test mentioned above, if no explicit specification is made the program will calculate the number of cells using the Mann-Wald criterion. The format for the Chi-square

Test Parameter Card is as follows:

Chi-Square Test Parameter Card

Card Column	Label	Description
1-3	NC	Number of cells for tally of frequency distribution. Maximum 500. =0 program will calculate NC using Mann-Wald criterion.

If the gap test, pairs test or chi-square test is not requested on the header card by the appropriate IND(i), its corresponding parameter card must not appear.

D. Output

The standard output from the Uniform Random Number Evaluation includes the following:

1. the number of observations in the vector of pseudo-random numbers
2. the user supplied seed if the sequence was generated during execution
3. a listing of the random sequence (optional).

Output for each of the tests available in the program is as follows:

Gap Test

- the user specified gap interval
- the frequency distribution of observed and theoretical gap lengths from length 0 to MXGAP-1 and over

- the calculated χ^2 statistic and associated degrees of freedom
- the critical χ^2 value for 90% and 95% confidence levels

Runs Test

- the frequency distribution of observed and theoretical run lengths
- the calculated χ^2 statistic and associated degrees of freedom
- the critical χ^2 value for 90% and 95% confidence levels
- the Z-score for the test of the hypothesis $H_o : r = E(r)$

Pairs Test

- the number of intervals the horizontal and vertical axes have been divided into
- the end points of each interval and the frequency count of adjacent pairs in each grid
- the calculated χ^2 value and its associated degrees of freedom
- the critical χ^2 value for the 90% and 95% confidence levels

Chi-Square Test

- for each cell the interval end points, observed and theoretical frequency counts
- the calculated χ^2 value and its associated degrees of freedom
- the critical χ^2 value for the 90% and 95% confidence levels

Moments

- the calculated and theoretical mean, second moment, third moment and variance

- the normalized deviate of the observed mean from the theoretical mean of .5

Runs Above/Below Mean

- the frequency distribution of observed and theoretical run lengths
- the calculated χ^2 statistic and its associated degrees of freedom
- the critical χ^2 value for 90% and 95% confidence intervals

Autocorrelations

- the calculated autocorrelations of the series for all lags up to the minimum of 50 and NT/10
- the 95% confidence interval for the theoretical autocorrelations of zero
- the number of observed autocorrelations which fall outside the 95% limits

Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test involves sorting the vector of pseudo-random numbers. Since the vector is handled in blocks of 10,000, each block of 10,000 is sorted and the test performed on the sorted vector of length 10,000. The results are shown for each block up to twenty, or 200,000 random numbers and are as follows

- the maximum and minimum (ZP and ZM) deviations of the observed and theoretical distributions
- the probability of observing a ZP less than the one realized
- the probability of observing a ZM greater than the one realized

E. The Program and Necessary Subroutines

The Uniform Random Number Evaluations Program consists of a main program and fifteen functions and subroutines. It requires 114 K bytes of core on an IBM/360 Model 65. A run to generate 50,000 uniform random numbers and perform all eight tests uses approximately 4 1/2 CPU minutes of which the generation of the random numbers uses about half of the total time.

The main program acts as a control module among the subroutines. It performs necessary bookkeeping as well as all input/output. The subroutines are the modules that perform all the statistical tests and they are called by the main program and by other subroutines. A list of the functions and subroutines, their calling sequence and a brief description are shown below.

GAP2 (called by MAIN) - tallies gap lengths in the random vector and records results in appropriate table.

CHISQ (called by MAIN, RUNTS) - calculates the chi-square statistic and degrees of freedom for a pair of observed and theoretical frequencies.

TALLY (called by MAIN) - tallies vector of observations into a frequency distribution.

MOMNT (called by MAIN) - calculates mean, variance, second and third moments of a vector.

KOLMO (called by MAIN) - sorts vector into ascending order and finds the maximum and minimum deviations between the empirical and theoretical distributions.

RUNTL (called by MAIN) - tallies the number of runs up and down in a vector.

RUNTS (called by MAIN) - computes the theoretical frequencies of runs up and down and calculates the χ^2 statistic (using CHISQ).

PARTL (called by MAIN) - tallies the occurrence of the adjacent paired coordinates in a vector.

PARTS (called by MAIN) - performs the Pairs Test on the frequency of pairs tallied in PARTL.

CHSQD (called by MAIN) - calculates the critical chi-square value for an alpha level and a given degrees of freedom

GAUSD (called by CHSQD) - calculates the deviate associated with the cumulative probability of a normal distribution.

AUTOC (called by MAIN) - calculates autocorrelations in a vector for up to fifty lags.

SMIRN (called by MAIN) - computes the limiting distribution function of the Kolmogorov-Smirnov statistic.

MWALD (called by MAIN) - computes the optimal number of classes for a chi-square test according to the Mann-Wald criterion.

RAN (called by MAIN) - a user supplied function to generate uniform pseudo-random numbers.

A listing of the entire program and all functions and subroutines is shown in Appendix A.

CHAPTER 4

EVALUATING SELECTED UNIFORM RANDOM NUMBER GENERATORS

A. Introduction

In this chapter, the results of applying the subject program to a number of frequently used uniform pseudo-random number generators are presented and discussed. Additionally the subject program was run on "random" sequences from three intentionally biased generators. A summary of the output from these tests is presented, along with relevant remarks. A complete set of the output reports is available upon request. A sample output report is shown in Appendix B.

B. The Tested Random Number Generators

As previously mentioned, four frequently used uniform pseudo-random number generators were tested. The four selected and a brief description of each is shown below:

1. RAN - A function coded in FORTRAN supplied by Dr. C. E. Donaghey in the class I. E. 670, Operations Research - Digital Simulation, Fall 1972. The validity of the generator is supposedly machine independent. The code for RAN is as follows:

```
FUNCTION RAN (NSEED)
NSEED = IABS (NSEED * 655393)
RAN = FLOAT (MOD(NSEED, 33554432) ) / FLOAT (33554432)
RETURN
END
```

2. RANDU - A FORTRAN subroutine presented in the IBM Scientific Subroutine Package, p. 77. RANDU is used in PETROS, originally an IBM simulation game of the oil industry, which has since been substantially modified and improved by Gulf and is periodically presented to Gulf management as part of an executive training seminar. The subroutine coding for RANDU is as follows:

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

In its above form as a subroutine, RANDU, when used in conjunction with the evaluation program, would have to generate its sequence of random values externally to the test program. It would be appropriate to note here that with a few minor changes however, RANDU could be converted to a function program and the random sequence generated during execution of the test program. The converted subroutine (renamed RAN, as required) would be as follows:

```
  FUNCTION RAN (IX)
  IY = IX * 65539
  IF (IY) 5,6,6
5 IY = IY + 2147483647 + 1
6 RAN = IY
  IX = IY
  RAN = RAN * .4656613E-9
  RETURN
  END
```

3. GGU1 - An assembler language uniform pseudo-random number generator developed and distributed by IMSL (International Mathematical and Statistical Libraries, Inc.). GGU1 generates a sequence, $\{R\}$ of uniformly distributed numbers using a multiplier and a seed

   where:

$$R_{i+1} = A * R_i \qquad i = 0, 1, 2 \ldots$$

   where:

   A is a constant initialized in GGU1,

   R_o is the input seed, a floating point number in the interval (0, 1).

4. GGU2 - An assembler language uniform pseudo-random number generator developed and distributed by IMSL. GGU2 is similar to GGU1, except that two multipliers and two seeds are used in the former whereas GGU1 uses a single multiplier and seed. In GGU2,

each seed-multiplier continuation is used to produce a floating point deviate. The resulting random deviate is built using the characteristic (exponent) of one of the original deviates and "Exclusive OR" ing the two mantissas, securing, in a random manner that the resultant lies in the interval (0, 1).

Each of the above random number generators was used to generate random sequences of length 1000, 2500 and 5000. The initial seed(s) for each of the twelve runs (4 generators, 3 runs each) were selected from a table of random numbers. All tests available in the program were executed on each sequence. The results are summarized in Tables IV. 1 and IV. 2.

Generally speaking, all the generators did fairly well with respect to these tests. The one or two significant χ^2 values for the gap tests is not abnormal for the 30 tests on each generator as it only represents about a 5% incidence of failure. Both IMSL routines fared poorly on the Runs Test, exhibiting an excessive number of runs in the sequence of length 1000. RANDU revealed a poor distribution of run lengths above and below the mean for its sequence of length 1000.

All of the tests were identically specified with respect to the available user parameters. The Mann-Wald criterion was used for selecting the number of cells in the Pairs Test and Chi-square Test. The gap tests evaluated ten gap intervals from (. 0, . 1) through (. 9, 1.) It would be advisable for an analyst considering using one of these pseudo-random

TABLE IV.1
RESULTS OF GAP TESTS

GENERATOR

	RAN			RANDU			GGU1			GGU2		
Length of Sequence (M)	1	2.5	5	1	2.5	5	1	2.5	5	1	2.5	5
Gap Interval												
(.0 - .1)	.6	8.5	5.4	5.6	14.9*	3.9	2.0	7.1	14.5	4.6	9.6	6.6
(.1 - .2)	1.7	11.5	4.6	4.0	9.3	3.7	2.7	8.4	8.1	5.8	8.3	9.9
(.2 - .3)	3.6	8.7	6.5	3.5	11.3	6.1	2.1	7.1	10.7	2.1	9.1	10.7
(.3 - .4)	3.5	5.5	6.4	3.6	6.6	4.8	.9	16.5*	5.9	5.2	13.8	8.2
(.4 - .5)	4.7	7.4	3.9	.6	10.0	8.6	4.5	2.0	4.1	8.3*	7.1	7.1
(.5 - .6)	2.1	6.7	5.6	3.3	8.2	15.2*	3.5	10.3	13.6	3.4	5.7	13.5
(.6 - .7)	2.3	12.0	6.5	.0	12.5	6.8	9.0*	8.2	5.6	3.1	9.7	6.2
(.7 - .8)	2.4	8.4	11.7	4.9	4.5	7.6	2.1	3.2	13.3	2.6	6.1	4.9
(.8 - .9)	8.1*	5.7	7.2	5.1	10.5	10.6	2.6	9.6	8.4	2.1	16.8*	6.3
(.9 - 1.0)	3.0	4.3	10.0	5.3	7.4	11.0	3.7	7.6	13.0	2.8	6.4	3.6
Critical χ^2												
Alpha = .05	9.49	16.92	16.92	9.49	16.92	16.92	9.49	16.92	16.92	9.49	16.92	16.92
Alpha = .10	7.78	14.68	14.68	7.78	14.68	14.68	7.78	14.68	14.68	7.78	14.68	14.68

* Significant χ^2 value at 90% confidence level

TABLE IV.2

SUMMARY OF STATISTICAL TESTS ON FOUR UNIFORM PSEUDO-RANDOM NUMBER GENERATORS

GENERATOR

	RAN			RANDU			GGU1			GGU2		
Length of Sequence (M)	1	2.5	5	1	2.5	5	1	2.5	5	1	2.5	5
Test												
1) Runs Test - χ^2	1.0	-	3.5	1.5	-	6.5	12.8**	-	3.7	8.9*	-	4.3
Critical values												
(α = .05)	9.5		11.1	9.5		11.1	9.5		11.1	9.5		11.1
(α = .10)	7.8		9.2	7.8		9.2	7.8		9.2	7.8		9.2
2) Runs Test - Z-score	-.6	-1.2	.0	.5	-.6	.1	-1.3	-.2	.8	1.5	.17	-1.0
Critical values												
(α = .05)	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96
(α = .10)	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65
3) Pairs Test - χ^2	54.1	94.2	109.2	45.5	91.2	100.1	36.7	91.6	104.6	40.4	80.1	82.7
Critical values												
(α = .05)	65.2	101.8	123.2	65.2	101.8	123.2	65.2	101.8	123.2	65.2	101.8	123.2
(α = .10)	60.9	96.6	117.4	60.9	96.6	117.4	60.9	96.6	117.4	60.9	96.6	117.4
4) Chi-Square Test - χ^2	42.7	65.4	115.0	45.5	88.1	73.6	53.3	72.8	98.6	48.4	94.3	130.0
Critical values												
(α = .05)	76.8	107.5	137.7	76.8	107.5	137.7	76.8	107.5	137.7	76.8	107.5	137.7
(α = .10)	72.2	102.1	131.6	72.2	102.1	131.6	72.2	102.1	131.6	72.2	102.1	131.6

TABLE IV.2 (Cont'd)

SUMMARY OF STATISTICAL TESTS ON FOUR UNIFORM PSEUDO-RANDOM NUMBER GENERATORS

	GENERATOR											
	RAN			RANDU			GGU1			GGU2		
Length of Sequence (M)	1	2.5	5	1	2.5	5	1	2.5	5	1	2.5	5
Test												
5) Moments - Z - score of mean	.7	.6	-.8	-.5	1.24	-.3	1.19	1.12	.3	.3	-1.8*	.2
Critical values ($\alpha = .05$)	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96
($\alpha = .10$)	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65	1.65
6) Runs Above/ Below Mean - χ^2	10.4	2.9	5.8	11.8*	5.0	9.7	4.1	7.3	6.4	5.1	9.1	4.0
Critical Values ($\alpha = .05$)	12.6	15.5	16.9	12.6	15.5	16.9	12.6	15.5	16.9	12.6	15.5	16.9
($\alpha = .10$)	10.7	13.4	14.7	10.7	13.4	14.7	10.7	13.4	14.7	10.7	13.4	14.7
7) Autocorrelations - % Outside $\pm$ 95% limits	8*	2	8*	2	0	2	6	2	4	12*	6	4
8) Kolmogorov-Smirnov Test - Max (Prob (ZP), Prob (ZM))	.74	.18	.73	.34	.77	.02	.49	.82	.07	.32	.88	.24

* statistically significant difference at 90% level

** statistically significant difference at 95% level number generators for a specific application, to retest the generator with the characteristics of that application in mind. Particulars to consider would be the number and length of gap intervals, the number of cells in the Chi-square and Pairs Test, the initial seed, NSEED, and the length of the random vector. These factors would obviously have a direct bearing on the meaningfulness of the tests in relationship to the validity of the simulation.

C. Intentionally Biased Generators

As mentioned above, the evaluation program was also run using three biased random number generators. The function RAN referred to earlier in this chapter was modified to produce non-random sequences of length 5000. The three biased generators can be characterized as follows:

1. Correlated random numbers - each generated pseudo-random number in the sequence was correlated with the previously generated number using the relationship

$$x_{i+1} = .3x_i + .7 \text{ RAN (NSEED)} \qquad i = 1,2, \ldots 4999$$

where

RAN (NSEED) is the call to the function RAN described in Section B of this chapter.

2. Gap between .8 and .85 - random values were generated by RAN with all values in the interval (.8, .85) ignored. The resultant vector thus consisted of 5000 values of which none were in the mentioned interval.

3. Periodicity length of 1000 - RAN was used to generate 1000 pseudo-random values. This sequence was then replicated four times and appended to itself to yield a vector of 5000 pseudo-random values having a periodicity or cycle length of 1000.

The summarized results of these three tests are shown in Table IV. 3. Critical values are not shown in this Table, but are identical to the critical values shown in Table IV. 2 for the sequences of length 5000.

The program does quite well in the detection of the aberrated pseudo-random number generators. The correlated random number generator shows significant differences in all tests, except the Moments Test Z - score for the mean. However, although not shown in Table IV. 3, the variance and second and third moments of this sequence are showing substantial difference from their respective expected values. With 8% of the autocorrelations significant, it is also interesting to note the value of the autocorrelation for the first lag is .296. The values for lags two and three are .098 and .019 which exhibit the pattern of autocorrelations from an autoregressive time series. The "gapped" generator shows its aberrations exceptionally well in the Pairs Test and Chi-square Test. The gap in the interval (.8, .85) lowers the mean significantly and also causes suspicious results in the Runs Above/Below the Mean and the Kolmogorov-Smirnov Tests. The cycled generator shows significant χ^2 values in all Gap Tests, the Runs Test, Pairs Test, Chi-square Test and Runs Above/Below the Mean. It also shows statistical significance

TABLE IV.3

RESULTS OF TESTS ON BIASED GENERATORS

	GENERATOR		
	Correlated	Gap at (.8, .85)	Cycle length 1000
1) % significant χ^2 values for maximum of 10 Gap Tests (α = .05)	100%	10%	100%
(α = .10)	100%	30%	100%
2) Runs Test - χ^2	118.9**	2.9	25.4**
3) Runs Test - Z - score	-8.6**	-1.0	1.4
4) Pairs Test - χ^2	1939.2**	234.7**	570.**
5) Chi-Square Test - χ^2	1531.8**	341.7**	626.3**
6) Moments - Z - score of mean	.50	-4.96**	1.34
7) Runs Above/Below Mean - χ^2	134.2**	24.1*	48.9**
8) Autocorrelations - % outside $\pm$ 95% limits	8*	14*	28**
9) Kolmogorov-Smirnov Test - Max (Prob (ZP), Prob (ZM))	1.00**	1.00**	.99**

*   statistically significant difference at 90% level

** statistically significant difference at 95% level with the high proportion of non-zero autocorrelations and a very small K-S probability of actually representing its supposed theoretical distribution.

D. Conclusions

The subject program to evaluate uniform pseudo-random number generators provides a consistent yet flexible tool for the analysis of computer generated random sequences. Additional areas of study relating to and expanding upon the work done to this point could prove to be interesting and informative.

The program might be used to evaluate additional uniform random number generators that are frequently used. This could be done as a general comparative test, similar to the ones described in this paper, or as a specific test with a particular application of the random number generator in mind and the tests parameters selected for that one application.

A study to determine the sensitivity of the program and define its discriminatory ability with regard to valid and invalid generators would be useful. This study could proceed by sequentially altering a valid pseudo-random number generator with more subtle aberrations until the program was no longer able to distinguish the biased generator from the presumably unbiased one. The various tests included in the program could be ranked according to their ability to detect defective generators and which tests are most likely to detect certain common deficiencies (e.g. autocorrelation, cycling, subtle patterns in the generated sequence).

The area of periodicity is one in which additional study might prove to be especially revealing. The theory presented in Chapter 1 detailing the relationships between multipliers, seeds and various congruential methods could be taken further, using the subject program as a means of evaluating possible alternatives. Along the lines of the cycling generator presented earlier in this chapter, the ratio of cycle length to length of the evaluated sequence might be varied to determine at what point cycling becomes apparent.

All of the above areas of additional study propose use of the subject program as it now exists. There are of course, a number of possible enhancements to the program which might also be considered. There are a number of additional tests which could be included in the program such as the maximum test, minimum test, poker test, triplets test and distance test. An option for the user to code his own test (called UTEST, for example) and link edit the code to the main program in a manner like RAN is now handled would be a valuable feature and would give the program virtually total flexibility.

BIBLIOGRAPHY

1. Coveyou, R. R. and Macpherson, R. D. "Fourier Analysis of Uniform Random Number Generators", Journal of the ACM, Vol. 14, No. 1 (Jan. 1967), pp. 100-119.

2. Cunningham, S. W. "From Normal Integral to Deviate", Applied Statistics, Vol. 18, Royal Statistical Society, 1969, pp. 290-293.

3. Goldstein, Richard B. "Chi-square Quantiles", Comm. of the ACM, Vol. 16, No. 8 (Aug. 1973), pp. 483-485.

4. Gorenstein, Samuel. "Testing a Random Number Generator", Comm. of the ACM, Vol. 10, No. 2 (Feb. 1967), pp. 111-118.

5. Hastings, Cecil, Jr. Approximations for Digital Computers, Princeton University Press, Princeton, N. J., 1957; p. 192.

6. IBM Corporation. System/360 Scientific Subroutine Package (360A-CM-03X) Version III Programmer's Manual, H20-0205-3, IBM Technical Publications Department, White Plains, N. Y., 1968.

7. Kendall, M. G. and Babington-Smith, B. "Randomness and Random Sampling Numbers", Journal of the Royal Statistical Society, Vol. 101 (1938), pp. 147-166.

8. Kendall, M. G. and Babington-Smith, B. "Second Paper on Random Sampling Numbers", Supplement to the Journal of the Royal Statistical Society, Vol. 6 (1939), pp. 51-61.

9. Knuth, Donald E. The Art of Computer Programming, Addison-Wesley, Reading, Mass., 1969.

10. Levene, H. and Wolfowitz, J. "The Covariance Matrix of Runs Up and Down", Ann. Math. Stat., Vol. 17 (1944), pp. 58-69.

11. MacLaren, M. Donald and Marsaglia, G. "Uniform Random Number Generators", Journal of the ACM, Vol. 12, No. 1 (Jan. 1965), pp. 83-89.

12. Mann, H. B. and Wald, A. "On the Choice of the Number of Class Intervals in the Application of the Chi-square Test", Ann. Math. Stat., Vol. 13 (1942), pp. 306-317.

13. Marsaglia, G. "Random Numbers Fall Mainly in the Planes", Procedures of the National Academy of Science, Vol. 60 (Sept. 1968), pp. 25-28.

14. Marsaglia, G. and Bray, T. A. "One Line Random Number Generators and Their Use in Combinations", Comm. of the ACM, Vol. 11, No. 11 (Nov. 1968), pp. 757-759.

15. Naylor, Thomas H. Computer Simulation Experiments with Models of Economic Systems, John Wiley & Sons, Inc., New York, 1971.

16. Olmstead, P. S. "Distribution of Sample Arrangements for Runs Up and Down", Ann. Math. Stat., Vol. 17 (1946), pp. 24-33.

17. Tippett, L. H. C. "Random Sampling Numbers", Tracts for Computers, Vol. 15 (1927), Cambridge University Press.

18. Westlake, W. J. "A Uniform Random Number Generator Based on the Combination of Two Congruential Generators", Journal of the ACM, Vol. 14, No. 2 (April 1967), pp. 337-340.

APPENDIX A

Source Code for the Program

```
C          DATA SET MSHOAEVRAN AT LEVEL 039 AS OF 06/09/74
      DIMENSION X(10000),KOUNT(10),CELLS(500),EXPCT(500),ANS(4)
      DIMENSION A(50,50),IND(10),IFOMT(20)
      DIMENSION CELLG(10),RUNS(8)
      DIMENSION KNTMN(10),AC(50),ZP(20),ZM(20)
      DIMENSION CELGP(10,10),CAG(10),CBG(10),MBG(10)
      DIMENSION KNTSV(10),TITLE(9)
      DATA IFOMT /'(20F','8.5)',18*'    '/
C
C     WHICH TESTS ARE  TO BE PERFORMED ARE SPECIFIED
C          BY IND(I) = 0  OR  1   WHERE
C            I = 1  GAP TEST
C            I = 2  RUNS TEST
C            I = 3  PAIRS TEST
C            I = 4  CHI-SCUARE TEST
C            I = 5  MOMENTS
C            I = 6  RUNS ABOVE/BELOW MEAN (GAP TEST ON (0,.5))
C            I = 7  AUTOCORRELATIONS
C            I = ,  KOLMOGOROV - SMIRNOV TEST
C
C     IFILE = UNIT NUMBER OF DATA SET WHERE RANDOM VALUES
C             ARE LOCATED
C
C           = 0  FOR NUMBERS TO BE GENERATED BY USER
C                SUPPLIED FUNCTION RAN
C
    1 READ(5,900,END=89000) NT,NSEED,IND,IFILE,NPRNT,TITLE
  900 FORMAT(2I10,12I2,9A4)
C
C     SET SYSTEM LIMITS
C
      MAXRN=10000
      KIN=5
      KOUT=6
      MXCLS=500
      MAXA=50
      MAXKN=10
      WRITE(KOUT,9001) TITLE,NT,NSEED
 9001 FORMAT('1RANDOM NUMBER EVALUATION',5X,9A4///'    NT =',I10/' NSEE
     1 =',I10)
      WRITE(KOUT,9002)
 9002 FORMAT(/////' TESTS REQUESTED')
      IF (IND(1) .GE. 1) WRITE(KOUT,9003) IND(1)
      IF(IND(2) .EQ. 1) WRITE(KOUT,9004)
      IF(IND(3) .EQ. 1) WRITE(KOUT,9005)
      IF(IND(4) .EQ. 1) WRITE(KOUT,9006)
      IF(IND(5) .EQ. 1) WRITE(KOUT,9007)
      IF(IND(6) .EQ. 1) WRITE(KOUT,9009)
      IF(IND(7) .EQ. 1) WRITE(KOUT,90091)
      IF(IND(8) .EQ. 1) WRITE(KOUT,9008)
 9009 FORMAT('    RUNS ABOVE/BELOW MEAN')
90091 FORMAT('    AUTOCORRELATIONS')
 9003 FORMAT('    GAP TESTS - ',I2)
 9004 FORMAT('    RUNS TEST')
 9005 FORMAT('    PAIRS TEST')
 9006 FORMAT('    CHI - SQUARE TEST')
 9007 FORMAT('    MOMENTS')
 9008 FORMAT('    KOLMOGOROV  - SMIRNOV TEST')
C
C     WILL ALL RANDOM NUMBERS FIT IN CORE?
```

```
C     ESTABLISH NUMBER OF ITERATIONS NECESSARY
C
      NITER= (NT-1)/ MAXRN + 1
C
C
C
C     IS DATA GENERATED BY RAN, IF SO PLACE
C     ON UNIT 2, PRINT DATA IF REQUESTED
C
      IEND=MAXRN
      DO 50 I=1,NITER
      IF( I .EQ. NITER)  IEND= NT- (I-1) * MAXRN
      IF( IFILE .NE. 0) GO TO 20
      DO 10 J=1,IEND
   10 X(J)=RAN(NSEED)
      WRITE( 2   ,IFOMT) (X(J),J=1,IEND)
      GO TO 40
   20 IF(I .EQ.1) READ(KIN,9010) IFOMT
 9010 FORMAT(20A4)
   40 IF(NPRNT .EQ. 0) GO TO 50
      IF(IFILE .GT. 0) READ(IFILE,IFOMT,END=45) X
   45 IF(I.EQ. 1) WRITE(KOUT,901) NT
  901 FORMAT(1H1,I10,' RANDOM VALUES')
      WRITE(KOUT,902) (X(K),K=1,IEND)
  902 FORMAT(1X,15F8.5)
   50 CONTINUE
C
C     BEGIN TESTS
C
      IF(IFILE .EQ.0) IFILE=2
      REWIND IFILE
      N=MAXRN
      DO 1000 I=1,NITER
      IF(I .EQ.NITER) N=NT-(I-1)*MAXRN
      READ(IFILE,IFOMT) (X(K),K=1,N)
C
C     GAP TEST
C
   60 IF(IND(1) .LE. 0 ) GO TO 100
      IF( I.GT. 1) GO TO 65
      NGT=IND(1)
      DO 61 J=1,NGT
   61 READ(KIN,903) CAG(J),CBG(J),MBG(J)
  903 FORMAT(2F5.0,I5)
      DO 63 K1=1,10
      DO 63 K2=1,10
   63 CELGP(K1,K2)=0.
   65 DO 90 KK=1,NGT
      CA=CAG(KK)
      CB=CBG(KK)
      MXGAP=MBG(KK)
      IF(CA .LT. CB) GO TO 70
      WRITE(KOUT,904)
  904 FORMAT('1GAP TEST',/'   * * * INPUT ERROR * * *')
      WRITE(KOUT,905) CA,CB
  905 FORMAT('    CA .GT.CB'//' CA =',F8.5,'  CB =',F8.5)
      IND(1)=-1
      GO TO 100
   70 IF (MXGAP .LE. 10) GO TO 80
      WRITE(KOUT,904)
```

```
      WRITE(KOUT,906)
  906 FORMAT('   MXGAP .GT. 10 ---SET TO 10')
      MXGAP=10
   80 DO 82 K=1,MXGAP
   82 KOUNT(K)=0
      CALL GAP2(N,KOUNT,MXGAP,X,CA,CB,I,KNTSV(KK))
      DO 83 K=1,MXGAP
   83 CELGP(KK,K)=CELGP(KK,K)+KOUNT(K)
      IF(I.NE. NITER) GO TO 90
      XTOT=0.
      DO 84 K=1,MXGAP
      XTOT=XTOT+CELGP(KK,K)
   84 CELLG(K)=CELGP(KK,K)
      WRITE(KOUT,9061) CA,CB
 9061 FORMAT('1GAP TEST'//'  GAP INTERVAL =(',F7.4,',',F7.4,')')
      WRITE(KOUT,9062)
 9062 FORMAT(//5X,'GAP LENGTH',7X,'OBSERVED',4X,'THEORETICAL')
      CBA=CB-CA
      PROB=CBA
      TP=0.
      DO 87 K=1,MXGAP
      K1=K-1
      IF( K.LT. MXGAP) GO TO 85
      PROB=1.-TP
   85 TP=TP+PROB
      EXPCT(K)=XTOT*PROB
      WRITE(KOUT,9063) K1,CELLG(K),EXPCT(K)
 9063 FORMAT(I15,F15.0,F15.3)
      IF(K.EQ.MXGAP) WRITE(KOUT, 9064)
 9064 FORMAT('+',T16,'+')
      PROB=PROB*(1.-CBA)
   87 CONTINUE
      WRITE(KOUT,9065) XTOT
 9065 FORMAT(10X,'TOTAL',F15.0)
      CALL CHISQ(CELLG,EXPCT,MXGAP,CS,IDF)
      CV05=CHSQD(.05,IDF)
      CV10=CHSQD(.10,IDF)
      WRITE(KOUT,9066) IDF,CS
      WRITE(KOUT,9067) CV05,CV10
 9066 FORMAT(//' CHI - SQUARE(',I4,' DF) =',F9.2)
 9067 FORMAT(//' CRITICAL VALUE (ALPHA=.05) =',F9.2,
     1  /' CRITICAL VALUE (ALPHA=.10) =',F9.2)
      IF(IDF .EQ. 0) WRITE(KOUT,9068)
 9068 FORMAT(///' * * * *  NO CHI - SQUARE TEST CALCULATED * * * *'
     1 /' ONE EXPECTED CELL COUNT .LT. 1 OR THREE EXPECTED COUNTS .LT.
     2')
   90 CONTINUE
C
C     RUNS TEST
C
  100 IF(IND(2) .EQ. 0) GO TO 200
      IF( I .GT. 1) GO TO 110
      NVAL=4
      IF( NT .GT. 500) NVAL=5
      IF( NT .GT. 1000) NVAL=6
      IF( NT .GT. 25000) NVAL=7
  110 CALL RUNTL (X,N,RUNS,I,NVAL)
      IF(I .NE. NITER) GO TO 200
      CALL RUNTS (RUNS,NT,EXPCT,CS,IDF,NVAL)
      WRITE(KOUT,9071)
```

```
      9071 FORMAT('1RUNS TEST'//5X,'RUN LENGTH',7X,'OBSERVED',4X,'THEORETICA
          1')
           XTOTO=0.
           XTOTT=0.
           DO 120 K=1,NVAL
           XTOTO=XTOTO+RUNS(K)
           XTOTT=XTOTT+EXPCT(K)
           WRITE(KOUT,9072) K,RUNS(K),EXPCT(K)
      9072 FORMAT(I15,F15.0,F15.3)
           IF ( K.EQ. NVAL) WRITE(KOUT,9073) XTOTO,XTOTT
      9073 FORMAT('+',T16,'+'/10X,'TOTAL',F15.0,F15.3)
       120 CONTINUE
           CV05=CHSQD (.05,IDF)
           CV10=CHSQD (.10,IDF)
           WRITE(KOUT,9066) IDF,CS
           WRITE(KOUT,9067) CV05,CV10
           IF(IDF .EQ. 0) WRITE(KOUT,9068)
           VAR=(16.*NT-29.)/90.
           Z=(XTOTO-XTOTT)/SQRT(VAR)
           WRITE(KOUT,9074) Z
      9074 FORMAT(///' Z - SCORE (TOTAL RUNS) =',F8.2)
     C
     C     PAIRS TEST
     C
       200 IF(IND(3) .EQ. 0) GO TO 300
           IF(I .EQ.1) READ(KIN,907) NCP
       907 FORMAT(I3,2F10.0)
           IF(NCP .EQ. 0) NCP=SQRT(FLOAT(MWALD(NT)))
           IF(NCP .LE.50) GO TO 220
           WRITE(KOUT,908)
       908 FORMAT('1PAIRS TEST',/'  * * * INPUT ERROR * * *')
           WRITE(KOUT,909)
       909 FORMAT(' NCP .GT. 50 --- SET TO 50')
           NCP=50
       220 CALL PARTL(X,N,NCP,A,MAXA,I)
           IF(I .NE. NITER) GO TO 300
           CALL PARTS(A,MAXA,NCP,CS,STDCS)
           WRITE(KOUT,9091) NCP
      9091 FORMAT('1PAIRS TEST'//'  NO. OF INTERVALS =',I3)
           XST=0.
           XINT=1./NCP
           LS=1
           LF=NCP
           IF(NCP .GE. 11) LF=10
       240 WRITE(KOUT,9092) (JP,JP=LS,LF)
      9092 FORMAT(' INTERVAL   FROM - TO    ',10I8)
           DO 250 K=1,NCP
           XFN=XST+XINT
           WRITE(KOUT,9093) K,XST,XFN,(A(K,J),J=LS,LF)
      9093 FORMAT(I9,1X,2F7.4,10F8.0)
           XST=XFN
       250 CONTINUE
           IF(LF .EQ. NCP) GO TO 270
           LS=LF+1
           LF=LF+10
           IF(LF .GT.NCP) LF=NCP
           XST=0.
           WRITE(KOUT,9094)
      9094 FORMAT('1')
           GO TO 240
```

```
  270 IDF=NCP*NCP-1
      WRITE(KOUT,9066) IDF,CS
      CV10=CHSQD(.10,IDF)
      CV05=CHSQD(.05,IDF)
      WRITE(KOUT,9067) CV05,CV10
C
C
  300 IF (IND(4) .LE. 0) GO TO 500
C
C     CHI-SQUARE TEST
C
      IF (I .EQ. 1) READ(KIN,907) NC
      IF(NC .EQ. 0) NC=MWALD(NT)
      IF(NC .LE. MXCLS) GO TO 310
      WRITE(KOUT,910)
  910 FORMAT('1CHI-SQUARE TEST'/'  * * * INPUT ERROR * * *')
      WRITE(KOUT,911)
  911 FORMAT(' NC .GT. 500 --- SET TO 500')
      NC=MXCLS
  310 XM=1./NC
      XD=XM
      CALL TALLY(X,N,CELLS,NC,XM,XD,I)
      IF(I .NE. NITER) GO TO 400
      DO 330 KK=1,NC
  330 EXPCT(KK)=NT*XD
C
      CALL CHISQ(CELLS,EXPCT,NC,CS,IDF)
      WRITE(KOUT,9111)
 9111 FORMAT('1CHI - SQUARE TEST'//' INTERVAL   FROM -  TO   OBSERVED
     1HEORETICAL')
      XST=0.
      XINT=1./NC
      XTOT=0.
      DO 360 K=1,NC
      XTOT=XTOT+CELLS(K)
      XFN=XST+XINT
      WRITE(KOUT,9112) K,XST,XFN,CELLS(K),EXPCT(K)
 9112 FORMAT(I9,2F7.4,F10.0,F12.2)
      XST=XFN
  360 CONTINUE
      WRITE(KOUT,9113) XTOT
 9113 FORMAT(4X,'TOTAL',14X,F8.0)
      WRITE(KOUT,9066) IDF,CS
      CV05=CHSQD(.05,IDF)
      CV10=CHSQD(.10,IDF)
      WRITE(KOUT,9067) CV05,CV10
      IF(IDF .EQ. 0) WRITE(KOUT,9068)
C
C
  400 IF(IND(5) .EQ.0) GO TO 500
      IC=2
      IF(I .EQ.NITER) IC=3
      IF(I .EQ.1) IC=1
      IF(NITER .EQ.1) IC=4
      CALL MOMNT(X,N,ANS,IC)
      IF(IC .LT.3) GO TO 500
      WRITE(KOUT,912)
  912 FORMAT('1MOMENTS'//22X,'OBSERVED  THEORETICAL')
      WRITE(KOUT,913) ANS
  913 FORMAT(016X,'MEAN',F10.4,8X,'.5000'/10X,'2ND MOMENT',F10.4,8X,'.
```

```
     133'/10X,'3RD MOMENT',F10.4,8X,'.2500'/12X,'VARIANCE',F10.4,8X,
     2  '.0833')
      ADJSD=SQRT(.0833333/NT)
      Z= (ANS(1)-.5)/ADJSD
      WRITE(KOUT,9131) Z
 9131 FORMAT(///' Z - SCORE (XBAR-MU) = ',F8.2)
C
C      RUNS ABOVE / BELOW MEAN
C
  500 IF (IND(6) .EQ.0) GO TO 600
      C0=0.
      C5=.5
      MABMN=10
      IF (NT .LE. 3000) MABMN=9
      IF (NT .LE. 1500) MABMN=8
      IF (NT .LE. 1000) MABMN=7
      IF (NT .LE. 500)  MABMN=6
      CALL GAP2(N,KNTMN,MABMN,X,C0,C5,I,KTMSV)
      IF(I. NE. NITER) GO TO 600
      XTOT=0.
      DO 520 K=1,MABMN
      XTOT=XTOT+KNTMN(K)
  520 CELLG(K)=KNTMN(K)
      WRITE(KOUT,9150)
 9150 FORMAT('1RUNS ABOVE/BELOW MEAN (.5)')
      WRITE(KOUT,9151)
 9151 FORMAT(///5X,'RUN LENGTH',7X,'OBSERVED     THEORETICAL')
      PROB=.5
      TP=0.
      DO 540 K=1,MABMN
      K1=K-1
      IF(K .LT. MABMN) GO TO 535
      PROB=1.0-TP
  535 TP=TP+PROB
      EXPCT(K)=XTOT*PROB
      WRITE(KOUT,9063) K1,CELLG(K),EXPCT(K)
      IF(K .EQ. MABMN) WRITE(KOUT,9064)
      PROB=.5*PROB
  540 CONTINUE
      WRITE(KOUT,9065) XTOT
      CALL CHISQ(CELLG,EXPCT,MABMN,CS,IDF)
      CV05=CHSQD(.05,IDF)
      CV10=CHSQD(.10,IDF)
      WRITE(KOUT,9066) IDF,CS
      WRITE(KOUT,9067) CV05,CV10
      IF(IDF .EQ. 0) WRITE(KOUT,9068)
  600 IF(IND(7) .EQ.0) GO TO 700
      NLAG=50
      IF( NT/10 .LT. NLAG) NLAG=NT/10
      CALL AUTOC(X,N,NLAG,AC,I,NITER)
      IF(I.NE. NITER) GO TO 700
      WRITE(KOUT,9152)
 9152 FORMAT('1AUTOCORRELATIONS'//7X,'LAG',8X,'AC')
      KAC=0
      SD=SQRT(1./NT)*2.
      DO 620 K=1,NLAG
      WRITE(KOUT,9153) K,AC(K)
      IF(ABS(AC(K)) .GT. SD)  KAC=KAC+1
  620 CONTINUE
 9153 FORMAT(I10,F10.3)
```

```
      WRITE(KOUT,9154) SD,KAC
 9154 FORMAT(//' 95% LIMITS ON AUTOCORRELATIONS= (+/-)',F6.3/' NO. AC
     1SERVED OUTSIDE LIMITS= ',I4)
  700 IF(IND(8) .EQ.0) GO TO 1000
      IF(I .GE. 21) GO TO 790
      CALL KOLMO(X,N,ZP(I),ZM(I))
      XN=N
      ZP(I)= SQRT(XN)*ZP(I)
      ZM(I)=SQRT(XN)*ZM(I)
      IF(I .NE. NITER) GO TO 1000
  710 WRITE(KOUT,914)
  914 FORMAT('1KOLMOGOROV - SMIRNOV TEST'//'  ITER NO. ',8X,'ZP',8X,'Z
     1,'  PROB(ZP)  PROB(ZM)')
      DO 720 K=1,NITER
      CALL SMIRN(ZP(K),PZP)
      ZMA=ABS(ZM(K))
      CALL SMIRN(ZMA,PZM)
      WRITE(KOUT,9141) K,ZP(K),ZM(K),PZP,PZM
  720 CONTINUE
 9141 FORMAT(I10,4F10.4)
      GO TO 1000
  790 IF(I .EQ.NITER) GO TO 710
 1000 CONTINUE
      REWIND IFILE
      GO TO 1
89000 CALL EXIT
      END
```

```
C          DATA SET MSHOAGAP2  AT LEVEL 004 AS OF 05/24/74
      SUBROUTINE GAP2(N,KNT,MXGAP,X,A,B,IC,KSV)
      DIMENSION KNT(1),X(1)
C      THIS ROUTINE FINDS  GAPS OF THE LENGTHS 0,1,2, ...,MXGAP-2,
C          >=MXGAP-1 IN A SEQUENCE OF 'N' INPUTS NUMBERS.
C     A GAP IS THE LENGHT OF OBSERVASTIONS WHERE NO OBSERVATION
C    IN THE RANGE (A,B) IS RECORDED.
      IF(IC .GT.1) GO TO 20
      DO 10 I=1,MXGAP
   10 KNT(I)=0
      KSV=1
   20 KR=KSV
      DO 50 J=1,N
      IF(X(J) .LT. A) GO TO 30
      IF(X(J) .LT. B) GO TO 40
   30 KR=KR+1
      GO TO 50
   40 IF(KR .GT. MXGAP) KR=MXGAP
      KNT(KR)=KNT(KR)+1
      KR=1
   50 CONTINUE
      KSV=KR
      RETURN
      END
```

```
C            DATA SET MSHOACHISQ AT LEVEL 005 AS OF 05/18/74
      SUBROUTINE CHISQ(CELLS,EXP,K,CS,IDF)
C     THIS PROGRAM CALCULATES THE CHI-SQUARE STATISTIC OF K CELLS
C      WITHOBSERVED FREQUENCY COUNTS IN VECTOR CELLS AND THEORETICAL
C     VALUES IN 'EXP'.
      DIMENSION CELLS(1),EXP(1)
      KADJ5=0
      CS=0.
      DO 20 I=1,K
      IF (EXP(I) .GE. 5) GO TO 10
      IF(EXP(I) .LE.1) GO TO 50
      KADJ5=KADJ5+1
      IF(KADJ5 .EQ. 3) GO TO 50
   10 CS=CS+(CELLS(I)-EXP(I))**2/EXP(I)
   20 CONTINUE
      IDF=K-1
      RETURN
   50 CS=0.
      IDF=0
      RETURN
      END
```

```
C           DATA SET MSHOATALLY AT LEVEL 003 AS OF 03/18/74
      SUBROUTINE TALLY (X,N,CELLS,K,XMIN,XDELT,ICALL)
      DIMENSION X(1),CELLS(1)
C     TALLIES VECTOR X OF LENGTH N INTO CELLS <XMIN,XMIN TO XMIN+XDELT,
C      XMIN+XDELT TO XMIN+2(XDELT),....,XMIN+(K-2)XDELT TO XMIN+(K-1)XDE
C     AND >(K-1)XDELT
C     ICALL =1 ON FIRST CALL
C     ICALL #1 ON SUBSEQUENT CALLS
      IF(ICALL .NE. 1) GO TO 10
      DO 5 I=1,K
    5 CELLS(I)=0.
   10 DO 100 I=1,N
      ISUB=(X(I)-XMIN)/XDELT+2.
      IF(ISUB .GT.K) ISUB=K
      IF(ISUB .LE.0) ISUB=1
      CELLS(ISUB)=CELLS(ISUB)+1
  100 CONTINUE
      RETURN
      END
```

```
C          DATA SET MSHOAMOMNT AT LEVEL 004 AS OF 03/20/74
      SUBROUTINE MOMNT (X,N,ANS,ICALL)
      DIMENSION X(1),ANS(1)
C     THIS SUBROUTINE CALCULATES 1ST, 2ND, 3RD MOMENTS AND
C      THE VARIANCE OF A VECTOR OF LENGTH N.
C        ANS(1) = MEAN
C        ANS(2) = 2ND MOMENT
C        ANS(3) = 3RD MOMENT
C        ANS(4) = VARIANCE
C
C        ICALL = 1  FIRST PASS
C        ICALL = 2 SUCCEEDING PASSES EXCEPT LAST PASS
C        ICALL = 3  LAST PASS
C
C      SERO ANS ON 1ST PASS
      IF(ICALL .GT.1 .AND. ICALL .NE. 4) GO TO 10
      DO 5 I=1,4
    5 ANS(I)=0.
      NTOT=0
   10 DO 100 KK=1,N
      XI=X(KK)
      ANS(1)=ANS(1)+XI
      ANS(2)=ANS(2)+XI*XI
      ANS(3)=ANS(3)+XI**3
  100 CONTINUE
      NTOT=NTOT+N
      IF (ICALL .LT. 3) RETURN
C      CALCULATE RESULTS
      ANS(1)=ANS(1)/NTOT
      ANS(2)=ANS(2)/NTOT
      ANS(3)=ANS(3)/NTOT
      ANS(4)=ANS(2)-ANS(1)**2
      RETURN
      END
```

```
C          DATA SET MSHOAKOLMO AT LEVEL 003 AS OF 05/23/74
           SUBROUTINE KOLMO (X,N,ZPLUS,ZMIN)
C           THIS SUBROUTINE TESTS THE  DIFFERENCE BETWEEN AN
C           EMPIRICAL AND THEORECTICAL DISTRIBUTION USING THE
C           KOLMOGOROV-SMIRNOV GOODNESS OF FIT TEST.
C
C           REFERENCE  IBM - SSP  COPYRIGHT 1968 PP. 63-64
           DIMENSION X(1)
C
C           SORT X INTO ASCENDING ORDER
C
           M=N
        20 M=M/2
           IF(M .EQ. 0) GO TO 40
           K=N-M
           J=1
        41 I=J
        49 L=I+M
           IF(X(I)-X(L)) 60,60,50
        50 XS=X(I)
           X(I)=X(L)
           X(L)=XS
           I=I-M
           IF(I-1) 60,49,49
        60 J=J+1
           IF(J-K) 41,41,20
        40 CONTINUE
C
C           FIND MIN AND MAX DEVIATION
C
           NM1=N-1
           XN=N
           ZPLUS=-1000.
           ZMIN=+1000.
           IL=1
         6 DO 7 I=IL,NM1
           J=I
           IF( X(J) .NE. X(J+1)) GO TO 9
         7 CONTINUE
         8 J=N
         9 IL=J+1
           FS=FLOAT(J)/XN
           IF (X(J) .GT. 0) GO TO 23
           Y=0.
           GO TO 27
        23 IF(X(J) .LT. 1.)  GO TO 25
           Y=1.
           GO TO 27
        25 Y=X(J)
        27 DIFF=Y-FS
           IF(DIFF .GT. ZPLUS) ZPLUS=DIFF
           IF(DIFF .LT. ZMIN) ZMIN=DIFF
           IF(IL-N) 6,8,28
        28 RETURN
           END
```

```
C             DATA SET MSHOARUNTL AT LEVEL 005 AS OF 05/23/74
           SUBROUTINE RUNTL (X,N,RUNS,ICALL,NVAL)
           DIMENSION X(1),RUNS(1)
C           THIS SUBROUTINE TALLIES THE NUMBER OF RUNS OF
C          LENGTH I, I=1,8 .   RUNS OF LENGTH >= TO 8 ARE TALLIED IN RUNS(8)
C            THE FIRST AND LAST RUN TOF THE TOTAL SEQUENCE ARE NOT TALLIED.
C
           IF(ICALL .NE.1) GO TO 50
C          ZERO OUT RUNS
           DO 5 I=1,8
         5 RUNS(I)=0.
C          IGNORE 1ST RUN
C         IS FIRST RUN UP OR DOWN
           IUPS=+1
           IF(X(2)-X(1) .LT.0) IUPS=-1
           DO 10 I=3,N
           IUP=+1
           IF(X(I)-X(I-1) .LT. 0) IUP=-1
           IF(IUPS .EQ. IUP) GO TO 10
           IUPS=IUP
           NSTART=I+2
           GO TO  20
        10 CONTINUE
           STOP 99
        20 KNT=1
           XSAVE=X(NSTART-1)
           GO TO 100
        50 IUP=+1
           NSTART=2
           IF(X(NSTART-1)-XSAVE .LT.0) IUP=-1
           IF(IUPS .EQ. IUP) GO TO 80
           IUPS=IUP
           IF (KNT .GT. NVAL) KNT=NVAL
           RUNS(KNT)=RUNS(KNT)+1
           KNT=1
           GO TO 100
        80 KNT=KNT+1
       100 DO 200 I=NSTART,N
           IUP=+1
           IF(X(I)-X(I-1) .LT.0) IUP=-1
           IF(IUPS .EQ.IUP) GO TO 180
           IUPS=IUP
           IF (KNT .GT. NVAL) KNT=NVAL
           RUNS(KNT)=RUNS(KNT)+1
           KNT=1
           GO TO 200
       180 KNT=KNT+1
       200 CONTINUE
           XSAVE=X(N)
           RETURN
           END
```

```
C           DATA SET MSHOARUNTS AT LEVEL 007 AS OF 05/24/74
      SUBROUTINE RUNTS (RUNS,N,E,CS ,IDF,NVAL)
C      THIS SUBROUTINE COMPUTES THE CHI-SQUARE STATISTIC FOR THE RUNS
C     TALLIED IN SUBROUTINE  RUNTL.
C
C     REFERENCE IBM SYSTEMS JOURNAL 1969 PP. 136-46
      DIMENSION RUNS(1),E(1)
C     EXPECTED NUMBER OF RUNS
      DENOM=6.
      NV1=NVAL-1
      DO 100 I=1,NV1
      SUM1=N*(I*I+3*I+1)
      SUM2=I**3+3*I*I-I-4
      DENOM=DENOM*(I+3.)
  100 E(I)=2.*(SUM1-SUM2)/DENOM
      E(NVAL)=2.*(N*(NVAL+1.)-(NVAL**2+NVAL-1.))
      E(NVAL)=E(NVAL)/DENOM
      CALL CHISQ(RUNS,E,NVAL,CS,IDF)
      RETURN
      END
```

```
C           DATA SET MSHOAPARTL AT LEVEL 001 AS OF 03/11/74
            SUBROUTINE PARTL (X,N,K,A,IA,ICALL)
            DIMENSION X(1),A(IA,IA)
C           THIS PROGRAM TALLIES THE OCCURENCE OF THE PAIRED COORDINATES
C         X(I),X(I+1) OF A SEQUENCE OF PSEUDO-RANDOM
C            NUMBERS ON (0,1)  INTO A K X K ARRAY
C             IF ICALL = = 1 ROUTINE INITIALIZES A MATRIX.
C
            IF(ICALL .NE.1) GO TO 100
             DO 10 I=1,IA
            DO 10 J=1,IA
         10 A(I,J)=0.
        100 DO 150 I=1,N,2
            J=K*X(I)+1
            M=K*X(I+1)+1
        150 A(J,M) =A(J,M)+1.
            RETURN
            END
```

```
C           DATA SET MSHOAPARTS AT LEVEL 002 AS OF 03/23/74
      SUBROUTINE PARTS(A,IA,K,CS,STDCS)
      DIMENSION A(IA,IA)
C      THIS PROGRAM PERFORMS THE PAIRS TEST ON A  PREVIOUSLY
C      TALLIED SEQUENCE OF PSEUDO-RANDOM NUMBER(USING PARTL).
      TOT=0.
      DO 50 I=1,K
      DO 50 J=1,K
   50 TOT=TOT+A(I,J)
      E=TOT/K**2
      CS=0.
      DO 100 I=1,K
      DO 100 J=1,K
  100 CS=CS + (A(I,J)-E)**2/E
      F=K*K-1
      STDCS=(CS-F)/SQRT(2.*F)
      RETURN
      END
```

```
      C          DATA SET MSHOACHSQD AT LEVEL 002 AS OF 06/09/74
            FUNCTION CHSQD (P,N)
      C      THIS FUNCTION IS USED TO EVALUATE THE QUANTILE
      C     AT A GIVEN PROBABILITY LEVEL, P, FOR THE CHI-SQUARE
      C      DISTRIBUTION WITH N DEGREES OF FREEDOM.
      C
      C      REFERENCE  COMM. OF THE ACM  VOL 16 NO 8  PP. 483-5
      C
            DIMENSION C(21),A(19)
            DATA C/1.565326E-3,1.060438E-3,-6.950356E-3,
           *  -1.323293E-2,2.277679E-2,-8.986007E-3,
           *  -1.513904E-2,2.530010E-3,-1.450117E-3,
           *  5.169654E-3,-1.153761E-2,1.128186E-2,
           *  2.607083E-2,-0.2237368,9.780499E-5,
           *  -8.426812E-4,3.125580E-3,-8.553069E-3,
           *  1.348028E-4,0.4713941,1.0000886/
            DATA A/1.264616E-2,-1.425296E-2,1.400483E-2,
           *  -5.886090E-3,-1.091214E-2,-2.304527E-2,
           *  3.135411E-3,-2.728484E-4,-9.699681E-3,
           *  1.316872E-2,2.618914E-2,-0.2222222,5.406674E-5,
           *3.483789E-5,-7.274761E-4,3.292181E-3,
           *  -8.729713E-3,0.4714045,1./
            IF(N-2) 10,20,30
         10 CHSQD=GAUSD(.5*P)
            CHSQD=CHSQD*CHSQD
            RETURN
         20 CHSQD= -2.*ALOG(P)
            RETURN
         30 F=N
            F1=1./F
            T=GAUSD(1.-P)
            F2=SQRT(F1)*T
            IF(N .GE. (2+INT(4.*ABS(T))))  GO TO 40
            CHSQD=(((((((C(1)*F2+C(2))*F2+C(3))*F2+C(4))*F2
           1  +C(5))*F2+C(6))*F2+C(7))*F1 + ((((((C(8)+C(9)*F2)*F2
           2  +C(10))*F2+C(11))*F2+C(12))*F2+C(13))*F2+C(14)))*F1 +
           3  (((((C(15)*F2+C(16))*F2+C(17))*F2+C(18))*F2
           4  +C(19))*F2+C(20))*F2+C(21)
            GO TO 50
         40 CHSQD=(((A(1)+A(2)*F2)*F1+(((A(3)+A(4)*F2)*F2
           1  +A(5))*F2+A(6)))*F1+(((((A(7)+A(8)*F2)*F2+A(9))*F2
           2  +A(10))*F2+A(11))*F2+A(12)))*F1+(((((A(13)*F2
           3  +A(14))*F2+A(15))*F2+A(16))*F2+A(17))*F2*F2
           4  +A(18))*F2+A(19)
         50 CHSQD=CHSQD*CHSQD*CHSQD*F
            RETURN
            END
```

```
C           DATA SET MSHOAGAUSD AT LEVEL 003 AS OF 06/09/74
      FUNCTION GAUSD(P)
C      THIS FUNCTION CALCULATES THE NORMAL DEVIATE FOR THE VALUE
C       P  OF THE CUULMULATIVE PROBABILITY DISTRIBUTION.
C
C     ALGORITHM FROM HASTINGS, CECIL,JR.  APPROX FOR DIGITAL COMPUTERS
C        1957,  P. 192.
      DATA A0,A1,A2/2.515517,.802853,.010328/
      DATA B1,B2,B3/1.432788,.189269,.001308/
      B=P
      IF(B .GT. .5) B=1.-B
      U1=-ALOG(B)
      U=SQRT(2.*U1)
      U2=U*U
      U3=U2*U
      GAUSD=U-(A0+A1*U+A2*U2)/(1.+B1*U+B2*U2+B3*U3)
      IF(P .LT. .5) GAUSD=-GAUSD
      RETURN
      END
```

```
      C          DATA SET MSHOAAUTOC AT LEVEL 004 AS OF 05/18/74
            SUBROUTINE AUTCC(X,N,LAG,AC,IC,NITER)
            DIMENSION X(1)
            DIMENSION AC(1),XSAVE(50)
            XBAR=.5
            IF(IC .GT. 1) GO TO 50
            NT=N
            CO=0.
            DO 10 I=1,LAG
         10 AC(I)=0.
            GO TO 200
         50 NT=NT+N
            DO 100 K=1,LAG
            NK=LAG-K+1
            DO 80 I=1,K
            AC(K)=AC(K)+(XSAVE(NK)-XBAR)*(X(I)-XBAR)
         80 NK=NK+1
        100 CONTINUE
        200 DO 300 K=1,LAG
            NK=N-K
            DO 300 I=1,NK
            NKI=I+K
            AC(K)=AC(K)+(X(I)-XBAR)*(X(NKI)-XBAR)
            IF(K.EQ.1) CO=CO+X(I)*X(I)
        300 CONTINUE
            CO=CO+X(N)*X(N)
            DO 400 I=1,LAG
            NLI=N-LAG+I
        400 XSAVE(I)=X(NLI)
            IF(IC .NE. NITER) RETURN
            CO=(CO-NT*XBAR*XBAR)/NT
            DO 450 K=1,LAG
        450 AC(K)=AC(K)/(NT*CO)
            RETURN
            END
```

```
C           DATA SET MSHOASMIRN AT LEVEL 003 AS OF 05/11/74
            SUBROUTINE SMIRN(X,Y)
C
C            THIS SUBROUTINE COMPUTES THE LIMITING DISTRIBUTION
C             FUNCTION OF THE KOLMOGOROV-SMIRNOV STATISTIC.
C            REF.  I B M  SSP   PP. 66-67
C
            IF(X-.27) 1,1,2
          1 Y=0.
            RETURN
          2 IF(X-1.) 3,6,6
          3 Q1=EXP(-1.233701/X**2)
            Q2=Q1*Q1
            Q4=Q2*Q2
            Q8=Q4*Q4
            IF(Q8   .LT. 1.E-25)  Q8=0.
            Y=(2.506628/X)*Q1*(1.+Q8*(1.+Q8*Q8))
            RETURN
          6 IF(X -3.1) 8,7,7
          7 Y=1.
            RETURN
          8 Q1=EXP(-2*X*X)
            Q2=Q1*Q1
            Q4=Q2*Q2
            Q8=Q4*Q4
            Y=1.-2.*(Q1-Q4+Q8*(Q1-Q8))
            RETURN
            END
```

```
TRAN IV G LEVEL  21                 MAIN                DATE = 74158          22/46

          C        DATA SET MSHOAMWALD AT LEVEL 001 AS OF 05/25/74
001             FUNCTION MWALD(N)
          C
          C     THIS FUNCTION COMPUTES THE OPTIMAL NUMBER OF CLASS
          C      FOR A CHI-AQUARE TEST ACCORDING TO THE MANN-WALD CRITERIA
          C
002             B=4.
003             CCRIT=1.645
004             XN=N-1
005             MWALD=B*(2.*XN*XN/CCRIT**2)**.2
006             RETURN
007             END
```

```
RAN IV G LEVEL  21                      RAN                  DATE = 74158              22/46/1

          FUNCTION RAN(NX)
          RAN=0.0
          RETURN
          END
```

APPENDIX B

Sample Output Listing

RAN

NT = 5000

```
  NT =      5000
NSEED =     95605
```

TESTS REQUESTED
- GAP TESTS - 10
- RUNS TEST
- PAIRS TEST
- CHI - SQUARE TEST
- MOMENTS
- RUNS ABOVE/BELOW MEAN
- AUTOCORRELATIONS
- KOLMOGOROV - SMIRNOV TEST

GAP TEST

GAP INTERVAL =(0.0 , 0.1000)

GAP LENGTH	OBSERVED	THEORETICAL
0	57.	51.900
1	54.	46.710
2	33.	42.039
3	39.	37.835
4	30.	34.052
5	28.	30.646
6	31.	27.582
7	27.	24.824
8	19.	22.341
9+	201.	201.072
TOTAL	519.	

CHI - SQUARE(9 DF) = 5.44

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP TEST

GAP INTERVAL =(0.1000, 0.2000)

GAP LENGTH	OBSERVED	THEORETICAL
0	54.	48.700
1	49.	43.830
2	39.	39.447
3	31.	35.502
4	28.	31.952
5	29.	28.757
6	23.	25.881
7	18.	23.293
8	25.	20.964
9+	191.	188.674
TOTAL	487.	

CHI - SQUARE(9 DF) = 4.58

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP TEST

GAP INTERVAL =(0.2000, 0.3000)

GAP LENGTH	OBSERVED	THEORETICAL
0	47.	50.200
1	41.	45.180
2	43.	40.662
3	34.	36.596
4	29.	32.936
5	30.	29.643
6	28.	26.678
7	32.	24.010
8	15.	21.609
9+	203.	194.486
TOTAL	502.	

CHI - SQUARE(9 DF) = 6.50

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP TEST

GAP INTERVAL =(0.2000, 0.4000)

GAP LENGTH	OBSERVED	THEORETICAL
0	59.	52.800
1	44.	47.520
2	47.	42.768
3	38.	38.491
4	43.	34.642
5	29.	31.178
6	24.	28.060
7	31.	25.254
8	21.	22.729
9+	192.	204.558
TOTAL	528.	

CHI - SQUARE(9 DF) = 6.38

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP TEST

GAP INTERVAL =(0.4000, 0.5000)

GAP LENGTH	OBSERVED	THEORETICAL
0	44.	47.100
1	44.	42.390
2	36.	38.151
3	32.	34.336
4	33.	30.902
5	22.	27.812
6	31.	25.031
7	26.	22.528
8	20.	20.275
9+	183.	182.475
TOTAL	471.	

CHI - SQUARE(9 DF) = 3.87

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP INTERVAL =(0.5000, 0.6000)

GAP LENGTH	OBSERVED	THEORETICAL
0	44.	50.100
1	45.	45.090
2	39.	40.581
3	47.	36.523
4	30.	32.871
5	25.	29.583
6	26.	26.625
7	22.	23.963
8	25.	21.566
9+	198.	194.098
TOTAL	501.	

CHI - SQUARE(9 DF) = 5.57

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP INTERVAL =(0.6000, 0.7000)

GAP LENGTH	OBSERVED	THEORETICAL
0	51.	52.300
1	57.	47.070
2	35.	42.363
3	43.	38.127
4	36.	34.314
5	33.	30.883
6	30.	27.794
7	23.	25.015
8	16.	22.513
9+	199.	202.621
TOTAL	523.	

CHI - SQUARE(9 DF) = 6.54

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP TEST

GAP INTERVAL =(0.7000, 0.8000)

GAP LENGTH	OBSERVED	THEORETICAL
0	51.	52.300
1	58.	47.070
2	42.	42.363
3	36.	38.127
4	24.	34.314
5	27.	30.883
6	33.	27.794
7	32.	25.015
8	29.	22.513
9+	191.	202.621
TOTAL	523.	

CHI - SQUARE(9 DF) = 11.74

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP INTERVAL =(0.8000, 0.9000)

GAP LENGTH	OBSERVED	THEORETICAL
0	38.	45.400
1	37.	40.860
2	37.	36.774
3	34.	33.097
4	27.	29.787
5	20.	26.808
6	20.	24.127
7	26.	21.715
8	23.	19.543
9+	192.	175.889
TOTAL	454.	

CHI - SQUARE(9 DF) = 7.23

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

GAP TEST

GAP INTERVAL =(0.9000, 1.0000)

GAP LENGTH	OBSERVED	THEORETICAL
0	60.	49.200
1	47.	44.280
2	42.	39.852
3	32.	35.867
4	31.	32.280
5	23.	29.052
6	21.	26.147
7	17.	23.532
8	15.	21.179
9+	204.	190.611
TOTAL	492.	

CHI - SQUARE(9 DF) = 9.95

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

RUNS TEST

RUN LENGTH	OBSERVED	THEORETICAL
1	2097.	2083.417
2	888.	916.433
3	277.	263.758
4	60.	57.498
5	10.	10.159
6+	0.	1.734
TOTAL	3332.	3332.999

CHI - SQUARE(5 DF) = 3.48

CRITICAL VALUE (ALPHA=.05) = 11.07
CRITICAL VALUE (ALPHA=.10) = 9.24

Z - SCORE (TOTAL RUNS) = -0.03

PAIRS TE

NO. OF INTERVALS = 10

INTERVAL	FROM	TO	1	2	3	4	5	6	7	8	9	10
1	0.0	0.1000	28.	21.	27.	27.	24.	25.	22.	16.	20.	25.
2	0.1000	0.2000	18.	25.	20.	24.	24.	12.	26.	30.	14.	28.
3	0.2000	0.3000	34.	28.	21.	32.	25.	28.	21.	31.	20.	27.
4	0.3000	0.4000	30.	33.	23.	29.	22.	28.	29.	25.	25.	24.
5	0.4000	0.5000	10.	27.	19.	23.	21.	22.	26.	32.	26.	25.
6	0.5000	0.6000	31.	33.	30.	27.	27.	24.	18.	20.	22.	24.
7	0.6000	0.7000	34.	24.	26.	26.	25.	29.	23.	32.	32.	32.
8	0.7000	0.8000	35.	24.	21.	30.	35.	31.	24.	23.	19.	34.
9	0.8000	0.9000	36.	25.	21.	25.	21.	21.	22.	23.	21.	23.
10	0.9000	1.0000	28.	26.	27.	17.	16.	25.	27.	15.	16.	26.

CHI - SQUARE(99 DF) = 109.20

CRITICAL VALUE (ALPHA=.05) = 123.23
CRITICAL VALUE (ALPHA=.10) = 117.41

CHI - SQUARE TEST

INTERVAL	FROM -	TO	OBSERVED	THEORETICAL
1	0.0	0.0088	45.	44.25
2	0.0088	0.0177	57.	44.25
3	0.0177	0.0265	46.	44.25
4	0.0265	0.0354	42.	44.25
5	0.0354	0.0442	37.	44.25
6	0.0442	0.0531	42.	44.25
7	0.0531	0.0619	38.	44.25
8	0.0619	0.0708	53.	44.25
9	0.0708	0.0796	53.	44.25
10	0.0796	0.0885	56.	44.25
11	0.0885	0.0973	36.	44.25
12	0.0973	0.1062	42.	44.25
13	0.1062	0.1150	44.	44.25
14	0.1150	0.1239	44.	44.25
15	0.1239	0.1327	39.	44.25
16	0.1327	0.1416	40.	44.25
17	0.1416	0.1504	39.	44.25
18	0.1504	0.1593	59.	44.25
19	0.1593	0.1681	37.	44.25
20	0.1681	0.1770	41.	44.25
21	0.1770	0.1858	45.	44.25
22	0.1858	0.1947	48.	44.25
23	0.1947	0.2035	46.	44.25
24	0.2035	0.2124	34.	44.25
25	0.2124	0.2212	36.	44.25
26	0.2212	0.2301	45.	44.25
27	0.2301	0.2389	47.	44.25
28	0.2389	0.2478	39.	44.25
29	0.2478	0.2566	56.	44.25
30	0.2566	0.2655	39.	44.25
31	0.2655	0.2743	54.	44.25
32	0.2743	0.2832	44.	44.25
33	0.2832	0.2920	49.	44.25
34	0.2920	0.3009	36.	44.25
35	0.3009	0.3097	54.	44.25
36	0.3097	0.3186	44.	44.25
37	0.3186	0.3274	45.	44.25
38	0.3274	0.3363	55.	44.25
39	0.3363	0.3451	48.	44.25
40	0.3451	0.3540	52.	44.25
41	0.3540	0.3628	37.	44.25
42	0.3628	0.3717	36.	44.25
43	0.3717	0.3805	58.	44.25
44	0.3805	0.3894	52.	44.25
45	0.3894	0.3982	48.	44.25
46	0.3982	0.4071	45.	44.25
47	0.4071	0.4159	41.	44.25
48	0.4159	0.4248	31.	44.25
49	0.4248	0.4336	36.	44.25
50	0.4336	0.4425	41.	44.25
51	0.4425	0.4513	46.	44.25
52	0.4513	0.4602	44.	44.25
53	0.4602	0.4690	36.	44.25
54	0.4690	0.4779	47.	44.25
55	0.4779	0.4867	53.	44.25
56	0.4867	0.4956	46.	44.25
57	0.4956	0.5044	48.	44.25
58	0.5044	0.5133	42.	44.25
59	0.5133	0.5221	39.	44.25

61	0.5310	[illegible]	[illegible]	[illegible]
62	0.5398	0.5487	44.	44.25
63	0.5487	0.5575	56.	44.25
64	0.5575	0.5664	43.	44.25
65	0.5664	0.5752	38.	44.25
66	0.5752	0.5841	47.	44.25
67	0.5841	0.5929	45.	44.25
68	0.5929	0.6018	36.	44.25
69	0.6018	0.6106	53.	44.25
70	0.6106	0.6195	54.	44.25
71	0.6195	0.6283	36.	44.25
72	0.6283	0.6372	56.	44.25
73	0.6372	0.6460	39.	44.25
74	0.6460	0.6549	41.	44.25
75	0.6549	0.6637	39.	44.25
76	0.6637	0.6726	39.	44.25
77	0.6726	0.6814	52.	44.25
78	0.6814	0.6903	48.	44.25
79	0.6903	0.6991	55.	44.25
80	0.6991	0.7080	39.	44.25
81	0.7080	0.7168	44.	44.25
82	0.7168	0.7257	35.	44.25
83	0.7257	0.7345	51.	44.25
84	0.7345	0.7434	52.	44.25
85	0.7434	0.7522	60.	44.25
86	0.7522	0.7611	53.	44.25
87	0.7611	0.7699	53.	44.25
88	0.7699	0.7788	32.	44.25
89	0.7788	0.7876	37.	44.25
90	0.7876	0.7965	48.	44.25
91	0.7965	0.8053	44.	44.25
92	0.8053	0.8142	38.	44.25
93	0.8142	0.8230	37.	44.25
94	0.8230	0.8319	37.	44.25
95	0.8319	0.8407	32.	44.25
96	0.8407	0.8496	40.	44.25
97	0.8496	0.8584	42.	44.25
98	0.8584	0.8673	40.	44.25
99	0.8673	0.8761	51.	44.25
100	0.8761	0.8850	44.	44.25
101	0.8850	0.8938	42.	44.25
102	0.8938	0.9026	40.	44.25
103	0.9026	0.9115	43.	44.25
104	0.9115	0.9203	49.	44.25
105	0.9203	0.9292	41.	44.25
106	0.9292	0.9380	44.	44.25
107	0.9380	0.9469	44.	44.25
108	0.9469	0.9557	40.	44.25
109	0.9557	0.9646	50.	44.25
110	0.9646	0.9734	41.	44.25
111	0.9734	0.9823	49.	44.25
112	0.9823	0.9911	43.	44.25
113	0.9911	1.0000	36.	44.25
TOTAL			5000.	

CHI - SQUARE(112 DF) = 115.01

CRITICAL VALUE (ALPHA=.05) = 137.71
CRITICAL VALUE (ALPHA=.10) = 131.56

MOMENTS

	OBSERVED	THEORETICAL
MEAN	0.4969	.5000
2ND MOMENT	0.2296	.3333
3RD MOMENT	0.2461	.2500
VARIANCE	0.0827	.0833

Z - SCORE (XBAR-MU) = -0.77

RUN LENGTH	OBSERVED	THEORETICAL
0	1249.	1253.500
1	639.	626.750
2	321.	313.375
3	144.	156.688
4	69.	78.344
5	43.	39.172
6	24.	19.586
7	7.	9.793
8	7.	4.896
9+	4.	4.896
TOTAL	2507.	

CHI - SQUARE(9 DF) = 5.82

CRITICAL VALUE (ALPHA=.05) = 16.92
CRITICAL VALUE (ALPHA=.10) = 14.68

LAG	AC
1	0.008
2	-0.004
3	-0.002
4	0.015
5	-0.002
6	-0.011
7	0.004
8	0.016
9	0.012
10	-0.028
11	0.008
12	-0.030
13	-0.011
14	0.004
15	0.006
16	0.015
17	-0.003
18	-0.012
19	0.006
20	0.008
21	-0.015
22	0.013
23	0.000
24	0.005
25	-0.022
26	-0.035
27	0.009
28	-0.006
29	0.006
30	0.013
31	0.031
32	-0.004
33	-0.023
34	-0.016
35	-0.023
36	-0.010
37	0.008
38	-0.014
39	-0.002
40	-0.022
41	0.020
42	0.013
43	-0.016
44	-0.006
45	0.012
46	0.010
47	-0.016
48	0.014
49	-0.005
50	0.004

95% LIMITS ON AUTOCORRELATIONS= (+/-) 0.028
NO. AC OBSERVED OUTSIDE LIMITS= 4

KOLMOGOROV - SMIRNOV TEST

ITER NO.	ZP	ZM	PROB(ZP)	PROB(ZM)
1	0.1632	-1.0038	0.0	0.7341