Lund University

How to generate almost random numbers

Author:Anton Sjödin – *Supervisor:Phillipp Birken*

December 21, 2022

LUND UNIVERSITY

FACULTY OF SCIENCE

Contents

Abstract

In this thesis we will explore methods to generate random numbers from any distribution.

The basic idea behind generating random numbers is to use a mathematical sequence for which it should be impossible to guess what the next number is without knowing exactly how the sequence is generated. Although any hard to guess sequence would work, this thesis focuses on the linear congruential sequence, $X_{n+1} = aX_n + c \mod m$. There are two advantages to this sequence, firstly it is quick, secondly it has underlying theorems on how to use it. These theorems which have been studied in Donald Knuths the Art of Computer Programming volume 2, will be explored and proven in this paper.

To get any distribution we will also need the inversion and rejection methods. The effectiveness of these as methods to get random numbers is what W.Hörmann and G.Derflinger has studied in their paper. We will do the same testing of those methods and try to replicate their results. Replicating their results failed, however the same conclusion can be drawn.

Popular Abstract

A random number generator is a method to produce random numbers. In actuality the number we generate are not truly random, they are completely predetermined by a mathematical sequence, but without knowledge of how exactly the sequence is defined the numbers will seem and act like random numbers.

The method studied in this thesis can be thought of as a shuffled deck of cards with numbers from one to a very large number on each card. To generate each number one draws a card and puts it at the bottom of the deck. So if one knows how exactly one shuffled the deck, one can predict what number is coming next. For anyone who doesn't know exactly how the deck was shuffled the numbers will seem random. This metaphor just like the actual method shows that if we generate the number 5 then that number will not appear again until one has gone through the entire deck. Also a deck of card such as this will generate any number with the same probability. However sometimes one may want some numbers to be more likely to appear than others, so we also study two methods to generate numbers with other distributions.

Chapter 1

Introduction

Numbers that are randomly generated, or at least very hard to predict, has uses in various areas of science as well as outside of the sciences. For example, there exists monte carlo methods which employ random numbers to numerically calculate a result. Or one could use it to test statistical results. However getting a lot of random numbers can be a tedious process. For example doing 100 coin flips and recording the results can take quite a while. Thus before the computer there used to be large table of random number[7, pp. 1-6]. With the invention of the computer, the invention of random number generators (RNG) soon followed. A RNG is a method from which one calculates a number, such that it should be hard to predict, and seem random to someone who doesn't know the inner workings of the RNG.

In 1946 John von Neumann suggested the "middle-square" method. In which a number is generated by taking the square of the previous number and then use the middle digits as the next number[8]. However this method tends to either get stuck at 0 or a relatively short string of repeating digits.[7, pp. 1-6].

When generating random numbers it is important that one uses a method that has been studied, and has underlying theory about how it works, and which has been well tested to make sure it truly acts as if it was random.

A RNG which has been well studied is the linear congruential method(LCM),

$$X_{n+1} = aX_n + c \mod m$$

which will generate uniformly distributed numbers, where X_0, a, c and m are integers. This method is the focus of this thesis. The theory of choosing the integers above will be studied and theorems related to it will be proven. The theory behind the LCM is largely based on Donald Knuths the art of computer programming volume 2, chapter 3.

Once we have the LCM to generate numbers from a uniform distribution, we will introduce the inversion method. The inversion method converts a number from a uniform distribution between 0 and 1 by plugging it into the inverse of the wanted distributions cumulative distribution function (CDF). The theory for the inversion method is also taken from the art of computer programming

However as the inversion method requires an analytical CDF its not able to generate numbers from any distribution. To fill the gaps we also introduce the rejection method which will generate one number and then we randomly accept or reject it with a probability such that the accepted numbers will be from the wanted distribution. This method has been studied in the paper of W.Hörmann and G.Derflinger. With these three methods combined we will be able to generate numbers from any distribution.

We will finish the thesis by trying to replicate the results of W.Hörmann and G.Derflinger. The attempt to replicate the results of W.Hörmann and G.Derflinger did fail, however the results can still be used to draw the same conclusion, that a small multiplier in the linear congruential sequence is bad when one is pairing it with the rejection method.

Chapter 2

Generation of uniformly distributed numbers

2.1 Definitions and properties

To understand this thesis you are expected to have some prior mathematical understanding. We will however here explore a brief explanation of modular arithmetic as it lays the ground work for the generator. We start with a few definitions.

Definition 2.1.1. *a is a divisor of b, if there exists some integer c such that $b = ac$, we write this as $a|b$. [4, pp.83-85]*

Definition 2.1.2. *If $a|b$ and $a|c$ then a is a common divisor of b and c. In particular if it is the greatest such number it is called the greatest common divisor (GCD).[4, pp. 83-85]*

Definition 2.1.3. *If the greatest common divisor of a and b is 1, then they are said to be coprime or relatively prime.[4, pp. 83-85]*

Definition 2.1.4. *For any $a, b \in \mathbb{N}$ we say that h is a common multiple of a and b if $a|h$ and $b|h$. We say that a common multiple h of a and b is a least common multiple if h divides every common multiple of a and b. We denote the least common multiple of two numbers by $lcm(a, b)$. [4, pp. 83-85]*

Definition 2.1.5. *Two integers a, b are said to be congruent modulus m if,*

$$(a - b)|m.$$

This can be written as,

$$a \equiv b \mod m.[4, pp.83 - 85]$$

In words we can explain it as two numbers are congruent modulus c, if their remainder when divided by c are equal. A direct consequence of this definition is that

$$a = b + qm \textit{ for some integer } q. \tag{2.1}$$

With modular arithmetic we also have some important properties.

If we have $a \equiv b \mod m$ and arbitrary integers k,

Property 1. $a + k \equiv b + k \mod m$

Property 2. $ak \equiv bk \mod m$

Property 3. $ak \equiv a \mod m$ if k is coprime m[4, pp. 106-107]

2.2 The linear congruential sequence

Computers are inherently unable to produce randomness, since given a certain input they will always act in the same way, giving the same output. If we multiply 3 by 5 the computer will always get 15. If we take $15 \mod 11$ we will always get 4. However if we repeatedly multiply the result by 5 and take modulus 11, we get a sequence

$$3, 4, 9, 1, 5.$$

This sequence, whilst not random makes it hard to guess what the next number will be without doing the calculations. However, the very next number in the sequence is 3 after which it will continuously repeat the same five numbers. We can define such sequences in general as,

$$X_{n+1} = aX_n \mod m, \tag{2.2}$$

where a, m and x_0 are integers. By making better choices for the modulus m, the multiplier a and the starting value X_0, we can get a different sequence, with a longer period giving a more "random" sequence. However we will study a slightly different sequence, where we include an increment c,

Definition 2.2.1. *For integers a, c, m and x_0, a linear congruential sequence (a, c, m, x_0), is a sequence of the form,*

$$X_{n+1} = aX_n + c \mod m. \tag{2.3}$$

since it can yield an even longer sequence [7, pp. 15-24]. However both sequences are of interest, since equation 2.2, has slightly faster calculation times whilst still having the possibility to give a sequence that is long enough for most problems. If we know X_n, we may be interested in X_{n+k}, which we present as a lemma,

Lemma 2.2.2. *If X_n is the nth number of a linear congruential sequence where $a \neq 1$ then the $(n+k)$th number is given by*

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

[7, pp. 15-24].

Proof. We will prove this by induction over k. Thus we first let k=1 and get,

$$X_{n+1} = aX_n + c \mod m = a^1 X_n + \frac{(a^1-1)c}{a-1} \mod m.$$

Next we make the inductive hypothesis, that it holds for $k = t$,

$$X_{n+t} = a^t X_n + \frac{(a^t-1)c}{a-1}.$$

We now use equation (2.3) to generate X_{n+t+1} from X_{n+t}

$$X_{n+k+1} = aX_{n+t} + c \mod m = a(a^t X_n + \frac{(a^t-1)c}{a-1}) + c \mod m =$$
$$a^{t+1}X_n + \frac{(a^{t+1}-a)c}{a-1} + c \mod m = a^{t+1}X_n + \frac{(a^{t+1}-a)c + (a-1)c}{a-1} \mod m$$
$$= a^{k+1}X_n + \frac{(a^{k+1}-1)c}{a-1} \mod m,$$

which finishes the proof. □

2.3 Choice of m

Let us consider the modulus parameter. We note that X_n can at most be $m-1$ due to the modulus. Further a given number can only appear once per period. Thus in the longest possible linear congruential sequence, we can imagine every integer between 0 and $m-1$ will appear once, so we see that our sequence will never have a period longer than m. Since a long period is one of the things we want, we can conclude that we want m to be large. Naively one might conclude then that we choose m to be the word size. That is for a 10-bit computer we let $m = 2^{10}$. However this is a bad choice, to show why lets consider a sequence with $m = 2^{10}$, $a = 33$, $c = 0$, $x_0 = 102$. Then a part of the sequence looks as follows,

$$102, 294, 486, 678, 870, 38, 230, 422, 614, 806. \tag{2.4}$$

It might not be immediately evident, but upon closer inspection one may notice that all these numbers are even. It could also be possible that the sequence would be all odd or that it alternates between odd and even. This same problem would happen for a 64-bit computer as well and letting $m = 2^{64}$. To explain what exactly is happening here, we introduce a lemma,

Lemma 2.3.1. *If we have X_n belonging to a linear congruential sequence , with multiplier a, increment c and modulus m, with m being a multiple of d. If we then have,*

$$Y_n = X_n \mod d,$$

then it follows that,

$$Y_{n+1} = aY_n + c \mod d.$$

[7, pp. 15-24]

Proof. Let q be some integer and $m = pd$ then we have,

$$X_{n+1} = aX_n + c \mod m \implies X_{n+1} = aX_n + c - qm = aX_n + c - q(pd)$$
$$\implies X_{n+1} \mod d = aX_n + c - q(pd) \mod d \implies Y_{n+1} = aY_n + c \mod d$$

□

So the problem above happened because m is a multiple of $d = 2$. Today computers can quite easily go beyond the 2^{64} and we may choose any large number. Lemma 2.3.1 above guide us to choosing any large prime number and completely avoid this trouble. We will when choosing the other parameters find that this comes with more advantages, but also some limiting issues.

2.4 Lehmer random number generator

We continue by studying the special case when $c = 0$, that is sequence (2.2),

$$X_{n+1} = aX_n \mod m.$$

We saw earlier that m sets the upper limit for the period. A period of m means that each integer between 0 and $m - 1$ appears once, but if $X_n = 0$ then all following numbers will also be 0. So we can't reach a period of m, however we can still get a satisfactory period for some m, according to a theorem which we soon will introduce, but first we will prove a lemma.

Lemma 2.4.1. *Let the decomposition of m into prime factors be,*

$$m = p_1^{e_1} ... p_t^{e_t}.$$

The length, λ, of the period of the linear congruential sequence determined by X_0, a, c, m is the lowest common multiple of the periods λ_j of the linear congruential sequences determined by $X_0 \mod p_j^{e_j}, a \mod p_j^{e_j}, c \mod p_j^{e_j}, p_j^{e_j}$ [7, pp. 15-24]

Proof. We prove this by induction on t. When $t = 1$ the lemma is trivial, since it states that the length of the period of a LCS is the same as an identical LCS. The case when $t = 2$ means we have $m = p_1^{e_1} p_2^{e_2} = m_1 m_2$. If we have linear congruential sequences defined by, $a \mod m_1, c \mod m_1, m_1, Y_0 = X_0 \mod (m_1)$ and $a \mod m_2, c \mod m_2, m_2, Z_0 = X_0 \mod (m_2)$, then according to lemma 2.3.1 we have that

$$Y_n = X_n \mod m_1 \textit{ and } Z_n = X_n \mod m_2, \textit{ for all } n. \tag{2.5}$$

According to a law in[6], since m_1 and m_2 are coprime then $a \equiv b \mod m_1 m_2$ iff $a \equiv b \mod m_1$ and $a \equiv b \mod m_2$. That is,

$$\begin{aligned} X_n \equiv X_k \mod m_1 m_2 \textit{ if and only if} \\ Y_n \equiv X_k \mod m_1 m_2 \textit{ and} \\ Z_n \equiv X_k \mod m_1 m_2. \end{aligned} \tag{2.6}$$

So according to equation 2.5 we get,

$$X_n = X_k \textit{ if and only if } Y_n = Y_k \textit{ and } Z_n = Z_k. \tag{2.7}$$

Let λ' be the least common multiple of the period lenghts λ_1 and λ_2. Then we have $Y_n = Y_{n+\lambda'}$ and, $Z_n = Z_{n+\lambda'}$, hence by equation 2.7 we have $X_n = X_{n+\lambda'}$. Hence we know λ' is a multiple of the period λ and therefore $\lambda \leq \lambda'$. By definition of the period we have $X_n = X_{n+\lambda}$, again by equation 2.7 we get $Y_n = Y_{n+\lambda}$ and, $Z_n = Z_{n+\lambda}$. Thus λ is a multiple of both λ_1, λ_2, thus $\lambda \geq \lambda'$. Thus we have that $\lambda = \lambda'$.
When $t = n + 1$, where the induction hypothesis is that it holds for $t = n$, we set $m_1 = p_1^{e_t}$, $m_2 = p_2^{e_1} ... p_t^{e_n}$. Here, m_2 has period $\lambda_2 = LCM(p_2^{e_1} ... p_t^{e_n})$ by the induction hypothesis. Now showing that $\lambda = LCM(\lambda_1, \lambda_2)$, is identical to what we did for $t = 2$. Thus, the proof is complete. □

Theorem 2.4.2. *The maximum period possible fora linear congruential sequence when $c = 0$ is $\lambda(m)$, defined as*

$$\lambda(m) = \begin{cases} \lambda(2) = 1 & \lambda(4) = 2 \\ \lambda(2^e) = 2^{e-2} \textit{ if } e \geq 3 \\ \lambda(p^e) = p^{e-1}(p-1) \textit{ if } p > 2 \\ \lambda(p_1^{e_1} ... p_t^{e_t}) = lcm(\lambda(p_1^{e_1}), ..., \lambda(p_t^{e_t})). \end{cases}$$

This period is achieved if

i) X_0 is relatively prime to m;

ii) a is a primitive element modulo m.[7, pp. 15-24][3]

Proof. Since $\lambda(p_1^{e_1}...p_t^{e_t} = lcm(\lambda(p_1^{e_1}), ..., \lambda(p_t^{e_t})$ and by lemma 2.4.1, we can see that we only need to consider the case when $m = p^e$. Thus we have the sequence $X_{n+1} = aX_n \mod p^e$ and according to lemma 2.2.2 we have

$$X_n = a^n X_0 \mod p^e.$$

If a is a multiple of p, ie. $a = qp$, then when $n = e$ we get $X_e = q^e p^e X_0 \mod p^e = 0$. So we need a to be relatively prime p to not converge to a sequence of only 0. If we let p^f be the greatest common divisor of X_0 and p^e, and $X_0 = qp^f$ we get,

$$X_0 = a^\lambda X_0 \mod p^e \iff qp^f = a^\lambda qp^f \mod p^f p^{e-f} \iff 1 = a^\lambda \mod p^{e-f}.$$

Since a is relatively prime to p Euler's theorem holds, and thus we find λ is a divisor of $\varphi(p^{e-f})$, where $\varphi(m)$ is Euler's totient function.

$$\varphi(m) = \begin{cases} \varphi(2) = 1 & \varphi(4) = 2 \\ \frac{1}{2}\varphi(2^e) = 2^{e-2} \text{ if } e \geq 3 & \\ \lambda(p^e) = \varphi(p^e) = p^{e-1}(p-1) \text{ if } p > 2 & \end{cases},$$

We note that $\lambda(m)$ is a divisor of $\varphi(m)$,

$$\lambda(m) = \begin{cases} \lambda(2) = \varphi(2) = 1 & \lambda(4) = \varphi(4) = 2 \\ \lambda(2^e) = \frac{1}{2}\varphi(2^e) = 2^{e-2} \text{ if } e \geq 3 & \\ \lambda(p^e) = \varphi(p^e) = p^{e-1}(p-1) \text{ if } p > 2 & \end{cases},$$

and , it is the period.[3]

□

In particular we note that if we choose m to be a large prime, $X_0 \neq 0$ is relatively prime, and so is every a, thus by Fermat's little theorem $a^{p-1} \equiv 1 \mod p$, a is a primitive element.

2.5 Choice of a and c

In a previous section we saw that m sets the upper limit for the period, but the sequence 2.4 clearly did not have a a maximum period since it only had even values so at most it had a period of $m/2$. So clearly we also need to make smart choices with the other parameters. We may ask our self if achieving this maximum period is achievable. The sequence

$$X_{n+1} = X_n + 1 \mod m, X_0 = 0 \tag{2.8}$$

is an linear congruential sequence, and clearly every value $[0, m]$ appears, so it has the maximum period. However this sequence is simply

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

but if we want to use the sequence as random numbers this is clearly a very poor choice. So we need consider more than simply getting the maximum period. We will prove a theorem which lets us find other linear congruential sequences with period length m. The proof for this theorem requires a few lemmas, so lets introduce and prove those first.

Lemma 2.5.1. *Let p be a prime number, and e a positive integer such that $p^e > 2$. Then if*

$$x \equiv 1 \mod p^e \text{ and } x \not\equiv 1 \mod p^{e+1}$$

then

$$x^p \equiv 1 \mod p^{e+1} \text{ and } x^p \not\equiv 1 \mod p^{e+2}$$

[7, pp. 15-24]

Proof. By the first condition we have $x = 1 + qp^e$, where q is not a multiple of p, since then the 2nd condition wouldn't be met. Using the binomial expansion we get

$$\begin{aligned} x^p = (1 + qp^e)^p = \sum_{n=0}^{p} \binom{p}{n} (1^{p-n} q^n p^{ne}) = 1 + \sum_{n=1}^{p} \binom{p}{n} (q^n p^{ne}) \\ 1 + qp^{e+1}(1 + \sum_{n=2}^{p} \binom{p}{n} (q^{n-1} p^{e(n-1)-1})). \end{aligned} \tag{2.9}$$

Its clear that 2.9 is congruent to $1 \mod p^{e+1}$. We now need to show that its not congruent to $1 \mod p^{e+2}$. We start by noticing that every term in the summation is divisible by p, this is clear for every $n > 2$ as then we have an exponent greater than 1. For $n = 2$ this is not the case when $e = 1$, however then $p \neq 2$ by the condition that $p^e > 2$ and thus odd, and as such $\binom{p}{2} = \frac{p(p-1)}{2} = p\frac{p-1}{2}$ is a multiple of p. It follows that since every term in the sum is divisible by p the whole sum is divisible by p. So when multiplied by qp^{e+1} it is a multiple of qp^{e+2} and vanishes under modulus pe^{e+2} leaving behind,

$$x^p \equiv 1 + qp^{e+1} \mod p^{e+2}. \tag{2.10}$$

Since q is not a multiple of p, this is not congruent to 1. □

Corollary 2.5.2. *Let p be a prime number, and e a positive integer such that $p^e > 2$. Then if*

$$x \equiv 1 \mod p^e$$

then

$$x^p \equiv 1 \mod p^{e+1}$$

Proof. The proof is identical to to that of lemma 2.5.1 except we don't have that q is not a multiple of p and hence equation 2.10 may be congruent to 1. But we still have that equation 2.9 is congruent to $1 \mod p^e$ $\square$

Proposition 2.5.3. *If $a \not\equiv 1 \mod p$, then $\frac{a^n-1}{a-1} \equiv 0 \mod p^e$ if and only if $a^n - 1 \equiv 0 \mod p^e$[7, pp. 15-24]*

Proof. We start by noting that $a \not\equiv 1 \mod p$, means that $a-1$ is coprime to p^e so we are justified to cancel the division. First we show the implication,

$$\begin{aligned} \frac{a^n-1}{a-1} \equiv 0 \mod p^e \implies \frac{a^n-1}{a-1} = qp^e \implies a^n - 1 = (a-1)qp^e \\ \implies a^n - 1 \equiv 0 \mod p^e. \end{aligned} \tag{2.11}$$

For the implication the other way we assume that $a^n - 1 \equiv 0 \mod p^e$, and get

$$\frac{a^n-1}{a-1} \equiv t \mod p^e \implies a^n - 1 \equiv t(a-1) \mod p^e \implies t(a-1) \equiv 0 \mod p^e, \tag{2.12}$$

the last equivalence can hold only if $a \equiv 1 \mod p^e$, or if $t \equiv 0 \mod p^e$. But the first case cannot be true, since $a \not\equiv 1 \mod p$. $\square$

Lemma 2.5.4. *Assume that $1 < a < p^e$, where p is prime. If λ is the smallest positive integer for which $(a^\lambda - 1)/(a-1) \equiv 0 \mod p^e$, then*

$$\lambda = p^e \text{ if and only if } \begin{cases} a \equiv 1 \mod p \text{ when } p > 2 \\ a \equiv 1 \mod 4 \text{ when } p = 2 \end{cases} \tag{2.13}$$

[7, pp. 15-24]

Proof. We first show that $\lambda = p^e \implies a \equiv 1 \mod p$ by contradiction. We assume that $\lambda = p^e$ and $a \not\equiv 1 \mod p$. The definition of λ and proposition 2.5.3 then implies that $a^{p^e} - 1 \equiv 0 \mod p^e$. From which we get that $a^{p^e} \equiv 1 \mod p$.

However by Fermats little theorem $a^p \equiv a \mod p$, we get that $a^{p^e} \equiv a \mod p$, which is a contradiction.

Lets also show that $a \equiv 1 \mod 4$ when $p = 2$, again we do this by contradiction. We already know that $a \equiv 1 \mod 2$ by the previous argument, so lets make the assumption that $a \equiv 3 \mod 4$. We have,

$$\begin{aligned} a \equiv 3 \mod 4 \iff a = 3 + 4q \implies a^2 = 9 + 24q + 16q^2 = \\ 1 + 8(1 + 3q + 2q^2) \implies a \equiv 1 \mod 8, \end{aligned} \tag{2.14}$$

for some integer q. Thus we have

$$a^2 \equiv 1 \mod 2^3.$$

Now we can use lemma 2.5.1 repeatedly and get that

$$a^{2^{e-2}} \equiv 1 \mod 2^e. \tag{2.15}$$

Further if we let q be an integer and denote $a^{2^{e-2}} = x$, we have

$$\begin{aligned} x \equiv 1 \mod 2^e \equiv x = 1 + q2^e \implies x^2 = 1 + q2^{e+1} + q^2 2^{e+1} \\ \implies x^2 = a^{2^{e-1}} \equiv 1 \mod 2^{e+1} \implies a^{2^{e-1}} - 1 \equiv 0 \mod 2^{e+1}. \end{aligned} \tag{2.16}$$

We note that,

$$a \equiv 3 \mod 4 \implies a - 1 \equiv 2 \mod 4 \implies a - 1 = 2 + 4q = 2(1 + 2q) \tag{2.17}$$

Note that $1 + 2q$ is coprime 2^e. Thus by equations 2.16 and 2.17, we have that for some integer t,

$$\begin{aligned} a^{2^{e-1}} - 1 \equiv 0 \mod 2^{e+1} \implies a^{2^{e-1}} - 1 = t2^{e+1} \implies \frac{a^{2^{e-1}} - 1}{2} = t2^e \\ \implies \frac{a^{2^{e-1}} - 1}{2} \equiv 0 \mod 2^e \implies \frac{a^{2^{e-1}} - 1}{a - 1} \equiv 0 \mod 2^e \end{aligned} \tag{2.18}$$

But then we have that $\lambda \neq p^e$ since there exists a smaller number satisfying $(a^\lambda - 1)/(a - 1) \equiv 0 \mod p^e$. Thus the condition is necessary. If we have $a \equiv 1 \mod p$ then there exists some value f such that,

$$a \equiv 1 \mod p^f,\ a \not\equiv 1 \mod p^{f+1}. \tag{2.19}$$

since we have $a = 1 + qp$ for some integer $q \neq 0$ since $a > 1$.

By 2.19 and $a \equiv 1 \mod 4$ we can use lemma 2.5.1 g times to get,

$$a^{p^g} \equiv 1 \mod p^{f+g}, a^{p^g} \not\equiv 1 \mod p^{f+g+1}. \tag{2.20}$$

So by equation 2.19, and 2.20 we have $a^{p^g} = 1 + qp^{f+g}$, and $a = 1 + tp^f$, where q, t are not multiples of p. Giving

$$\frac{a^{p^g} - 1}{a - 1} = \frac{qp^{g+f}}{tp^f} = \frac{qp^g}{t} \equiv q^p \equiv 0 \mod p^g, \tag{2.21}$$

where we could cancel, $\frac{t}{q}$ since it's not multiples of p thus we also have $\frac{a^{p^g}-1}{a-1} \not\equiv 0 \mod p^{g+1}$. In particular by letting $g = e$ we see that we have $\frac{a^{p^e}-1}{a-1} \equiv 0 \mod p^e$, and any smaller integer satisfying the condition would be a divisor of p^e. But by letting $g = e - 1$ we see, $\frac{a^{p^{e-1}}-1}{a-1} \not\equiv 0 \mod p^e$, thus p^e is the smallest positive integer satisfying the modulus, that is $\lambda = p^e$. □

Now we are equipped with the lemmas to introduce and prove the theorem

Theorem 2.5.5. *The linear congruential sequence defined by m, a, c and X_0 has a period length m if and only if,*
i) c is relatively prime to m.
ii) $a - 1$ is a multiple of p, for every prime p dividing m.
iii) $a - 1$ is a multiple of 4 if m is a multiple of 4.[7, pp. 15-24]

Proof. We assume $\lambda = m$, then according to lemma 2.4.1 we have that,

$$m = p_1^{e_1} ... p_t^{e_t} = \lambda = lcm(\lambda_1, ..., \lambda_t) \leq \lambda_1 ... \lambda_t \leq p_1^{e_1} ... p_t^{e_t}$$

where the equalities holds if and only if $\lambda_j = p_j^{e_j}$ for all j. Therefore we can prove the theorem for $m = p^e$ without loss of generality.
Case1 :$a = 1$
Condition $ii)$ and $iii)$ are always met, thus we only need to show condition $i)$. If we have period m then x_n must be able to take any value $0, ..., m$ thus we can assume $X_0 = 0$. Let $d \neq 1$ be such that $m = ld$ and $c = kd$ then we get,

$$x_n = nkd \mod ld \implies x_n = nkd + qld.$$

But then X_n is always a multiple of d and as such we can't have a period m. If we have c coprime m, we get,

$$X_n = nc \mod m \iff X_n = nc + qm,$$

Bézout's identity claims that $ax + by = 1$ has solutions if and only if x and y are relatively prime. That is in our case, since c and m are relatively prime, we have that

$$X_n = 1 = nc + qm.$$

By multiplying n and q by any number, t we can then get $X_n = t$, so X_n can take any value.
Case2 :$a \neq 1$
Again we can use $X_0 = 0$ without loss of generality. So by lemma 2.2.2 we get,

$$X_n = \frac{a^n - 1}{a - 1}c \mod m. \tag{2.22}$$

Again Bézout's identity claims that $ax + by = 1$ has solutions if and only if x and y are relatively prime. That is in our case,

$$X_n = 1 = \frac{a^n - 1}{a - 1}c + bm,$$

has solutions if and only if c and m are relatively prime. If 1 doesn't appear in the sequence, then the period isn't m. So condition $i)$ is implied by the claim. Now lets assume we have condition $i)$. Again assuming $X_0 = 0$, we have that the period $\lambda = m$ if and only if its the smallest value such that $X_\lambda = 0$. Since c is relatively prime to m, equation 2.22, gives,

$$\frac{a^n - 1}{a - 1} \equiv 0 \mod m. \tag{2.23}$$

Which by lemma 2.5.4 holds if and only if condition $ii)$ and $iii)$. This completes the proof. □

We learn a thing from this theorem. We need m to have a prime factorization $m = p_1^{e_1}...p_t^{e_t}$, where at least one $e_n > 1, 1 \leq n \leq t$, since otherwise the only number which is a multiple of all p_n is m itself and thus $a = 1$ is the only multiplier which produces the maximum period length of m.

2.6 Potency

We finish the discussion about the linear congruenital sequence by analyzing how well it works to produce random numbers. We do this by introducing the concept of potency. We start by mentioning that potency will be used to reject poor generators but it will not suffice as proof that our random number generator is good. First we want to introduce $b = a - 1$ as this will simplify some notation.

Definition 2.6.1. *The potency of a linear congruential sequence with maximum period is the smallest integer s such that,*

$$(b)^s \equiv 0 \mod m.$$

[7, pp. 15-24]

We note that such an s always exists as theorem 2.1 implies that b is a multiple of each prime dividing m. Thus if $m = p_1^{e_1}...p_t^{e_t}$, then $s = e_1...e_t$ satisfies the condition. As the definition of potency demands the maximum period we can use $X_0 = 0$, when analyzing what its meaning is. Thus for a linear congruential sequence with potency s we get that

$$X_n = \frac{(b+1)^n - 1}{b} c \mod m,$$

which we can expand into

$$X_n \equiv \frac{c}{b}(-1 + \sum_{k=0}^{n} \binom{n}{k} b^k) \equiv \frac{c}{b}(\sum_{k=1}^{n} \binom{n}{k} b^k) \equiv c(\sum_{k=1}^{n} \binom{n}{k} b^{k-1}) \mod m. \quad (2.24)$$

Now we note that since $b^s = 0 \mod m$, every term in the sum with $k > s$ is zero, so we get

$$X_n \equiv c(\sum_{k=1}^{s} \binom{n}{k} b^{k-1}) \mod m. \quad (2.25)$$

We recall the deliberately bad sequence 2.8, which had $a = 1$ for which $b = 0$ and thus the potency is 1, equation (2.25) gives $X_n = cn$. Lets consider a slightly better sequence which has potency 2. Then $X_n = c(n + \frac{n(n-1)}{2}b)$ and $X_{n+1} = c(n + 1 + \frac{n(n+1)}{2}b)$ so the difference between each number in the sequence is,

$$X_{n+1} - X_n = c(1 + bn).$$

Which is a simple relation between one random variable and the next, thus it is not sufficient as a PRNG. In fact according to Donald Knuth, a potency of at least 5 is needed for a sufficiently random behaviour. [7, pp. 15-24]

Chapter 3

Other distributions

3.1 The inversion method

So far we have a way to generate numbers between 0 and $m - 1$, and since each number can only appear once per period, each number is equally probable to appear. If we generate a number X_n using the linear congruential method, then

$$U_n = \frac{X_n}{m-1}$$

is a number between 0 and 1. However we may be interested in other distributions. One method to do so is the inversion method. The idea is simple enough, we start by generating a number U_n from a uniform distribution $U_n \in Un(0,1)$ and then we plug this into the inverse of the wanted distributions cumulative distribution function(CDF)[7, pp. 102-103]. As an example lets say we want to sample from the Cauchy distribution $X_0 = 0$, $\gamma = 1$[1]. This has CDF

$$f(x) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}, \tag{3.1}$$

and inverse

$$f^{-1}(x) = \tan(\pi x - \frac{\pi}{2}). \tag{3.2}$$

When using the LCG with $a = 5776, c = 28561, m = 33078375, x_0 = 0$, to generate 2000 random numbers, we get the histogram in figure 3.1.

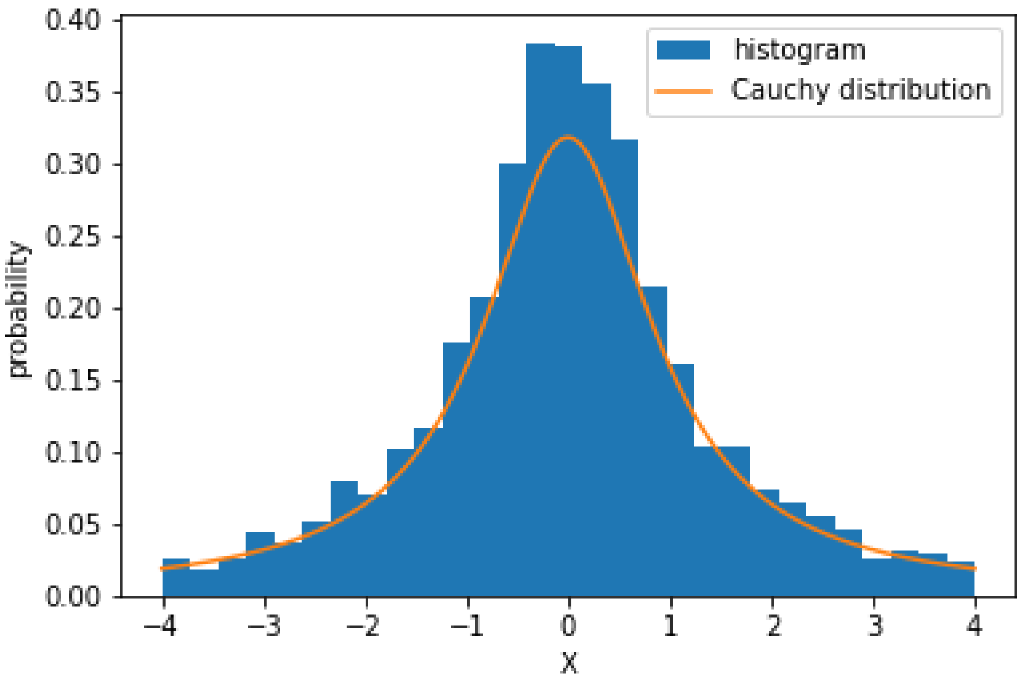


Figure 3.1: Inversion method to generate 2000 Cauchy distributed random numbers $X_0 = 0$,$\gamma = 1$, plotted as a histogram.

This method however has a problem, not all distributions have a CDF and thus we can't use this to get any distribution. In particular the normal distribution has no analytical CDF and thus we can't use the inversion method to sample from it. Thus in the next chapter we will discuss another method.

3.2 The rejection method

As previously mentioned, we can't always use the inversion method since it requires a CDF. The idea behind the rejection method is to first generate a random number from a distribution $h(x)$ such that $\alpha f(x) \leq h(x)$ for all x, with some number α, and PDF of the wanted distribution $f(x)$. In general $h(x)$ can have any distribution which one knows how to generate if it satisfies the mentioned condition. In our case we start with $h(x)$ being the uniform distribution and there is always a valid α, since the distribution is never 0. Once $h(x)$ is chosen we generate a number X from its distribution. Now we want to either accept or reject X. To decide what we do, we generate a new random number V from the uniform distribution $(0, h(X))$. In our case since h is the uniform distribution, we get $V = Un(0, 1)$. Now if $V \leq \alpha f(X)$, we accept X as a number with density f, and otherwise we reject it and repeat the process.[9] To try the method, we use the LCG with $a = 25, c = 1, m = 972, x_0 = 0$), and try to sample from the $beta(2,3)$ density $f(x) = 12x(1-x)^2$[9]. The result, which is rather poor, can be seen in figure 3.2.

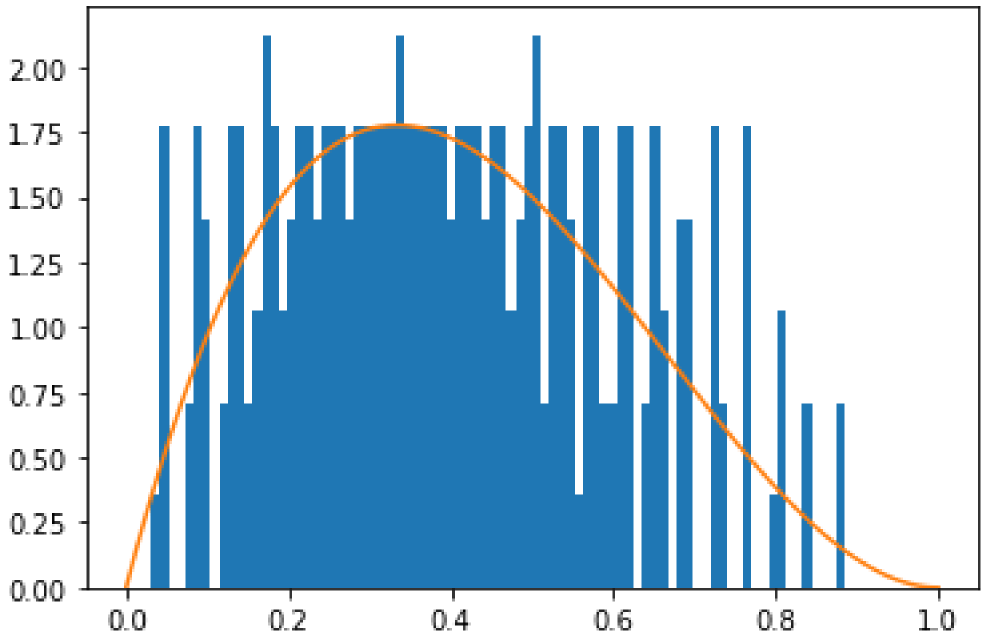
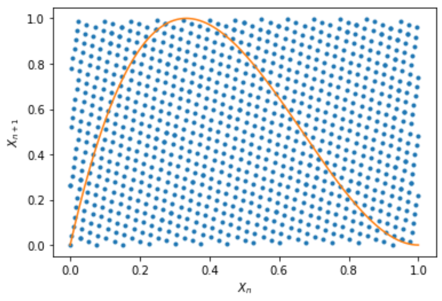


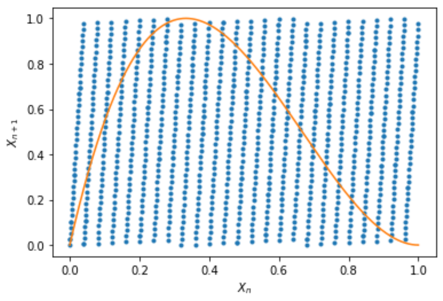
Figure 3.2: Histogram of the accepted points using the rejection method of a beta(2,3) distribution and the LCG (1024,33,1).

The issue is illustrated in figure 3.3. In the figure the blue points represents two consecutive random numbers from an LCG that is X and V. With the orange function being $\alpha f(X)$, so each point below the orange function is an accepted number. As can be seen the points X, V forms a lattice, the shape of which depends on the choice of the multiplier. Note that in figure 3.3b the lattice is such that there are regions where many generated numbers are accepted followed by regions where none are accepted. This issue will occur anytime we choose a dominating function such that $\frac{\alpha f(x)}{h(x)}$ is small compared to 1. But the issue is amplified since a is approximately $\sqrt{m}$ which makes the rows almost parallel to the y-axis.

The areas where $\frac{\alpha f(x)}{h(x)}$ is small can be reduced by choosing $h(x)$ to be distribution other than the uniform. If for example we want the normal distribution, we can choose $h(x)$ to be Cauchy distributed which we get by using the inversion method.



(a) All the pairs of consecutive random numbers produced by the LCG with $a = 253$,$c = 1$, and $m = 972$.



(b) All the pairs of consecutive random numbers produced by the LCG with $a = 25$,$c = 1$, and $m = 972$.

Figure 3.3: Random numbers generated by two different LCGs, and the beta distribution in orange. Every point below the orange function are accepted.

3.3 Testing

In this section we will analyse how good our linear congruential generator combined with the rejection method is, and try to replicate the results of W. Hörmann and G. Derflinger.[9] In their paper they introduced the discrepancy of a sequence as,

$$D_N^1(F) := \sup_{s\leq t}|\frac{\#\{y_l : s < y_l \leq t, i = 1, ..., N\}}{N} - (F(t) - F(s))| \tag{3.3}$$

where y_l is a sequence of all the accepted random numbers, N is the total number of accepted numbers over the entirety of the domain, and F is the distribution function. The $F(t) - F(s)$ is derived from,

$$\int_s^t f(x)dx = F(t) - F(s) \tag{3.4}$$

where $f(x)$ is the probability density function. That is, the correct probability that a random number should be in the region $(s, t]$. The fraction part is the probability that our random sequence lands in $(s, t]$. Thus the difference between them is a measure on how wrong our sequence is in the region $(s, t]$. By then taking the supremum of the unsigned value we get the measure on how wrong our sequence is, in the worst possible case. However, it is difficult to numerically calculate this value so instead we have calculated the largest discrepancy over 10^6 subintervals with $m/10^6$, generated numbers X each. Our first step is calculating N. This is simply estimated to be αm, since both $f(x)$ and $h(x)$ are probability functions the area under their graphs are 1 thus, the area under $\alpha f(x)$ is α. Next we need the cumulative distribution function of the beta(2,3) distribution,

$$F_\beta^{2,3}(x) = 1 - (1 - x)^3 - 3x(1 - x)^3. \tag{3.5}$$

	Anton Sjödin				W. Hörmann and G. Derflinger			
	χ^2 statistic		$m\cdot$ discrepancy		χ^2 statistic		$m\cdot$ discrepancy	
multiplier	normal	beta	normal	beta	normal	beta	normal	beta
742938000	114567	100050	6.86	8.4	1000020	99404	194	164.27
95076400	92312.6	99977	8.18	8.6	100195	100071	189	233.14
630360000	108382	99865	8.1	9.4	99529	99495	202	148.34
39373	217150	293290	3398	3655	206056	367132	48582	56799
16807	306143	496821	3478	3752	144163	214154	20789	24297
48271	209535	241573	3367	3628	131314	177831	16937	20077
69621	171119	149834	3286	3550	113625	131067	11734	13722

Table 3.1: Theχ^2 statistic and discrepancies for Lehmer generators using the modulus $2^{31}-1$ and various multiplier as seen in the first column. The large difference between my results and W. Hörmann and G. Derflinger in discrepancy is due to poor documentation of where they calculated discrepancy.

Further we also generated 10^6 random numbers, and calculated the χ^2 value,

$$\chi^2 = \sum_{k=0}^{10^5} \frac{(O_i - E_i)^2}{E_i}, \tag{3.6}$$

where O_i is how many numbers we observed in one of 10^5 sub intervals, and E_i is how many we were expecting[2]. Using $E_i = np_i$, we can estimate each subinterval as a uniform region with uniform probability, where the probability function is $P = f(i + 0.5)$ and thus,

$$p_i = P * 10^{-5} \implies E_i = 10P = 10f(i + 0.5) \tag{3.7}$$

The results can be seen in table 3.1.

Although our discrepancy values doesn't agree with that of W. Hörmann and G. Derflinger, we do note that an LCG with an multiplier which is of the order of $\sqrt{m}$ produces random variables with relatively high discrepancy. As for the χ^2 statistic we note that when we have a large multiplier the statistic is of the order of 10^5 meaning that it is a good fit. Thus despite not getting identical values, we can still support their conclusion. When combining the LCG with the rejection method, one wants to use multipliers which are much larger than $\sqrt{m}$.

Chapter 4

Summary

We went out with the goal to create a random number generator of any distribution. In doing so we started by defining the linear congruential generator, as a method to get uniformly distributed numbers,

$$X_{n+1} = aX_n + c \mod m.$$

Thus we discussed a lot of theory on how too choose a, c, m and X_0, to get a hard to predict sequence with a long period. We noted that m should be a large number with a prime factorization where the primes had to be raised to some power. This was necessary since when choosing a, we found that $b = a - 1$ by theorem 2.5.5 had to be a multiple of every prime factor of m, thus if m only had unique factors the only valid a would be 1. We also discussed in the chapter on potency how we can't just take any a that achieves maximum period, we wanted an a such that,

$$b^4 \neq 0 \mod m.$$

As for the increment c we found that it only needs to be relatively prime with m, in particular this means that $c = 0$ could never achieve the maximum period. Nonetheless we discussed the case when $c = 0$ and m is a prime, and found that such a generator has a period of $m - 1$ which is satisfactory.

We then went on to discuss the inversion method and rejection method, so that we could transform our uniformly distributed numbers and get numbers with any distribution. The inversion method was limited to distribution which had a cumulative distribution function. The rejection method however can give any distribution that we can give a distribution function for. We combined the rejection and inversion method as doing so gives better results[9]. We then went ahead and tried to replicate the results of W. Hörmann and G. Derflinger.

Appendix A

1

Below is the python program used to get the values in table 3.1 can be seen. The program also has some other functions related to the theory of linear congruential generators, such as finding the potency. The code is largely written in a straight forward naive way. In the function max_period a loop which repeatedly divides odd numbers to find shared prime factors between b and the modulus is used[5].

Listing A.1: Python code

```
import numpy as np
import math as ma


#%%

class Linear_congruential_method:
    def __init__(self,a,c,m): #initialize
        self.multiplier=a
        self.increment=c
        self.modulus=m
    def Next_in_sequence(self,X_0): #definition of the sequence
        X_1=self.multiplier*X_0+self.increment
        return X_1%self.modulus
    def N_th_in_sequence(self,X_0,n):
        a=self.multiplier
        mod=self.modulus
        c=self.increment
        a_to_n=a**n #calculate this only once
        term2=((a_to_n-1)*c//(a-1))
        X_n=(a_to_n*X_0+term2)
        return(X_n%mod)
    def zero_to_n(self,X_0,n): #get n numbers from the sequence
        sequence=[]
        for i in range (n):
            print (i)
            sequence.append(X_0)
            X_0=self.Next_in_sequence(X_0)
        return(np.array(sequence))
    def max_period(self,Print=True,Factors_of_m=False):
        b=self.multiplier-1
        c=self.increment
        m=self.modulus
        if ma.gcd(c,m)!=1:
            if Print:
                print('c, and m need to be relatively prime')
            return(False)

        if ((m%2==0) and not(b%2==0)):
            if Print:
                print('m is divisible by 2, but a-1 is not')
            return(False)
        if ((m%4==0) and not(b%4==0)):
            if Print:
                print('m is divisible by 4, but a-1 is not')
            return(False)
        while m%2==0:
            m//=2
        while b%2==0:
            b//=2
        i=3
        while i<=ma.ceil(np.sqrt(m)):
            if m==1:
                break
            if b==1:
                if Print:
                    print('''m={} has atleast 1 prime divisor,
                        which doesn\'t divide a-1={}'''.format(m,b))
                return(False)
            if (m%i==0) and not(b%i==0):
                if Print:
                    print('m is divisible by {},
                    but a-1={} is not'.format(i,b))
                return(False)
            while m%i==0: #removes all factors i from m
                m//=i
            while b%i==0:
                b//=i
            i+=2
        if m!=1 and m!=b:
            if Print:
                print('m is divisible by {} but a-1 is not'.format(m))
            return False
        return True
    def potency(self):
        if not self.max_period(False):
            print('The sequence needs maximum
                    period for a potency to exist')
            return(False)
        b=self.multiplier-1
        m=self.modulus
        i=1
        while True:
            i+=1
            if b**i%m==0:
                return i
    def rejection_method_beta(self,alpha,x_0,n):
        m=self.modulus
        N=0
        K=m**2
        List=[]
        while N<=n:
            V=self.Next_in_sequence(x_0)

            if V*m**2 <= alpha*(12*x_0*(K+x_0*(x_0-2*m))):
                N+=1
                List.append(V)
            x_0=self.Next_in_sequence(x_0)
        return(List)
    def rejection_methodod_cauchy(self,alpha,x_0,n):
        m=self.modulus
        List=[]
        N=0
        divide=np.sqrt(2*np.pi)
        while N<=n:
            V=self.Next_in_sequence(x_0)/m
            X=x_0/m
            X=np.tan(np.pi*(X-0.5))
            V*=1/(np.pi*(1+X**2))

            if V <= alpha/(divide)*np.exp(-0.5*X**2):
                List.append(X)
                N+=1
            x_0=self.Next_in_sequence(x_0)
        return(List)
    def discrepancy(self,alpha,areas):
        m=self.modulus
        N=m*alpha
        numbers_per_area=m//areas+1
        width=1/areas
        worst=0
        for i in range(areas):
            accepted=0
            for X in range(i*numbers_per_area,(i+1)*numbers_per_area):
                V=self.Next_in_sequence(X)/m
                X=X/m
                if V <= alpha*(12*X*(1+X*(X-2))):
                    accepted+=1
            low=i*width
            high=(i+1)*width
            I_x=(high**2*(6*-high*(8-3*high))
                 -low**2*(6-low*(8-3*low)))
            discrepancy=accepted/N-I_x
            worst=max([discrepancy,worst],key=abs)
        return(m*worst)
    def discrepancy_cauchy(self,alpha,areas):
        m=self.modulus
        N=m*alpha
        numbers_per_area=int(np.ceil(m/areas))
        worst=0
        left=-10**9
        divide=np.sqrt(2*np.pi)
        for i in range(areas):
            accepted=0
            for X in range(i*numbers_per_area,(i+1)*numbers_per_area):
                V=self.Next_in_sequence(X)/m
                X=X/m
                X=np.tan(np.pi*(X-0.5))
                V*=1/(np.pi*(1+X**2))
                RHS=1/divide*np.exp(-0.5*X**2)
                if V <= RHS*alpha:
                    accepted+=1
            right=X
            width=(right-left)
            left=right
            I_x=RHS*width
            discrepancy=accepted/N-I_x
            worst=max([discrepancy,worst],key=abs)
            print(accepted/N,I_x,worst)
        return(m*worst)


#%%

def chi_square(LCM,alpha,multipliers):
    m=LCM.modulus
    O=0
    chilist=[]
    divide=np.sqrt(2*np.pi)
    for a in multipliers:
        chi=0
        LCM.multiplier=a
        List1=np.array(LCM.rejection_methodod_cauchy2(alpha,1,10**6-1
        start=np.min(List1)
        end=np.max(List1)
        stepsize=(end-start)*10**-5
        for x in np.linspace(start,end,10**5):
            A=List1[x<List1]
            A=A[A<=x+stepsize]
            O=len(A)
```

```
                p=(1/divide*np.exp(-0.5*x**2))*stepsize
                E=10**6*p
                print(O,E,chi)
                chi+=(O-E)**2/E
            print(chi)
            chilist.append(chi)
        return(chilist)
#%%
alpha_normal=0.60653
alpha_beta=0.5626
a=[742938285,95076376,630360016,397204094,39373,16807,48271,69621]
c=0
m=2**31-1
LCG_for_rejection=Linear_congruential_method(1,c, m)
normal_disc=[]
beta_disc=[]
for i in a:
    LCG_for_rejection.multiplier=i
    s=LCG_for_rejection.discrepancy(alpha_beta, 10**6)
    t=LCG_for_rejection.discrepancy_cauchy(alpha_normal,10**6)
    normal_disc.append(t)
    beta_disc.append(s)
```

Bibliography

[1] In: *A Primer on Statistical Distributions.* John Wiley Sons, Ltd, 2003. Chap. 12, pp. 119–122. ISBN: 9780471722229. DOI: `https://doi.org/10.1002/0471722227.ch12`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471722227.ch12`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/0471722227.ch12`.

[2] N. Balakrishnan, Vassilly Voinov, and M.S Nikulin. "Chapter 2 - Pearson's Sum and Pearson-Fisher Test". In: *Chi-Squared Goodness of Fit Tests with Applications.* Ed. by N. Balakrishnan, Vassilly Voinov, and M.S Nikulin. Boston: Academic Press, 2013, pp. 11–26. ISBN: 978-0-12-397194-4. DOI: `https://doi.org/10.1016/B978-0-12-397194-4.00002-8`. URL: `https://www.sciencedirect.com/science/article/pii/B9780123971944000028`.

[3] R.D. Carmichael. "Note on a New Number Theory Function". In: *American Mathematical Society* 16 (5 1910), pp. 232–238.

[4] W.A. Coppel. *Number Theory An Introduction to Mathematics 2ed.* Universitext. Springer, 2009, pp. 83–85, 106–107.

[5] Siddharth Jain. *Prime factorization: How to find prime factors of a number in Python.* June 2021. URL: `https://www.pythonpool.com/prime-factorization-python/`.

[6] Donald Ervin Knuth. *The Art of Computer Programming, volume 1.* Addison-Wesley Series in computer science and information processing. Addison Wesley, 1997, p. 40. ISBN: 0201896834.

[7] Donald Ervin Knuth. *The Art of Computer Programming, volume 2.* Addison-Wesley Series in computer science and information processing. Addison Wesley, 1969, pp. 1–6, 15–24, 102–103.

[8] John von Neuman. "Various Techniques Used in Connection With Random Digits". In: *J. Res. Nat. Bur. Stand. Appl. Math* 12 (3 1951), pp. 36–38.

[9] W.Hörmann and G.Derflinger. "A Portable Random Number Generator Well Suited for the Rejection Method". In: *ACM Transactions on Mathematical Software* 19 (4 1993), pp. 489–495.