# Testing Pseudo-Random Number Generators

Julián Ortiz C. and Clayton V. Deutsch
(jmo1@ualberta.ca - cdeutsch@civil.ualberta.ca)
Department of Civil & Environmental Engineering, University of Alberta

## Abstract

*Random numbers are at the heart of all geostatistical simulation methods. Practitioners assume that the software they are using uses an appropriate pseudo-random number generator; however, this may not be true. The history of pseudo-random number generation is reviewed. Tests for randomness are described and applied to five different pseudo random number generators. Results show that generators that were considered good a few years ago fail some recent tests. We recommend careful testing and monitoring of the literature.*

## 1 Literature Review

In the early twentieth century people needed random numbers for their scientific work. They started replacing the basic methods of drawing balls of a well stirred urn or rolling dice with tables of numbers taken from some source or with random numbers generated by mechanical devices. In 1927 a table of over 40,000 digits taken at random from census reports was published by L. H. C. Tippett [26]. M. G. Kendall and B. Babington-Smith [11] presented in 1938 a mechanical device to generate random digits. They proposed 4 different tests that they applied to a sequence of 5,000 digits generated by their machine. The same tests were applied to two series of 1,000 digits obtained from Tippett's table. All the sequences passed the tests and were considered locally random. In 1939 a table with 100,000 digits was published by Kendall and Babington-Smith [12] using the same randomizing machine. They tested their digits and those published by Fisher and Yates [5] with satisfactory results. The well-known RAND [21] table of random digits was published in 1955. It included 1 million digits generated by another machine from electronic noise. Most of those tables showed undesirable properties when new tests were applied.

The introduction of computers led to other ways to generate random number sequences. Tables had limited utility because of their size. Instead, arithmetic operations were proposed to efficiently generate sequences of random numbers on computers. J. Von Neumann [20] presented in 1946 the middle square method, which consists in taking the middle digits of the previous number squared. This method does not generate random sequences [14].

One of the most popular methods to generate sequences of random numbers was the Linear Congruential Method, which is covered in more detail in the following section. These generators present some undesirable properties such as lattice structure [18]. In 1965, M.D. MacLaren and G. Marsaglia presented a procedure to combine two generators to have better sequences (more random). In 1989, R.S. Wikramaratna proposed the Additive Congruential Method which has proved to give satisfactory results. More details about the history of pseudo- random number generators can be found in Knuth [14], Ripley [22], and Kennedy and Gentle [13].

## 1.1 Linear Congruential Method

D. H. Lehmer [15] introduced in 1948 the idea of generating a random number sequence using the following formula:

$$X_{n+1} = (aX_n + c)_{modM}, \quad n > 1$$

where $X_0$ is the starting value or seed of the sequence ($X_0 \geq 0$), $a$ is called the multiplier ($a \geq 0$), $c$ is the increment ($c \geq 0$), and $M$ is the modulus ($M \geq X_0$, $M \geq a$, $M \geq c$).

When $c$ is set to zero (as it was in the original sequence proposed by Lehmer) the method is called *Multiplicative Congruential Method*, otherwise (i.e. if $c \neq 0$), it is called *Mixed Congruential Method*. The first examples of the mixed generator were given independently by Thomson [25] and Rotenberg [23]. Other applications were presented by Franklin [6] and Greenberger [9].

## 1.2 Additive Congruential Method

Additive generators calculate each number as some additive combination of the previous $n$ numbers in the sequence. R. S. Wikramaratna [28, 29, 30] proposed the $k^{th}$ order ACORN (additive congruential random number) generator $X_j^k$, a more general recursive method than the linear congruential, which combines the previous number in the sequence with a corresponding number from the $(k-1)^{th}$ order sequence. $X_j^k$ is defined recursively from a seed $X_0^0$ ($0 < X_0^0 < 1$) and a set of $k$ initial values $X_0^m$, $m$=1, ..., $k$ each satisfying $0 \leq X_0^m \leq 1$ by:

$$\begin{array}{ll} X_n^0 = X_{n-1}^0, & n \geq 1 \\ X_n^m = (X_n^{m-1} + X_{n-1}^m)_{mod1}, & n \geq 1, \quad m = 1, ..., k \end{array}$$

This generator has three features: it is faster to compute (the algorithm is very simple), the period length can be set arbitrarily large, and it gives the same sequence in any machine (differing only in the number of significant digits).

**Figure 1** presents a schematic of this method. The user has to choose the numbers in the first column. All the numbers in the Zero Order row are the same (the seed number $X_0^0$). The arrows show which previous numbers are used to calculate the current one. The numbers generated in the row of the $k^{th}$ order are considered to be pseudo-random numbers. One should not take the first few numbers, since for seed numbers close to each other they may be similar. It is recommended to initialize the sequence not considering the first thousands.

## 1.3 Other Methods

Many other methods to generate sequences of random numbers may be cited. R. R. Coveyou created a quadratic method (which is, in fact, a double precision middle square method). The seed has to be chosen such that:

$$X_{0\,mod4} = 2$$

$$
\begin{array}{lllccccccc}
Zero\ \ Order: & X_0^0 & \rightarrow & X_1^0 & \rightarrow & X_2^0 & \rightarrow & \cdots & \rightarrow & X_N^0 \\
 & & & \downarrow & & \downarrow & & & & \downarrow \\
1^{st}\ \ Order: & X_0^1 & \rightarrow & X_1^1 & \rightarrow & X_2^1 & \rightarrow & \cdots & \rightarrow & X_N^1 \\
 & & & \downarrow & & \downarrow & & & & \downarrow \\
2^{nd}\ \ Order: & X_0^2 & \rightarrow & X_1^2 & \rightarrow & X_2^2 & \rightarrow & \cdots & \rightarrow & X_N^2 \\
 & & & \downarrow & & \downarrow & & & & \downarrow \\
 & & & \vdots & & \vdots & & & & \vdots \\
 & & & \downarrow & & \downarrow & & & & \downarrow \\
k^{th}\ \ Order: & X_0^k & \rightarrow & X_1^k & \rightarrow & X_2^k & \rightarrow & \cdots & \rightarrow & X_N^k
\end{array}
$$

Figure 1: Schematic showing how `acorn` generates random numbers.

and the sequence is then defined by:

$$X_{n+1} = X_n \cdot (X_n + 1)_{mod 2^e}, \quad n > 0$$

The well-known Fibonacci sequence (originated in the early 1950's) present a long period (longer than $M$), but it is not satisfactorily random. It is defined by:

$$X_{n+1} = (X_n + X_{n-1})_{mod M}$$

Variations such as the one presented by Green, Smith and Klem [8], do not improve the randomness of the sequences considerably:

$$X_{n+1} = (X_n + X_{n-k})_{mod M}$$

An interesting approach was presented by MacLaren and Marsaglia [17] and consists in combining two sequences to get another "even more random". This approach has been accepted by some authors and rejected by others [3, 22]. In any case, the algorithm proposed by MacLaren and Marsaglia seems to work well, as shown by F. Gebhardt [7].

A different method to combine two sequences was proposed by W. J. Westlake [27], based on circular shifting and exclusive "or" on a binary computer.

## 2 Statistical Tests

The sequences generated by any algorithm must be tested in order to know quantitatively its randomness. The statistical tests applied to pseudo- random number sequences can be grouped as:

**Empirical tests** : Non-parametric test of a sample sequence of numbers. The evaluation is based on "goodness of fit" of observed distributions with respect to expected ones (predicted theoretically).

**Theoretical tests** : Based on theoretical properties of the generators that may be deduced without a sample sequence. Generators such as the congruential ones are predictable in the sense that knowing the values $a$, $c$, $M$, and $X_0$, the period of the generator may be predicted, as well as other properties.

## 2.1 Empirical Tests

Many different tests have been used to test sequences of pseudo-random numbers. Most of them are based in a comparison between observed and expected frequencies. A $\chi^2$ test or a Kolmogorov-Smirnov test can be applied to quantify the mismatch between both distributions, based on probability at some level of significance [1, 2]. A brief description of those tests is given below:

1. $\chi^2$ **Test**: The following statistic is used:

$$Q = \sum_{i=1}^{n} \frac{(x_i - m_i)^2}{m_i} \sim \chi^2_{n-1}$$

   where $x_i$ is the experimental frequency in interval $i$ and $m_i$ is the expected (theoretical) frequency in the same interval. The quadratic form $Q$ follows a $\chi^2$ distribution with $n-1$ degrees of freedom [1]. Once the value of $Q$ has been calculated, the user can refer to tables to find the percentile for a $\chi^2$ distribution with $n-1$ degrees of freedom. One should expect to be between the fifth and ninety fifth percentiles.

2. **Kolmogorov-Smirnov Test**: The statistic $D$ may be compared with the critical value for a given significancy level:

$$D = Max|F_i - S_i|$$

   where $F_i$ denotes the cumulative relative frequency for each category of the theoretical distribution and $S_i$ is the value from the observed data.

   Tables of critical values of $D$ for different probability values may be found in most statistics books [10, 14].

The following are considered the most powerful tests for randomness [3, 11, 14, 24].

- **Frequency test (uniformity or equidistribution test)**: in a sequence of random digits the observed frequency can be compared with the expected frequency (each digit should appear $\frac{n}{10}$ times, where $n$ is the total number of digits in the series). A $\chi^2$ test can be applied to quantify the departure between observed and expected results. In a sequence of numbers, the interval can be divided into $n$ subsets (e.g. $U \leq 0.01, 0.01 < U \leq 0.02, ...$), the observed and expected frequencies are calculated and a $\chi^2$ test is applied. Srivastava [24] proposed to check uniformity over extreme intervals close to 0 and close to 1 (for applications of random numbers in mining simulations, where extreme values will be critical when transformed to actual grades).

- **Serial test (k-dimensional uniformity)**: Given that numbers in the sequence should be independent, a good random number generator should produce pairs of numbers that uniformly fill the unit square, triplets that uniformly fill the unit cube, etc. The frequency of k-tuples $\{U_i, U_{i+1}, ..., U_{i+k-1}\}$ is calculated and compared with the expected frequency. We expect to have the same number of observations in each one of the $n^k$ equal-sized cells, where $n$ is the number of cells in which each dimension is divided and $k$ is the dimension or size of the k-tuple.

- **Poker test (partition test)**: This test was originally proposed by Kendall and Babington-Smith [11] for sequences of digits. When digits are arranged in blocks of five, there will be certain expectation of the numbers in which the five digits are all the same, the numbers in which there are four of one kind, and so on. Knuth [14] explains the "classical" poker test in the following manner: The numbers are arranged in $n$ groups of five successive integers, $(Y_{5j}, Y_{5j+1}, ..., Y_{5j+4})$, $0 \leq j < n$. We observe which of the following seven patterns each quintuple matches:

| | | | |
|---|---|---|---|
| All different: | abcde | Full house: | aaabb |
| One pair: | aabcd | Four of a kind: | aaaab |
| Two pairs: | aabbc | Five of a kind: | aaaaa |
| Three of a kind: | aaabc | | |

  A $\chi^2$ test is based on the number of quintuples in each category. A simpler version is proposed by Knuth. The test can also be applied using only four digits [12].

- **Gap test (Runs above and below the median)**: We can consider the gaps occurring between the same digit in the series. For example, one digit will be followed immediately by the same digit in about one-tenth of the cases (in this case, there will be no gap). There will be one digit (different) between two equal digits in about nine-hundredths of the cases. In about eighty-one-thousandths of the cases there will be a gap of two between a repeated digit, and so on.

  A generalization for a series of numbers (not just digits) would be to examine the length of gaps between occurrences of $U_j$ in a certain range. Let $0 \leq \alpha < \beta \leq 1$, we consider the length of consecutive subsequences $U_j, U_{j+1}, ..., U_{j+r-1}$ in which $U_{j-1}$ and $U_{j+r}$ lies between $\alpha$ and $\beta$ but the elements in the subsequence do not. This subsequence represents a gap of length $r$.

  The special cases $(\alpha, \beta) = (0, \frac{1}{2})$ or $(\frac{1}{2}, 1)$ originated the so called tests of runs above and below the mean (or the median). In order to implement this test, we have to produce another sequence from the sequence being tested, by counting the length of successive runs above and/or below the median. For example, the sequence:

  0.35, 0.56, 0.12, 0.11, 0.84, 0.76, 0.77, 0.45, 0.61, 0.51, ...

  would generate the following sequence of above/below the median observations:

  below, above, below, below, above, above, above, below, above, above, ...

  and the sequence of lengths of runs above/below the median would be:

  1, 1, 2, 3, 1, 2,...

The same procedure can be applied for any threshold (not only the median). Depending on the proportion of values above and below the threshold, the total number of runs above and below the threshold should follow a normal distribution with the following mean and variance [19]:

$$E[r] = 2 \cdot n \cdot p_A \cdot p_B$$

$$\sigma_r^2 = 4 \cdot n \cdot p_A \cdot p_B \cdot (1 - 3 \cdot p_A \cdot p_B)$$

where $p_A$ and $p_B$ represent the proportion (probability) of values above and below some threshold respectively, and $n$ is the total number of values in the sequence.

When the threshold is the median (or the mean) of a uniform distribution then, the parameters are simply:

$$E[r] = \frac{n}{2} \qquad \sigma_r^2 = \frac{n}{4}$$

A $\chi^2$ test can be applied to the observed values in order to determine if there is any significant difference with the expected value.

- **Runs up and down**: This test examines the length of monotone subsequences of the original sequence, i.e. segments which are increasing or decreasing.

  Again, to implement this test, we have to produce another sequence from the sequence being tested, by counting the length of successive runs up and down. For example, the sequence:

  0.35, 0.56, 0.12, 0.11, 0.84, 0.76, 0.77, 0.45, 0.61, 0.51, ...

  would generate the following sequence of runs up and down:

  up, down, down, up, down, up, down, up, down, ...

  and the sequence of lengths of runs up and down would be:

  1, 2, 1, 1, 1, 1, 1, 1, ...

  For an independent and uniform sequence of numbers, the number of runs up and down should come from a normal distribution with the following mean and variance:

$$E[r] = \frac{2 \cdot n - 1}{3}$$

$$\sigma_r^2 = \sqrt{\frac{16 \cdot n - 29}{90}}$$

  In this case, the conventional $\chi^2$ test has to be modified to take into account the fact that the number of runs of various lengths are negatively correlated [13, 14, 16] (covariances are used to make this correction).

- **Extreme values (maximum of $k$)**: Given that in most applications extreme values are important (e.g. in mining simulations), a test over the extreme values of the sequence is required. If we group the sequence into k-tuples, and we extract, for each k-tuple, the maximum value, then the distribution of maximums from k-tuples should show no serial correlation and should have a cumulative distribution that follows a power law: $F_k(x) = x^k$. Now, we must show that the distribution of maximums follows a power law. The probability of $max(U_1, U_2, ..., U_k) \leq x$ is the probability that $U_1 \leq x$ and $U_2 \leq x$ and ... and $U_k \leq x$, and this is the product of the individual probabilities, $x \cdot x \cdot ... \cdot x = x^k$. The closeness of the observed distribution to the expected can be checked comparing the distribution of $(max(U_1, U_2, ..., U_k))^k$ with the uniform distribution, using a $\chi^2$ test. The serial correlation can be calculated. This test can also be applied with the minimum of $k$.

- **Coupon collector's test**: This test consists in calculate the length of segments required to get at least one observation per cell, when the interval [0,1] is divided in some equally sized number of classes $d$. A $\chi^2$ test can be applied to the observed counting of length $r$. The corresponding probabilities are:

$$p_r = \frac{d!}{d^r} \left\{ \begin{array}{c} r-1 \\ d-1 \end{array} \right\}, \quad d \leq r < t \qquad p_t = 1 - \frac{d!}{d^{t-1}} \left\{ \begin{array}{c} t-1 \\ d \end{array} \right\}$$

  where $r$ is the length of the segment, $d$ is the number of classes and $t$ is some length such that all the segments longer than $t$ are put together.

- **Permutation test**: If we divide the sequence in k-tuples, then each k-tuple can have $t!$ possible relative orderings. The number of times each ordering appears is counted and a $\chi^2$ test is applied with $k = t!$ classes and with probability $1/t!$ for each ordering.

- **Test on subsequences**: All the tests previously presented can be applied to a subset of the sequence, so we can verify if these subsets behave equally random than the whole sequence.

## 2.2 Theoretical Tests

Some random number generators are suitable for analysis *a priori*, so that the parameters needed to generate a sequence can be understood and chosen properly. Linear congruential generators have been thoroughly studied [9, 13, 14, 18, 22]. Some other generators, as the additive method presented by Green, Smith and Klem [8] allows some theoretical analysis as well. Wikramaratna [28] shows some theoretical results for his additive congruential generator. The interested reader can check those references for further explanations of the tests.

Some authors recommend against methods that do not allow those analysis [22], such as the one proposed by MacLaren and Marsaglia [17];however those generators have shown to perform well for many applications [3].

# 3 Testing five random number generators

This section contains the results of testing five different random number generators. The tests presented here are only those which have proven to be the more effective to detect poor pseudo-random number generators [14, 24].

The following methods were tested:

- Linear Congruential Method (`lcorn`): the parameters of this generator are: $m = 2^{16} + 1$, $a = 75$, $c = 1$. Three seed numbers were used: 69069, 112063, and 76715.
- Mixed Congruential Method (`mcorn`): this is the generator proposed by MacLaren and Marsaglia [17].The seeds used are the same than those for `lcorn`.
- Additive Congruential Method (`acorn`): This is the generator proposed by Wikramaratna [28] in real arithmetic. The seeds used must be real values between 0 and 1. In this application the initial values are: 0.10, 0.81, and 0.12.
- Additive Congruential Method (`acorni`): This is the generator proposed by Wikramaratna [30] in integer arithmetic. Again, the seeds used are those for `lcorn`.
- `excel`: this generator comes with the commercial software `Microsoft Excel`. The pseudo-random sequences are generated without specifying a seed number.

For each generator, nine sequences of numbers have been created. Three sequences of 10,000 values, three of 30,000, and three of 90,000 values between 0 and 1.

## 3.1 Serial Correlation Test

The serial correlation was calculated using the routine `gam` of the public domain software GSLIB [4]. Results are presented in **Table 1**. Correlations greater than 0.02 in absolute value were highlighted. `lcorn` presents four of those high values for sequences of 10,000 numbers, however they are still not significant. `excel` also has one sequence with correlation greater than 0.02. All the random number generators passed this test, since all the correlations are acceptably close to zero.

## 3.2 Uniformity Test

The uniformity of the sequences was tested dividing the interval [0,1] into 100 subintervals ([0,0.01), [0.01,0.02), ..., [0.99,1]). A $\chi^2$ test was applied to the observed frequencies. The results are presented in **Table 2**. All the $\chi^2$ percentiles outside of the 90% central confidence interval were highlighted. `lcorn` failed this test when sequences of 30,000 and 90,000 numbers were used. `mcorn` and `acorn` seem to perform the best in this test.

## 3.3 K-dimensional Uniformity Test

After partitioning the space of 3, 4, and 5 dimensions regularly, the frequency of observed values in each subset should be approximately the same if the numbers are random. In 3-D the space was divided into $15^3 = 3375$ cells, in 4-D it was divided into $8^4 = 4096$ cells, and

| 10000 Data | | | | | | |
|---|---|---|---|---|---|---|
| Algorithm | Seed | h = 1 | h = 2 | h = 3 | h = 4 | h = 5 |
| LCORN | 69069 | 0.01241 | 0.00501 | -0.01182 | **0.02127** | 0.00371 |
| | 112063 | 0.01217 | -0.00515 | 0.00106 | 0.00048 | **-0.02291** |
| | 76715 | **0.02587** | 0.00015 | -0.00788 | 0.00931 | **0.02054** |
| MCORN | 69069 | 0.00415 | 0.00214 | 0.01294 | -0.00814 | 0.01826 |
| | 112063 | 0.01501 | -0.01339 | 0.00885 | 0.00631 | -0.00444 |
| | 76715 | -0.01881 | -0.00113 | 0.00805 | 0.00106 | 0.00052 |
| ACORN | 0.10 | -0.00323 | 0.00461 | -0.00661 | 0.00579 | 0.00957 |
| | 0.81 | 0.01117 | -0.00783 | 0.00027 | -0.00634 | 0.00761 |
| | 0.12 | 0.00106 | 0.01729 | 0.00602 | -0.00205 | -0.00486 |
| ACORNI | 69069 | -0.00318 | -0.00831 | -0.00647 | 0.00002 | 0.01980 |
| | 112063 | 0.01048 | 0.00654 | 0.00904 | 0.00217 | 0.01222 |
| | 76715 | -0.00810 | -0.00755 | 0.00938 | -0.01783 | 0.00373 |
| EXCEL | — | 0.00140 | -0.00602 | 0.00419 | -0.00220 | -0.01285 |
| | — | 0.00053 | -0.01690 | -0.00025 | -0.00903 | **0.02864** |
| | — | -0.01667 | 0.00894 | 0.01240 | -0.00189 | 0.00137 |
| 30000 Data | | | | | | |
| Algorithm | Seed | h = 1 | h = 2 | h = 3 | h = 4 | h = 5 |
| LCORN | 69069 | 0.01407 | -0.00198 | -0.00580 | 0.00700 | -0.00875 |
| | 112063 | 0.01490 | -0.00093 | 0.00497 | -0.00486 | -0.00137 |
| | 76715 | 0.01271 | -0.00278 | -0.00719 | 0.00513 | -0.00495 |
| MCORN | 69069 | 0.00287 | 0.00656 | 0.00737 | 0.00176 | 0.01153 |
| | 112063 | 0.00381 | -0.00892 | 0.00192 | 0.00185 | 0.00234 |
| | 76715 | 0.00750 | -0.00119 | 0.01183 | 0.01169 | -0.00305 |
| ACORN | 0.10 | -0.01548 | -0.00196 | 0.00473 | 0.00011 | 0.00683 |
| | 0.81 | -0.00086 | -0.00071 | 0.00155 | -0.00761 | -0.00681 |
| | 0.12 | -0.00124 | 0.01308 | -0.00241 | 0.00494 | 0.00354 |
| ACORNI | 69069 | 0.00059 | -0.01027 | -0.00921 | 0.00255 | 0.01482 |
| | 112063 | 0.00676 | 0.00775 | 0.01339 | -0.00249 | 0.00221 |
| | 76715 | 0.00112 | -0.00593 | 0.00509 | -0.01035 | -0.00144 |
| EXCEL | — | 0.00670 | 0.00648 | -0.00099 | -0.00416 | 0.00308 |
| | — | -0.00762 | 0.00476 | 0.00342 | -0.00472 | 0.00165 |
| | — | 0.00140 | 0.00038 | -0.00897 | 0.00318 | -0.00352 |
| 90000 Data | | | | | | |
| Algorithm | Seed | h = 1 | h = 2 | h = 3 | h = 4 | h = 5 |
| LCORN | 69069 | 0.01243 | 0.00017 | -0.00173 | 0.00183 | -0.00309 |
| | 112063 | 0.01292 | 0.00139 | 0.00140 | -0.00179 | -0.00167 |
| | 76715 | 0.01454 | -0.00013 | -0.00205 | 0.00116 | -0.00167 |
| MCORN | 69069 | 0.00345 | 0.00552 | 0.00289 | -0.00112 | 0.00471 |
| | 112063 | 0.00154 | -0.00329 | -0.00122 | 0.00077 | 0.00068 |
| | 76715 | 0.00297 | -0.00136 | 0.00255 | 0.00958 | 0.00109 |
| ACORN | 0.10 | -0.00726 | 0.00494 | -0.00331 | -0.00377 | 0.00019 |
| | 0.81 | 0.00509 | -0.00195 | -0.00065 | -0.00388 | -0.00622 |
| | 0.12 | 0.00076 | 0.00376 | 0.00073 | 0.00437 | 0.00008 |
| ACORNI | 69069 | -0.00177 | -0.00184 | -0.00487 | -0.00057 | 0.00174 |
| | 112063 | 0.00354 | 0.00328 | 0.00482 | -0.00270 | 0.00163 |
| | 76715 | 0.00171 | -0.00427 | 0.00036 | -0.00178 | 0.00024 |
| EXCEL | — | -0.00062 | -0.00570 | 0.00039 | 0.00339 | 0.00538 |
| | — | -0.00721 | 0.00331 | -0.00560 | 0.00153 | 0.00066 |
| | — | -0.00487 | 0.00319 | 0.00007 | 0.00010 | 0.00163 |

Table 1: Results of serial correlation test for 5 pseudo-random number generators and sequences of length 10000, 30000 and 90000.

| 10000 Data | | | | |
|---|---|---|---|---|
| Algorithm | Seed | 0.0 - 1.0 | 0.0 - 0.1 | 0.9 - 1.0 |
| LCORN | 69069 | 56 | 42 | 63 |
| | 112063 | **1** | 14 | 67 |
| | 76715 | 37 | 50 | 14 |
| MCORN | 69069 | 27 | 40 | 39 |
| | 112063 | 80 | 62 | 66 |
| | 76715 | 86 | 62 | 9 |
| ACORN | .10 | 24 | 72 | **95** |
| | .81 | 63 | 48 | 11 |
| | .12 | 6 | 47 | 27 |
| ACORNI | 69069 | 74 | 94 | 66 |
| | 112063 | 14 | **2** | **97** |
| | 76715 | 41 | 20 | 93 |
| EXCEL | — | 6 | 68 | 49 |
| | — | 41 | 84 | 14 |
| | — | 71 | **95** | 14 |
| 30000 Data | | | | |
| Algorithm | Seed | 0.0 - 1.0 | 0.0 - 0.1 | 0.9 - 1.0 |
| LCORN | 69069 | **0** | **0** | **0** |
| | 112063 | **0** | **0** | **0** |
| | 76715 | **0** | **0** | **0** |
| MCORN | 69069 | 38 | 31 | 40 |
| | 112063 | 60 | 73 | 16 |
| | 76715 | 24 | 40 | 25 |
| ACORN | .10 | 9 | 66 | 86 |
| | .81 | 92 | 80 | 7 |
| | .12 | 16 | 76 | 24 |
| ACORNI | 69069 | 54 | 65 | **96** |
| | 112063 | 47 | 81 | 45 |
| | 76715 | 64 | 60 | **98** |
| EXCEL | — | **98** | 64 | 85 |
| | — | 72 | **98** | 88 |
| | — | 6 | 8 | 22 |
| 90000 Data | | | | |
| Algorithm | Seed | 0.0 - 1.0 | 0.0 - 0.1 | 0.9 - 1.0 |
| LCORN | 69069 | **0** | **0** | **0** |
| | 112063 | **0** | **0** | **0** |
| | 76715 | **0** | **0** | **0** |
| MCORN | 69069 | 31 | 12 | **0** |
| | 112063 | **5** | 48 | 87 |
| | 76715 | 44 | 40 | 76 |
| ACORN | .10 | 23 | **99** | 60 |
| | .81 | 96 | 35 | 8 |
| | .12 | 44 | 79 | 11 |
| ACORNI | 69069 | 60 | 80 | 56 |
| | 112063 | 30 | 14 | 15 |
| | 76715 | 32 | 60 | 51 |
| EXCEL | — | 37 | 70 | 46 |
| | — | 33 | 11 | 94 |
| | — | 42 | 33 | 60 |

Table 2: Results of uniformity test for 5 pseudo-random number generators and sequences of length 10000, 30000 and 90000.

in 5-D it was divided into $5^5 = 3125$ cells. **Table 3** presents the $\chi^2$ percentiles for this test. `lcorn` failed this test for all the sequences tested. Again, `mcorn` seems to perform the best. `excel` also gives good results. `acorn` and `acorni` give acceptable results.

### 3.4 Runs Up and Down

The total number of runs up and down and the number of runs for each length were calculated and are shown in **Table 4**. The total number of runs should fall between the $5^{th}$ and $95^{th}$ percentile of the expected distribution. For 10,000 numbers, the acceptable interval is (6598,6736), for 30,000 it is (19880,20120), and for 90,000 the confidence interval is (59793,60207).

`lcorn` gives too few runs in most of the sequences, while `mcorn` failed in one sequence, which is acceptable. `acorn` and `acorni` failed in two cases. `excel` performed excellent in this test. In summary, we can say that `lcorn` failed and `excel` gave the best results, and the other generators gave acceptable results.

### 3.5 Runs Above and Below the Median

**Table 5** presents the total number of runs above and below the mean, and the detailed list of number of runs of different lengths. According to the limit distribution of the total number of runs, the observed number of runs should be into the interval $[m - 1.645 \cdot \sigma, m + 1.645 \cdot \sigma]$ . That means that for the sequences of 10,000 values, they should be within (4918,5082). In the case of 30,000 numbers, the number of runs should be into (14858,15142),and finally, for 90,000 numbers, it should be in (44753,45247).

The results again show that `lcorn` gives bad results for most of the sequences. The other generators only have minor problems with this test.

### 3.6 Extreme Values

Generators were tested for maximum values in a k-tuple. The distribution of maximums should follow a power law. A $\chi^2$ test was applied to compare the observed frequencies with the expected ones. **Table 6** presents the percentile of the $\chi^2$ test for each sequence.

`lcorn` failed the test for uniformity ($k = 1$) (see **Table 1**) and for higher values of $k$. All the other generator presented some problems, however in general they seemed to pass this test. `mcorn` gave the best results.

## 4 Discussion

Pseudo-random generators are required for geostatistical simulation. Since truly random numbers cannot be generated by computer, we need to quantify the randomness of the pseudo-random sequences generated by different algorithms. Many different tests have been proposed to measure randomness, however, only a few of them are able to detect important departures from randomness.

Some powerful tests have been applied to five commonly used generators. They have shown that the widely used `lcorn` generator does not give satisfactory results. Many applications that required random numbers a few years ago used this generator. We should

| 10000 Data | | | | |
|---|---|---|---|---|
| Algorithm | Seed | k = 3 | k = 4 | k = 5 |
| LCORN | 69069 | **4** | **1** | 30 |
| | 112063 | **0** | **3** | 11 |
| | 76715 | **1** | **2** | **4** |
| MCORN | 69069 | 47 | 70 | 28 |
| | 112063 | 51 | **5** | 6 |
| | 76715 | 34 | 82 | 60 |
| ACORN | .10 | 55 | 83 | 35 |
| | .81 | 6 | **96** | 20 |
| | .12 | 41 | 18 | 90 |
| ACORNI | 69069 | 27 | 71 | 26 |
| | 112063 | 69 | 49 | 75 |
| | 76715 | 93 | **99** | 26 |
| EXCEL | — | 7 | 19 | 6 |
| | — | 10 | 33 | 78 |
| | — | 35 | 62 | 83 |
| 30000 Data | | | | |
| Algorithm | Seed | k = 3 | k = 4 | k = 5 |
| LCORN | 69069 | **0** | **0** | **0** |
| | 112063 | **0** | **0** | **0** |
| | 76715 | **0** | **0** | **0** |
| MCORN | 69069 | 14 | 57 | 92 |
| | 112063 | 14 | 19 | 6 |
| | 76715 | 62 | 35 | 63 |
| ACORN | .10 | 86 | 49 | **98** |
| | .81 | 8 | 55 | 60 |
| | .12 | 35 | 22 | 85 |
| ACORNI | 69069 | 12 | 82 | 41 |
| | 112063 | 54 | 55 | 89 |
| | 76715 | **98** | 20 | 56 |
| EXCEL | — | 41 | 82 | 61 |
| | — | 57 | 87 | 47 |
| | — | 87 | 41 | 55 |
| 90000 Data | | | | |
| Algorithm | Seed | k = 3 | k = 4 | k = 5 |
| LCORN | 69069 | **0** | **100** | **0** |
| | 112063 | **0** | **100** | **0** |
| | 76715 | **0** | **100** | **0** |
| MCORN | 69069 | 47 | 63 | 59 |
| | 112063 | 11 | 53 | 7 |
| | 76715 | 65 | 41 | 83 |
| ACORN | .10 | 13 | 28 | 57 |
| | .81 | 69 | 59 | 76 |
| | .12 | 78 | **1** | 21 |
| ACORNI | 69069 | 58 | 52 | 67 |
| | 112063 | 81 | 68 | 65 |
| | 76715 | 91 | 76 | **99** |
| EXCEL | — | **97** | 36 | 68 |
| | — | 66 | 71 | 62 |
| | — | 89 | 15 | 20 |

Table 3: Results of k-dimensional uniformity test for 5 pseudo-random number generators and sequences of length 10000, 30000 and 90000.

| 10000 Data | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | Seed | Runs | l=1 | l=2 | l=3 | l=4 | l=5 | l=6 | l=7 | l=8 |
| LCORN | 69069 | 6638 | 4137 | 1809 | 549 | 121 | 19 | 3 | 0 | 0 |
| | 112063 | **6568** | 4053 | 1794 | 566 | 122 | 27 | 5 | 1 | 0 |
| | 76715 | **6582** | 4033 | 1861 | 539 | 122 | 23 | 4 | 0 | 0 |
| MCORN | 69069 | 6624 | 4115 | 1838 | 507 | 138 | 21 | 5 | 0 | 0 |
| | 112063 | 6642 | 4132 | 1827 | 543 | 119 | 18 | 3 | 0 | 0 |
| | 76715 | 6701 | 4219 | 1844 | 496 | 112 | 24 | 6 | 0 | 0 |
| ACORN | .10 | 6702 | 4222 | 1829 | 508 | 121 | 21 | 1 | 0 | 0 |
| | .81 | 6657 | 4166 | 1807 | 549 | 110 | 20 | 4 | 0 | 1 |
| | .12 | 6691 | 4222 | 1798 | 528 | 120 | 22 | 0 | 1 | 0 |
| ACORNI | 69069 | 6630 | 4099 | 1879 | 502 | 119 | 27 | 3 | 1 | 0 |
| | 112063 | 6679 | 4176 | 1844 | 527 | 108 | 22 | 2 | 0 | 0 |
| | 76715 | **6590** | 4058 | 1847 | 530 | 123 | 28 | 3 | 1 | 0 |
| EXCEL | — | 6618 | 4103 | 1821 | 549 | 121 | 21 | 3 | 0 | 0 |
| | — | 6611 | 4087 | 1822 | 557 | 130 | 13 | 2 | 0 | 0 |
| | — | 6716 | 4235 | 1837 | 514 | 108 | 16 | 6 | 0 | 0 |
| 30000 Data | | | | | | | | | | |
| Algorithm | Seed | Runs | l=1 | l=2 | l=3 | l=4 | l=5 | l=6 | l=7 | l=8 |
| LCORN | 69069 | **19817** | 12305 | 5401 | 1656 | 371 | 67 | 15 | 1 | 1 |
| | 112063 | **19863** | 12307 | 5516 | 1592 | 371 | 65 | 10 | 1 | 1 |
| | 76715 | **19834** | 12307 | 5428 | 1660 | 355 | 69 | 14 | 1 | 0 |
| MCORN | 69069 | 19935 | 12399 | 5553 | 1534 | 372 | 62 | 11 | 4 | 0 |
| | 112063 | 19884 | 12319 | 5516 | 1634 | 341 | 63 | 10 | 1 | 0 |
| | 76715 | 19975 | 12443 | 5546 | 1569 | 341 | 63 | 13 | 0 | 0 |
| ACORN | .10 | **20157** | 12707 | 5556 | 1467 | 362 | 60 | 4 | 1 | 0 |
| | .81 | 20073 | 12612 | 5480 | 1578 | 335 | 58 | 8 | 1 | 1 |
| | .12 | 20054 | 12616 | 5419 | 1615 | 327 | 71 | 5 | 1 | 0 |
| ACORNI | 69069 | 19905 | 12360 | 5548 | 1543 | 369 | 73 | 11 | 1 | 0 |
| | 112063 | 20047 | 12512 | 5606 | 1526 | 335 | 55 | 9 | 4 | 0 |
| | 76715 | **19842** | 12294 | 5482 | 1620 | 360 | 77 | 7 | 2 | 0 |
| EXCEL | — | 20036 | 12622 | 5408 | 1560 | 368 | 64 | 10 | 3 | 1 |
| | — | 20056 | 12533 | 5575 | 1545 | 342 | 54 | 6 | 1 | 0 |
| | — | 19946 | 12434 | 5500 | 1586 | 338 | 75 | 12 | 0 | 1 |
| 90000 Data | | | | | | | | | | |
| Algorithm | Seed | Runs | l=1 | l=2 | l=3 | l=4 | l=5 | l=6 | l=7 | l=8 |
| LCORN | 69069 | **59580** | 37010 | 16347 | 4892 | 1083 | 206 | 38 | 3 | 1 |
| | 112063 | **59601** | 37019 | 16399 | 4845 | 1092 | 204 | 37 | 3 | 2 |
| | 76715 | **59512** | 36887 | 16390 | 4908 | 1076 | 207 | 40 | 3 | 1 |
| MCORN | 69069 | **59783** | 37222 | 16498 | 4719 | 1137 | 172 | 29 | 6 | 0 |
| | 112063 | 59838 | 37274 | 16492 | 4799 | 1052 | 193 | 25 | 3 | 0 |
| | 76715 | 59989 | 37469 | 16543 | 4702 | 1070 | 173 | 31 | 1 | 0 |
| ACORN | .10 | **60387** | 38167 | 16320 | 4652 | 1044 | 170 | 28 | 6 | 0 |
| | .81 | 59954 | 37392 | 16620 | 4670 | 1040 | 201 | 26 | 4 | 1 |
| | .12 | 59975 | 37504 | 16425 | 4814 | 990 | 215 | 21 | 6 | 0 |
| ACORNI | 69069 | 60045 | 37528 | 16565 | 4725 | 1006 | 187 | 31 | 3 | 0 |
| | 112063 | 59925 | 37383 | 16555 | 4696 | 1072 | 191 | 22 | 5 | 1 |
| | 76715 | 59816 | 37194 | 16548 | 4824 | 1037 | 193 | 16 | 4 | 0 |
| EXCEL | — | 59904 | 37277 | 16625 | 4770 | 1032 | 173 | 21 | 5 | 1 |
| | — | 60162 | 37850 | 16276 | 4820 | 995 | 176 | 38 | 7 | 0 |
| | — | 60060 | 37579 | 16559 | 4663 | 1027 | 192 | 35 | 5 | 0 |

Table 4: Results of runs up and down test for 5 pseudo-random number generators and sequences of length 10000, 30000 and 90000.

| 10000 Data | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | Seed | Runs | l=1 | l=2 | l=3 | l=4 | l=5 | l=6 | l=7 | l=8 | l=9 | l=10 | l=11 | l=12 |
| Theory | | 5000 | 2500 | 1250 | 625 | 313 | 156 | 78 | 39 | 20 | 10 | 5 | 2 | 1 |
| LCORN | 69069 | 4938 | 2450 | 1198 | 662 | 309 | 143 | 92 | 44 | 24 | 5 | 5 | 3 | 2 |
| | 112063 | **4913** | 2389 | 1249 | 644 | 292 | 179 | 81 | 39 | 21 | 10 | 4 | 1 | 3 |
| | 76715 | **4904** | 2408 | 1203 | 646 | 338 | 145 | 79 | 37 | 28 | 7 | 3 | 3 | 3 |
| MCORN | 69069 | 4970 | 2479 | 1266 | 576 | 310 | 169 | 93 | 40 | 17 | 8 | 4 | 6 | 2 |
| | 112063 | 4937 | 2413 | 1252 | 654 | 311 | 151 | 74 | 47 | 13 | 6 | 5 | 3 | 3 |
| | 76715 | 5061 | 2549 | 1309 | 591 | 293 | 167 | 72 | 46 | 20 | 5 | 6 | 2 | 1 |
| ACORN | .10 | 5032 | 2543 | 1242 | 633 | 305 | 159 | 80 | 34 | 12 | 12 | 3 | 4 | 2 |
| | .81 | 4981 | 2472 | 1243 | 638 | 335 | 135 | 77 | 34 | 31 | 6 | 4 | 2 | 3 |
| | .12 | 5038 | 2548 | 1255 | 608 | 304 | 167 | 84 | 44 | 10 | 12 | 1 | 3 | 2 |
| ACORNI | 69069 | 5043 | 2513 | 1286 | 619 | 327 | 149 | 81 | 41 | 16 | 8 | 1 | 2 | 0 |
| | 112063 | 4937 | 2425 | 1251 | 615 | 337 | 142 | 84 | 50 | 8 | 10 | 9 | 4 | 0 |
| | 76715 | 5038 | 2558 | 1265 | 592 | 300 | 154 | 87 | 43 | 19 | 13 | 4 | 2 | 1 |
| EXCEL | — | 5038 | 2546 | 1265 | 582 | 334 | 163 | 87 | 28 | 14 | 8 | 3 | 3 | 4 |
| | — | 4980 | 2442 | 1320 | 585 | 318 | 165 | 71 | 36 | 21 | 11 | 7 | 1 | 0 |
| | — | 5067 | 2604 | 1229 | 623 | 309 | 144 | 81 | 32 | 21 | 12 | 8 | 2 | 1 |
| 30000 Data | | | | | | | | | | | | | | |
| Algorithm | Seed | Runs | l=1 | l=2 | l=3 | l=4 | l=5 | l=6 | l=7 | l=8 | l=9 | l=10 | l=11 | l=12 |
| Theory | | 15000 | 7500 | 3750 | 1875 | 938 | 469 | 234 | 117 | 59 | 29 | 15 | 7 | 4 |
| LCORN | 69069 | **14750** | 7225 | 3688 | 1933 | 905 | 491 | 264 | 123 | 67 | 26 | 12 | 5 | 9 |
| | 112063 | **14816** | 7317 | 3706 | 1851 | 940 | 518 | 247 | 120 | 60 | 33 | 10 | 4 | 7 |
| | 76715 | **14803** | 7275 | 3697 | 1941 | 926 | 468 | 253 | 117 | 69 | 29 | 12 | 4 | 7 |
| MCORN | 69069 | 14859 | 7428 | 3646 | 1848 | 936 | 494 | 270 | 122 | 52 | 28 | 16 | 12 | 4 |
| | 112063 | 15049 | 7506 | 3828 | 1870 | 920 | 466 | 226 | 130 | 53 | 18 | 13 | 8 | 4 |
| | 76715 | 14936 | 7440 | 3813 | 1789 | 904 | 492 | 255 | 121 | 54 | 33 | 18 | 10 | 4 |
| ACORN | .10 | **15236** | 7719 | 3799 | 1917 | 937 | 442 | 207 | 104 | 54 | 30 | 10 | 7 | 5 |
| | .81 | 15046 | 7542 | 3789 | 1824 | 995 | 429 | 236 | 104 | 72 | 23 | 12 | 9 | 6 |
| | .12 | 15077 | 7608 | 3724 | 1859 | 944 | 473 | 255 | 114 | 49 | 29 | 6 | 10 | 5 |
| ACORNI | 69069 | 15017 | 7419 | 3811 | 1941 | 943 | 455 | 246 | 104 | 46 | 29 | 8 | 8 | 3 |
| | 112063 | 15019 | 7540 | 3740 | 1867 | 924 | 469 | 242 | 134 | 40 | 28 | 23 | 4 | 4 |
| | 76715 | 14974 | 7481 | 3786 | 1829 | 938 | 424 | 261 | 125 | 68 | 37 | 15 | 3 | 5 |
| EXCEL | — | 15077 | 7610 | 3780 | 1805 | 942 | 482 | 217 | 119 | 62 | 26 | 16 | 8 | 8 |
| | — | 15017 | 7561 | 3743 | 1816 | 949 | 457 | 253 | 121 | 54 | 32 | 14 | 11 | 4 |
| | — | 14905 | 7404 | 3683 | 1914 | 961 | 474 | 223 | 134 | 57 | 29 | 13 | 7 | 5 |
| 90000 Data | | | | | | | | | | | | | | |
| Algorithm | Seed | Runs | l=1 | l=2 | l=3 | l=4 | l=5 | l=6 | l=7 | l=8 | l=9 | l=10 | l=11 | l=12 |
| Theory | | 45000 | 22500 | 11250 | 5625 | 2813 | 1406 | 703 | 352 | 176 | 88 | 44 | 22 | 11 |
| LCORN | 69069 | **44409** | 21890 | 11067 | 5703 | 2813 | 1455 | 748 | 363 | 201 | 83 | 39 | 16 | 19 |
| | 112063 | **44414** | 21927 | 11073 | 5618 | 2829 | 1495 | 741 | 369 | 187 | 91 | 36 | 16 | 20 |
| | 76715 | **44357** | 21845 | 11052 | 5680 | 2853 | 1443 | 745 | 365 | 201 | 84 | 39 | 16 | 19 |
| MCORN | 69069 | **44727** | 22324 | 11087 | 5621 | 2783 | 1425 | 761 | 355 | 185 | 92 | 48 | 26 | 9 |
| | 112063 | 44901 | 22332 | 11351 | 5617 | 2772 | 1422 | 678 | 370 | 172 | 83 | 56 | 25 | 11 |
| | 76715 | 44942 | 22416 | 11326 | 5593 | 2772 | 1403 | 700 | 378 | 174 | 85 | 48 | 22 | 15 |
| ACORN | .10 | 45147 | 22740 | 11163 | 5638 | 2829 | 1353 | 705 | 368 | 179 | 93 | 39 | 24 | 7 |
| | .81 | 44779 | 22293 | 11173 | 5635 | 2824 | 1406 | 748 | 338 | 187 | 80 | 31 | 30 | 24 |
| | .12 | 45138 | 22635 | 11360 | 5511 | 2774 | 1431 | 752 | 353 | 167 | 86 | 28 | 19 | 14 |
| ACORNI | 69069 | 45132 | 22577 | 11280 | 5688 | 2783 | 1442 | 701 | 340 | 160 | 98 | 23 | 20 | 10 |
| | 112063 | 45095 | 22675 | 11234 | 5595 | 2769 | 1377 | 730 | 364 | 162 | 87 | 62 | 15 | 14 |
| | 76715 | 44967 | 22365 | 11388 | 5627 | 2834 | 1308 | 716 | 371 | 175 | 103 | 39 | 17 | 15 |
| EXCEL | — | 44924 | 22321 | 11309 | 5729 | 2786 | 1391 | 691 | 345 | 171 | 87 | 40 | 21 | 19 |
| | — | 45227 | 22762 | 11211 | 5696 | 2780 | 1398 | 690 | 365 | 161 | 88 | 36 | 14 | 19 |
| | — | **45256** | 22760 | 11343 | 5607 | 2770 | 1375 | 704 | 348 | 177 | 93 | 40 | 22 | 7 |

Table 5: Results of runs above and below the median test for 5 pseudo-random number generators and sequences of length 10000, 30000 and 90000.

| 10000 Data | | | | | | |
|---|---|---|---|---|---|---|
| Algorithm | Seed | k = 1 | k = 2 | k = 3 | k = 4 | k = 5 |
| LCORN | 69069 | 56 | **100** | **98** | **98** | **98** |
| | 112063 | **1** | **99** | **100** | **100** | **95** |
| | 76715 | 37 | **100** | 85 | **100** | **99** |
| MCORN | 69069 | 27 | 54 | 73 | 37 | **2** |
| | 112063 | 80 | 29 | 78 | 85 | 8 |
| | 76715 | 86 | 78 | 70 | 85 | 66 |
| ACORN | .10 | 24 | 18 | 39 | 44 | 83 |
| | .81 | 63 | 91 | 94 | **1** | 10 |
| | .12 | 6 | 19 | 64 | 93 | 71 |
| ACORNI | 69069 | 74 | 19 | 6 | 16 | 46 |
| | 112063 | 14 | 30 | 76 | 76 | 94 |
| | 76715 | 41 | 12 | 50 | 40 | 40 |
| EXCEL | — | 6 | 24 | 15 | 28 | **5** |
| | — | 41 | 48 | 14 | 70 | 68 |
| | — | 71 | 72 | **96** | 63 | 46 |
| 30000 Data | | | | | | |
| Algorithm | Seed | k = 1 | k = 2 | k = 3 | k = 4 | k = 5 |
| LCORN | 69069 | **0** | **100** | **100** | **100** | **100** |
| | 112063 | **0** | **100** | **100** | **100** | **100** |
| | 76715 | **0** | **100** | **100** | **100** | **100** |
| MCORN | 69069 | 38 | 37 | 67 | 11 | **0** |
| | 112063 | 60 | 73 | 16 | 58 | 8 |
| | 76715 | 24 | 69 | 20 | 76 | 24 |
| ACORN | .10 | 9 | 64 | 15 | 15 | 34 |
| | .81 | 92 | 83 | 42 | 22 | 67 |
| | .12 | 16 | 15 | 81 | 85 | **96** |
| ACORNI | 69069 | 54 | **4** | 73 | 24 | 60 |
| | 112063 | 47 | 81 | 91 | **99** | **99** |
| | 76715 | 64 | 26 | 41 | 9 | 57 |
| EXCEL | — | **98** | 67 | 55 | 90 | 27 |
| | — | 72 | 87 | 76 | 86 | **99** |
| | — | 6 | **0** | 22 | 43 | 18 |
| 90000 Data | | | | | | |
| Algorithm | Seed | k = 1 | k= 2 | k = 3 | k = 4 | k = 5 |
| LCORN | 69069 | **0** | **100** | **100** | **100** | **100** |
| | 112063 | **0** | **100** | **100** | **100** | **100** |
| | 76715 | **0** | **100** | **100** | **100** | **100** |
| MCORN | 69069 | 31 | 18 | 28 | 7 | 24 |
| | 112063 | 5 | 38 | **2** | 22 | 28 |
| | 76715 | 44 | 93 | 20 | 57 | 11 |
| ACORN | .10 | 23 | 72 | 11 | **3** | **4** |
| | .81 | **96** | 66 | 25 | 19 | 62 |
| | .12 | 44 | 79 | 88 | 55 | 29 |
| ACORNI | 69069 | 60 | 10 | 39 | 56 | 32 |
| | 112063 | 30 | 14 | 68 | 9 | 52 |
| | 76715 | 32 | 42 | 28 | 73 | 55 |
| EXCEL | — | 37 | 75 | 56 | 71 | 60 |
| | — | 33 | 86 | 86 | 79 | 82 |
| | — | 42 | 63 | 50 | 53 | 82 |

Table 6: Results of maximum values test for 5 pseudo-random number generators and sequences of length 10000, 30000 and 90000.

test our pseudo- random number generators whenever a new powerful test is proposed. For the generators tested in this paper, `acorni` and `mcorn` performed the best, so they may be recommended as artifact-free pseudo- random number generators. `acorn` and the generator provided in `excel`, gave satisfactory results, but presented abnormal results more often than the previous two.

## References


[1] W. G. Cochran. The $\chi^2$ test of goodness of fit. *Annals of Mathematical Statistics*, 23:315–345, September 1952.

[2] D. A. Darling. The Kolmogorov-Smirnov, Cramer-von Mises tests. *Annals of Mathematical Statistics*, 28:823–838, December 1957.

[3] C. V. Deutsch. A comparative study of pseudo-random number generators. In *Report 5*, Stanford, CA, March 1992. Stanford Center for Reservoir Forecasting.

[4] C. V. Deutsch and A. G. Journel. *GSLIB: Geostatistical Software Library and User's Guide*. Oxford University Press, New York, 2nd edition, 1998.

[5] R. A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research*. Edinburg, 1938.

[6] J. N. Franklin. On the equidistribution of pseudo-random numbers. *Quarterly of Applied Mathematics*, 16:183–188, 1958.

[7] F. Gebhardt. Generating pseudo-random numbers by shuffling a Fibonacci sequence. *Mathematics of Computation*, 21:708–709, October 1967.

[8] B. F. Green, J. E. Keith Smith, and L. Klem. Empirical tests of an additive random number generator. *Journal of the ACM (Association for Computing Machinery)*, 6:527–537, October 1959.

[9] M. Greenberger. Notes on a new pseudo-random number generator. *Journal of the ACM (Association for Computing Machinery)*, 8:163–167, 1961.

[10] D. L. Harnett. *Statistical Methods*. Addison-Wesley, third edition edition, 1982.

[11] M. G. Kendall and B. B. Smith. Randomness and random sampling numbers. *Journal of the Royal Statistical Society*, 101:147–166, 1938.

[12] M. G. Kendall and B. B. Smith. Second paper on random sampling numbers. *Supplement to the Journal of the Royal Statistical Society*, 6:51–61, 1939.

[13] W. J. Kennedy Jr. and J. E. Gentle. *Statistical Computing*. Marcel Dekker, Inc., New York, 1980.

[14] D. E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison-Wesley, 1969.

[15] D. H. Lehmer. Mathematical methods in large scale computing units. In *Proceedings of the Second Symposium on Large Scale Digital Computing Machinery*, pages 141–146, Cammbridge, 1951. Harvard University Press.

[16] H. Levene and J. Wolfowitz. The covariance matrix of runs up and down. *Annals of Mathematical Statistics*, 15:58–69, March 1944.

[17] M. D. MacLaren and G. Marsaglia. Uniform random number generators. *Journal of the ACM (Association for Computing Machinery)*, 12:83–89, January 1965.

[18] G. Marsaglia. The structure of linear congruential sequences. In S. K. Zaremba, editor, *Applications of Number Theory to Numerical Analysis*, pages 249–285. Academic Press, London, 1972.

[19] A. M. Mood. The distribution theory of runs. *Annals of Mathematical Statistics*, 11:367–392, December 1940.

[20] J. V. Neumann. Various techniques used in connection with random digits. In *National Bureau of Standards symposium, NBS Applied Mathematics Series 12*, Washington, D. C., 1951. National Bureau of Standards.

[21] RAND Corporation. *A Million Random Digits with 100,000 Normal Deviates.* Free Press, Glencoe, IL, 1955.

[22] B. D. Ripley. *Stochastic Simulation.* John Wiley & Sons, New York, 1987.

[23] A. Rotenberg. A new pseudo-random number generator. *Journal of the ACM (Association for Computing Machinery)*, 7:75–77, January 1960.

[24] R. M. Srivastava. Testing of sequential indicator simulation. Personal communication, November 1991.

[25] W. E. Thomson. A modified congruence method of generating pseudo-random numbers. *Computer Journal*, 1:83,86, July 1958.

[26] L. H. C. Tippett. Random sampling numbers. *Tracts for Computers*, XV, 1927.

[27] W. J. Westlake. A uniform random number generator based on the combination of two congruential generators. *Journal of the ACM (Association for Computing Machinery)*, 14:337–340, April 1967.

[28] R. S. Wikramaratna. ACORN - a new method for generating sequences of uniformly distributed pseudo-random numbers. *Journal of Computational Physics*, 83:16–31, 1989.

[29] R. S. Wikramaratna. Acorn random number generator user documentation. User Documentation, October 1990.

[30] R. S. Wikramaratna. Theoretical analysis of the acorn random number generator, 1990. SIAM Conference on Applied Probability in Science and Engineering.