
Andy Mitchell
Course: Math 4901
Faculty Adviser: Barry McQuarrie
Second Reader: Peh Ng
University of Minnesota, Morris
Spring 2008

ABSTRACT
In this paper we will take a look at some of the most common and “best” pseudo-random number generators (PRNGs). We will discuss the various definitions and interpretations of randomness, as well as discuss some of the methods for evaluating the quality of PRNGs and the numbers they produce. We will complete the paper with a look at some examples of the uses of PRNGs, as well as look at some open questions in the area.

1. INTRODUCTION
In many fields of study we wish to deal with sequences of random numbers. We must rely on random number generators to provide us with these sequences. There are many such generators, each having benefits and drawbacks.

Randomness of a sequence is not a concept that easily fits one definition. There are several ways to define randomness, and many ways to measure it. Random sequences are present in many every day activities, in addition to many real-world applications; the concepts of random and pseudorandom are crucial to the study of data, formulation of ideas, algorithmic evaluation, and numerical analysis [24].

2. INTERPRETATIONS OF RANDOM
There are several ways that the concept of random can be interpreted and different requirements of random, which depend on the intended application.

2.1. Statistical Independence
According to probability theory, two events are statistically independent if the occurrence of one event has no bearing on the probability that the other occurs. When considering random variables, “two random variables are independent if the conditional probability distribution of either given the observed value of the other is the same as if the other's value had not been observed” [17]. We consider a sequence of numbers to be unpredictable if given the first \( n \) numbers of the sequence, it is impossible to accurately predict the next number in the sequence. It is important to note that unpredictability is considered in terms of reasonable time; that is, is there an efficient, polynomial time algorithm that can predict the next element of a random sequence. This question is closely related to the topics of computational complexity and NP-Complete algorithms.

2.2. Uniform Distribution
“When discussing single numbers, a random number is one that is drawn from a set of possible values, each of which is equally probable” [9]. Put more succintly, “if a random variable has any of \( n \) possible values \( k_1, k_2, ..., k_n \) that are equally probable, then it has a discrete uniform distribution” [21]. For a given,
sufficiently large, sequence of random numbers, all numbers should occur with approximately equal frequency. Consider the example of the roll of a six-sided die. The possible values are \( k = 1, 2, 3, 4, 5, 6 \). Each time the die is rolled each possible value has a probability \( 1/6 \).

### 3. TYPES OF RANDOM NUMBERS

In addition to the different interpretations of randomness, a sequence of numbers can be called random as one of three distinct types. The type of randomness is determined by the method used to generate the sequence, and by the properties of the sequence.

#### 3.1. True Random

For a number or a sequence of numbers to be truly random, they must be determined from physical randomness. There are many physical systems that exhibit randomness, ranging from the simple such as rolling a die or measuring variations in a computer user’s mouse movements or keystrokes, to the complex, such as measuring background noise or nuclear decay of an atom. Attention must be paid, however, to make sure that the system that is being measured doesn’t have unexpected patterns. For example, a computer user’s keystrokes may not be random, especially if the system buffers the input before it is interpreted [9].

The possibilities for physical sources are endless. The key idea is that subtle variations in the physical can results in large, unpredictable changes in the outcome – that the system exhibits chaos. Consider a sequence of numbers generated by the roll of the dice. The outcome of each roll is strictly determined by a physical process. However, it is generally considered impossible to properly measure all of the applicable variables and predict the outcome of the roll. The process is deterministic, but not realistically so.

This raises an interesting intellectual question; is there really such a thing as random? Many people argue that correctly predicting the outcome of any system is indeed possible given the necessary initial conditions, and as such nothing in nature is random. Some people agree that such prediction is possible, but that for practical purposes, this is the most “random” we can ever expect anything to be. Others still disagree that any event can be predicted given enough data. For example, is a system that collects data based on human interaction random? To answer this question would require evaluating whether or not a person’s thoughts and actions are random from an external sense, or are pre-determined from physical and chemical processes in the brain. Such a question takes the topic from the mathematical to the philosophical, which is quite beyond the scope of this paper [2, 9].

Some interesting examples of generating truly random numbers are the use of snapshots of lava lamps, such as in the Lavarand generator by Silicon Graphics, and the use of atmospheric noise measured by a radio, a technique used by Random.org, a service which provides truly random numbers [9].

It is often difficult and time consuming to produce large sequences of true random numbers, because of their dependence on physical phenomena. Not only can implementing such a system be difficult, but the physical measurements necessary to provide true random numbers can often be slow or computationally complex. The benefit however, is that truly random generators are non-periodic (non-repeating), and non-reproducible, which means that they are very well applied to systems that require secure randomness.

In most circumstances, the strict requirements of a truly-random number are not necessary. True-random numbers are needed in applications that require unpredictability more-so than uniformity. Common uses of truly-random numbers are encryption, gambling, or any application that requires strong security. An example is the RSA crypto-system which uses public and private keys that are generated by very large random primes. The prime numbers used to generate these keys must be secure; the generator must be non-reproducible.
3.2. Pseudorandom

Sequences of numbers are said to be pseudorandom if they “look random” and exhibit disorder [11]. A pseudorandom sequence is a deterministic sequence of numbers in [0, 1] having the same statistical properties as a sequence of independent random variables uniformly distributed on (0, 1).

Pseudorandom numbers are much easier to produce than truly random numbers. Because they are determined arithmetically, the same generator can be used again and again to produce the same sequence, which can be useful for testing and fixing software [18].

Pseudorandom number generators are algorithms that use mathematical formulae or pre-calculated tables to produce sequences of numbers that appear random. Research has shown that modern algorithms for generating pseudo-random numbers are so good that the numbers look exactly like they were truly random. Pseudorandom number sequences are also efficiently generated; that is, they can produce many numbers in a short time. It is important to note that due to the computational nature of pseudorandom number generators, they are periodic, meaning the sequence will eventually repeat itself. This is not a desirable trait, but the period is typically so large that for all practical purposes, it is ignored [9]. It is a common misconception that a pseudorandom sequence will begin again once the first element has been repeated. This is true in PRNGs where the n+1-th element is determined by the n-th element, but not in PRNGs that use several previous elements. This will be explained in further detail in Section 5.

Pseudorandom number sequences are the most commonly used “random” numbers, used in most modeling and approximating applications. Pseudorandom numbers are quite suitable for applications where many numbers are required and where it is useful that the same sequence can be reproduced. PRNGs are not suitable in applications that require rigorous unpredictability [9].

3.3. Quasi-random

Quasi-random numbers are a unique type of randomness. Quasi-random numbers are those that exhibit many of the same properties and characteristics of random numbers, such as uniform distribution, but are generated in a very systematic and observable pattern. Quasi-random numbers are neither truly-random nor pseudorandom.

Quasi-random number sequences are sequences that are said to have low-discrepancy. “Roughly speaking, the discrepancy of a sequence is low if the number of points in the sequence falling into an arbitrary set B is close to proportional to the measure of B, as would happen on average (but not for particular samples) in the case of a uniform distribution.” [14]. One definition of the discrepancy of a set of N points is as follows [13]

If Q is a rectangle contained in I^s (the s-dimensional unit interval) and m(Q) is its volume, then the discrepancy DN of the sequence \{x_i\} of N points is

\[
D_N = \sup_{Q \subseteq \Omega} \left| \frac{\text{# of points in } Q}{N} - m(Q) \right|
\]

Discrepancy is essentially a measure of the “clumping” effect that a sequence of numbers can exhibit. Quasi-random sequences are chosen so that such discrepancy is minimized, and the numbers are well distributed. Because quasi-random numbers have low-discrepancy (good distribution), yet are very predictable, they are suited only for modeling and approximating, and are not suited for secure needs.
4. EVALUATION OF A PSEUDORANDOM GENERATOR

There are many well-studied and statistically backed approaches to evaluating the performance of a pseudorandom number generator, based on the criteria mentioned in Section 2 (uniform distribution, statistical independence and unpredictability), as well as other statistical criteria. We shall discuss two common sets of tests, as well as other criteria.

4.1. Diehard Battery of Tests

George Marsaglia, Professor Emeritus of Pure and Applied Mathematics and Computer Science at Washington State University and Professor Emeritus of Statistics at Florida State University, has developed the Diehard Battery of Tests of Randomness [15, 16]. This is a series of tests based on the statistical properties of sequences of numbers. A few of these tests are listed below.

4.1.1. Birthday Spacings

Choose random points on a large interval. The spacings between the points should be asymptotically Poisson distributed.

4.1.2. Overlapping Permutations

Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability.

4.1.3. Parking Lot Test

Randomly place unit circles in a 100 x 100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully “parked” circles should follow a normal distribution.

4.1.4. Minimum Distance Test

Randomly place 8,000 points in a 10,000 x 10,000 square, and then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean.

4.1.5. Random Spheres Test

Randomly choose 4,000 points in a cube of edge 1,000. Center a sphere on each point, whose radius is the minimum distance to any other point. The smallest sphere’s volume should be exponentially distributed.

4.1.6. The Squeeze Test

Multiply $2^{31}$ by random floating point numbers on [0, 1) until 1 is reached. Repeat this 100,000 times. The number of floating point numbers required to reach 1 should follow a certain distribution.

4.1.7. Overlapping Sums Test

Generate a long sequence of random floating point numbers on [0, 1). Add sequences of 100 consecutive floating point numbers. The sums should be normally distributed.

4.2. IMSL Libraries

The International Mathematics and Statistics Library is a collection of numerical analysis software libraries, implemented in many common programming languages, that includes common tests of randomness [1].

4.3. Period

The period of a generator is the length of a produced sequence before it begins to repeat itself. The amount of random numbers needed for any given application is generally known, and the period of a generator is easily computed. As such, it is desirable to use a generator with a period longer than the amount of points to be used (to avoid repeating). Generators now exist with periods as high as $2^{2^{200,000}}$. 
4.4. Computational Complexity

Computational complexity deals with the amount of computer resources required for the execution of a pseudorandom number generator. Because most applications of PRNGs require vast amounts of numbers, usually produced via complex algorithms, it is imperative that these numbers be generated efficiently; in a relatively-fast amount of time, and with minimal use of system resources.

4.5. Entropy

Entropy is a measure of uncertainty associated with a random variable. Entropy quantifies the information contained in a sequence of numbers, usually represented in bits. The entropy of a sequence is the minimum message length necessary to communicate information about the sequence; that is, the minimum amount of information necessary to be able to accurately predict the sequence. Therefore, high entropy relates to stronger randomness [22].

The information entropy of a discrete random variable $X$ with possible values $\{x_1, \ldots, x_n\}$ is defined as

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

where $P(x_i) = P(X = x_i)$ is the probability mass function of $X$.

4.6. Plotting

Plotting is a simple way of determining if a certain set of data is sufficiently random. When plotting random 2-dimensional or 3-dimensional points, many generators and sets of data show unwanted properties, such as “empty space” – large regions without points, or regions that contain many more points than other regions, which is known as “clumping”.

5. TYPES OF PRNGS

Over the years there have been many types of pseudo-random number generators. As definitions of randomness have changed, and the use of random number generators has increased – both an increase in the use of generators, as well as an increase in the amount of random numbers needed in a calculation – new generators have come about, and others have been deemed obsolete [24]. This section presents several common PRNGs, as well as analysis of them. Some recommended further readings are [5, 6, 7, 12].

5.1. Random Tables

Random number tables are one of the earliest sources for random samples. Before it became common to use computers to generate random numbers, utilizing a table was more effective than manually selecting the sample, via means such as rolling dice or shuffling cards. Tables were created that displayed many of the desired properties of statistical randomness, but the tables still had many drawbacks. The primary drawback is a lack of enough numbers. The largest tables number up to 100,000 digits, which for any large computation is typically insufficient. Another key problem is that using a random table is not secure; given enough digits, it is possible to determine which table is being used, and thus predict the rest of the sequence.
This plot indicates that the random table used shows a fairly uniform distribution, visually, without much clumping or empty space. Using random tables, however, is no longer common, as the method is slow, provides limited results, and is ultimately predictable; if it is known that a random number table is being used, one need only check the most common tables to determine which is the source.

5.2. **Linear Congruential**

The linear congruential generator (LCG) is one of the most popular forms of pseudorandom generators. It was proposed by Lehmer in 1951, and is widely used [8]. The generation is defined by simple linear recursive relations with modulus \( p \), usually chosen as a large prime or of the form \( 2^w \). A common choice of the modulus is \( 2^{31} - 1 \), which is prime. LCGs are defined by the form

\[ x_{n+1} = ax_n + b \pmod{M}, \]

where \( 0 \leq x_n \leq M - 1 \), and \( x_0 \) is the seed.

The seed is the initialization of a generator, and it also allows the regeneration of a sequence. Even though \( a \) and \( b \) can be chosen to maximize the period of the generator, the period is always limited by the modulus \( p \). Linear generators perform quite well under randomness tests, but due to the limited period, and increased requirements for random sequences, LCGs may not be sufficient in all cases. Marsaglia has shown that LCGs can perform poorly when generating overlapping sequences, and that they can lead to poor results, such as in Monte Carlo multiple-integration [8].
5.2.1. RANDU

One particular linear congruential generator is worth mention. The RANDU generator was very popular in the 1960’s and 1970’s, and was implemented in many programming languages. RANDU had the recurrence relation

\[ X_{k+1} = (65539X_k) \mod 2^{31} \]

This is simply a linear congruential generator with \( a = 65539 \) (= \( 2^{16} + 3 \)), \( b = 0 \), and \( M = 2^{31} \). These numbers were chosen because they allowed for very quick computations on systems using 32-bit words. However, if we consider explicit calculations, \( \mod 2^{31} \), we find a disappointing result

\[
X_{k+2} = (2^{16} + 3)X_{k+1} \\
= (2^{16} + 3)X_k \\
= (2^{32} + 6 \times 2^{16} + 9)X_k \\
= [6 \times (2^{16} + 3) - 9]X_k
\]

We have the correlation between points

\[ X_{k+2} = 6X_{k+1} - 9X_k \]

We are now aware that RANDU has terrible randomness properties, as we can see in the following plot.

Figure 2. Plot of random 2-dimensional points obtained by a linear congruential generator.
RANDU is now widely regarded as one of the worst random number generators of all time, and many results obtained through its use have been rendered unusable. Donald Knuth, a renowned professor of computer science at Stanford, has expressed this quite simply: *its very name is enough to bring dismay into the eyes and stomachs of many computer scientists!*

### 1.1. Multiple Recursive Generator

The *multiple recursive generator* (MRG) is generated from a linear combination of the past $k$ random numbers in the sequence. The maximum period for an MRG is $p^k - 1$, rather than $p$ for LCG [8]. A multiple recursive generator sequence $X_i$ is determined by a degree $k$ primitive polynomial

$$f(x) = x^k - \alpha_1 x^{k-1} - \ldots - \alpha_k$$

and being of the form

$$X_i = \alpha_1 X_{i-1} + \ldots + \alpha_k X_{i-k}$$

While the MRG has a much longer period than the LCG, it is also more computationally intensive (roughly $k$ times more intensive) [8].
1.2. Matrix Congruential Generator

The Matrix Congruential Generator (MCG) is defined by the equation

$$X_i = BX_i \mod p, i \geq 1$$

Where the $X_i$ are $k$-dimensional vectors, and $X_0 \neq 0$, and $B$ is a $k \times k$ matrix, and $p$ chosen as a large prime number, similar to the LCG [8]. As in multiple recursive generators, the period of an MCG is $p^k - 1$. The MCG can be seen as the $k$-dimensional extension of the LCG. Much like the multiple recursive generator, the MCG offers long period at the cost of decreased computational efficiency.
1.3. Lagged Fibonacci Sequence

The Lagged Fibonacci Sequence is given by the recurrence relation

\[ S_n = S_{n-j} \odot S_{n-k} \pmod{M}, \quad 0 < j < k < n \]

with \( \odot \) as any binary operation. Similar to the linear congruential generator, M is usually chosen to be a large power of 2. The period of a lagged Fibonacci sequence is generally \( 10^{10} \) or better, and can be as high as \( 10^{100} \).

A property of this generator is that the period is maximal when the polynomial

\[ y = x^k + x^j + 1 \]

is primitive over the integers, mod 2. As such, \( k \) and \( j \) must be carefully chosen, and the generator can be difficult to initialize. In even the simplest generators of this type, no less than 10 values are needed as a seed.

It can be proven that if the binary operation is addition or subtraction the period will be \((2^k - 1) \times 2^{k1}\), if the binary operation is multiplication the period will be \((2^k - 1) \times 2^{k3}\), and if the binary operation is the bit-level XOR (exclusive or) operation the period will be \(2^k - 1\). The latter case is known as the Generalized Feedback Shift Register.

1.3.1. Subtract with Borrow

George Marsaglia and Florida State colleague Arif Zaman developed a class of random number generators that produce “astonishingly long” sequences of random numbers [24]. These classes of generators, specific lagged Fibonacci sequences, are called Subtract with Borrow and Add with Carry. These methods make the simple change of adding one additional bit-level operation, and are strongly rooted in number theory, which means that not only do they show many of the properties that are desired for random number generators, but these properties can be proved mathematically. These generators have periods that are \(10^{250}\) or more, far surpassing the possibilities of past techniques [24]. In fact, the Mathematica software suite, which is used in many scientific and mathematical fields, used Subtract with Borrow as its standard random generator, until recently.

![Figure 4. Plot of random 2-dimensional points obtained by Mathematica subtract-with-borrow generator.](image)
1.4. Corput Sequences (Quasi-random)

Corput sequences are a common form of a quasi-random generator. Corput sequences are constructed by reversing the base $n$ representation of the sequence of natural numbers ($1, 2, 3, \ldots$). For example, the binary representation of the first five natural numbers would be $(1.0, 10.0, 11.0, 100.0, 101.0)$. Reversing each number within the sequence we have $(0.1, 0.01, 0.11, 0.001, 0.101)$. Converting back into decimal (base 10), we have $(0.5, 0.25, 0.75, 0.125, 0.625)$. Corput sequences of different bases can be combined to generate multi-dimensional quasi-random points.

![Figure 5](image)

**Figure 5.** Plot of random 2-dimensional points obtained by linear Corput sequences with bases 2 and 3

1.5. Physical

There are many ways that random numbers can be generated by physical phenomena. One method is generating random digits using radiation measurements from radioactive nuclide, as described in [3]. Specifically, the method described measures gamma rays emitted by radioactivity. The method is based on several characteristics:

(i) Nuclei decay independently

(ii) Gamma rays produce enough energy to allow reliable measurement apart from background noise

(iii) Contamination by other radiation such as cosmic rays is unimportant, due to the nature of cosmic rays and gamma radiation

Inoue, Kumahora, and Yoshizawa compare the performance of this method to the results of several common PRNGs.

2. TECHNIQUES FOR IMPROVING PRNGS

Although many random generators pass criteria to be considered pseudo-random, there are still limitations, such as the period of a generator. Many pseudorandom generators suffer from intrinsic problems that cause poor performance in higher dimensions, such as the lattice-structure that is often formed in higher dimension when using an LCG, or clumping [1]. In this section we will discuss a technique for improving the behavior of PRNGs, or increasing the period of a generator.
2.1. Combination

A technique for improving performance is to combine the results of two or more different generators [10]. Consider two linear congruential generators with sequences $X_i$ and $Y_i$. These two generators can be combined in the following way [10]:

$$Z_i = X_i + Y_i \mod (m),$$

where $X_i$ and $Y_i$ take on values in $\{0, 1, \ldots, m-1\}$.

This provides us with a sequence of numbers $Z_i$ on $\{0, 1, \ldots, m-1\}$, but because it is formed by two separate LCGs with period $m$, it will have twice the period ($2m$). The generators being combined need not be linear congruential generators, and they need not be the same type of generator. From this simple technique, which does not require much more computation, we can get large increases in period and overall performance of a generator.

3. USES OF PSEUDORANDOM GENERATORS

Below are some examples of the uses of random number generators.

3.1. Monte Carlo Integration Using Subtract-with Borrow

The Monte Carlo method is a computational algorithm using randomly sampled points. It is commonly used in simulating physical and mathematical systems [20]. One such use is integral approximation. An integral such as,

$$\int_a^b f(x)dx$$

can be approximated by

$$(b - a) \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

where $x_i$ is a point randomly chosen within the domain of $f$ and $n$ is the number of random points used in the approximation.

Consider the following example.

$$f(x, y) = 7x^3 + 4y^4 + x^2y^3 + 10$$

We can use the Monte Carlo method outlined above (generalized to two dimensions) to approximate the following integral

$$\int_0^5 \int_0^5 f(x, y)dxdy$$

We will use 1,000,000 random points and calculate

$$(5 - 0) \times (5 - 0) \times \frac{1}{1,000,000} \sum_{i=1}^{1,000,000} f(x_i, y_i)$$

Where $x_i, y_i$ are points in the range $[0, 5]$ chosen randomly using the Subtract With Borrow method.

Running the above computation in Mathematica, we find an approximation value of 13187.385. We would like to analyze the error in our results. From Calculus techniques we find the indefinite integral to be

$$F(x, y) = \frac{1}{12} xy(120 + 21x^3 + 96y + x^2).$$
We calculate the definite integral to be approximately 13229.167. From this we can calculate the error

\[ err = \frac{|13187.385 - 13229.167|}{13229.167} \approx 0.3\% \]

We find an error of roughly 0.3%, which is suitable for many applications. Note that these results could be improved by using a larger sample of random points – one million is relatively small. As we have seen, there are generators capable of quickly providing billions of random points.

3.2. Monte Carlo Approach to Particle Simulation

An excellent description of an application of the Monte Carlo approach in a real physical experiment, as well as analysis and comparison with recorded experimental results can be found in [13].

3.3. Multi-variate Monte Carlo Integral Approximation

An in-depth explanation and analysis of a multivariate integral approximation using the Monte Carlo algorithm is given in [4].

4. OPEN QUESTIONS

Some questions pertaining to randomness and random number generation follow.

4.1. Future Statistical Tests of Random Numbers

There are many pseudorandom generators that were believed to be quite good until George Marsaglia released his Diehard battery of tests. Commonly used current generators pass all existing tests; is it reasonable to assume that one day there will be tests for which our current techniques fail. What might such tests be?

4.2. Future Improvements

We have seen that there are some simple techniques for improving the performance of a pseudorandom generator. What other techniques could be used to improve existing PRNGs?

4.3. Predictability and Security

It is an open question, and one central to the theory and practice of cryptography, whether there is any way to distinguish the output of a high-quality PRNG from a truly random sequence without knowing the algorithm(s) used and the state with which it was initialized. The security of most cryptographic algorithms and protocols using PRNGs is based on the assumption that it is infeasible to distinguish use of a suitable PRNG from a random sequence. [19].

REFERENCES