Linear Congruential Generators

Julian Schennach

December 10, 2023

1 Introduction

Random numbers are frequently used in a variety of subjects: In statistics, it is necessary to obtain random samples, or to perform methods requiring randomness like bootstrapping. We also need random numbers in order to have secure locks in cryptography, and finally, computer simulations often feature random particles at random locations. Unfortunately, there is no way to obtain *truly* random numbers — but there exist methods to create seemingly random numbers.

A pseudo-random number generator is an algorithm that creates a sequence of numbers that appear random and only repeat after a significant amount of time. In this paper, we explore the properties of one particular type of pseudo-random number generator, a linear congruential generator (abbreviated, LCG), by proving a few formal results, such as under what conditions a LCG has a period of m. We will also describe some limitations of this algorithm, including the predictability of the least significant digits or unwanted relations between consecutive terms, and ways to mitigate these issues. Finally, we will implement various examples of LCGs in Python.

2 Definition

A rather simple *pseudo*-random number generator using moduli, known as a linear congruential generator, is defined as follows: Define four positive integers, a "multiplier" a, the "increment" c, the "modulus" m, and the "seed" x_0. Then we define the following recursive sequence:

$$x_n \equiv ax_{n-1} + c \pmod{m}.$$

The resulting x_n terms form a sequence of pseudo-random numbers between 0 and $m-1$, inclusive. As the name suggests, an LCG is *linear* (no exponents), uses *congruents* (i.e. moduli), and *generates* "random" numbers.

Note that a, c, x_0 can be between 0 and $m-1$, inclusive, as otherwise we can reduce them mod m. This also means that all x_i are between 0 and $m-1$. To select the seed x_0, we probably want a random number. Since it is impossible to obtain a truly "random" number, we typically use the current computer time as the seed (mod m, of course), with high temporal precision so that it is highly unlikely that the same seed is ever selected

again. Some other sources exist, including time between key presses, radioactive decay, and, surprisingly enough, lava lamps!

3 Periods

Unfortunately, an LCG cannot generate "random" numbers forever. Since it generates numbers mod m, we can only have at most m numbers generated. If $x_j = x_i$ (i.e. when a number repeats in the LCG), where $j > i$, we know that $x_{j+1} \equiv ax_j + c \equiv ax_i + c \equiv x_{i+1} \pmod{m}$. In other words, whenever a number repeats itself, all future numbers repeat as well. Thus, an upper bound for a period of a LCG is clearly m: There are only m possible remainders mod m, so if we had a period of $m+1$ or more and no repeated values within a cycle, there would actually have to be at least two equal terms by the Pigeonhole Principle — a contradiction. Hence, the period of a LCG must be m or less.

Of course, we'd ideally want a period of m so that it is as difficult as possible to notice when the LCG repeats. Fortunately, there are some simple necessary and sufficient conditions on the LCG that result in a period of m.

Theorem 1. *An LCG has period m if and only if* 1) *c is relatively prime to m,* 2) *$a \equiv 1 \pmod{p}$ for every prime factor p of m, and* 3)*, when $m \equiv 0 \pmod{4}$, $a \equiv 1 \pmod{4}$.*

Remark 1. *The last two conditions imply that $a \equiv 1 \pmod{2\prod_{p|m} p}$ (or $a \equiv 1 \pmod{\prod_{p|m} p}$ if m is not a multiple of 4) by the Chinese Remainder Theorem — and if m has no repeated prime factors other than 2 (which only repeats at most once), we have $a \equiv 1 \pmod{m}$ or just $a = 1$. Of course, this latter case is not ideal, as it would be easy to notice an arithmetic progression in the LCG.*

Let us now proceed with the proof. We separate it into the "if" and "only if" conditions, the former adapted from Hull and Dobell [1], and the latter from Knuth [2]. We also add many clarifications to make the proofs clearer (albeit longer).

Proof. **(Sufficiency)** Suppose that the LCG has period n, and that we have selected a and c that satisfy the conditions of the theorem. We must have $x_{n+i} = x_i$ (where the repetition may occur at any x_i). Thus, we also have:

$$\begin{aligned} x_i = x_{n+i} &\equiv ax_{n+i-1} + c \\ &\equiv a^2 x_{n+i-2} + ac + c \equiv \cdots \\ &\equiv a^n x_i + (a^{n-1} + a^{n-2} + \cdots + a + 1)c \\ &\equiv a^n x_i + \frac{a^n - 1}{a-1} c \pmod{m}. \quad (1) \end{aligned}$$

We rearrange and factorize to get the following:

$$0 \equiv a^n x_i - x_i + \frac{a^n - 1}{a-1} c \pmod{m}$$

$$0 \equiv \frac{a^n - 1}{a-1}(x_i(a-1) + c) \pmod{m}.$$

Assume that the conditions listed in Theorem 1 hold. While $a-1$ does share all prime factors with m, the extra c that is relatively prime to c prevents $x_i(a-1)+c$ from sharing any factors with m. Hence, we must have $\frac{a^n-1}{a-1} \equiv 0 \pmod{m}$. We wish to prove that $n=m$ is the smallest possible n for which this congruence holds. So we must both prove that $n=m$ works, and that $n<m$ do not work. Since $a=1$ is a trivial case (with c relatively prime to m, we simply cycle through all $x_0+kc \pmod{m}$ and have a period of m), we assume that $a>1$.

Note that if $\frac{a^{n'}-1}{a-1} \equiv 0 \pmod{m}$, we also have $\frac{a^{kn'}-1}{a-1} = \frac{(a^{n'}-1)(a^{(k-1)n'}-a^{(k-2)n'}+\cdots)}{a-1} \equiv 0 \pmod{m}$ for any positive integer k. So we must only consider m' that are the highest powers of primes $p_i^{e_i}$ that divide m (as well as their respective n'). When we show that n' is at least and can be $p_i^{e_i}$, we simply take the multiples of each $p_1^{e_1}, p_2^{e_2}, \cdots$ to find that n is at least and can be m.

When $m'=p^e$, where $e>0$ (since the $m'=1$ case is useless), we let $a=kp^f+1$ ($\equiv 1 \pmod{p}$) where $k \not\equiv 0 \pmod{p}$ and $f \geq 1$. Now we expand $\frac{a^{n'}-1}{a-1}$, with $n'=p^e$, to get:

$$\begin{aligned}\frac{(kp^f+1)^{p^e}-1}{kp^f+1-1} &= \frac{(kp^f)^{p^e}+\binom{p^e}{1}(kp^f)^{p^e-1}+\cdots+\binom{p^e}{1}(kp^f)+1-1}{kp^f} \\ &= k(kp^f)^{p^e-1}+k\binom{p^e}{p^e-1}(kp^f)^{p^e-2}+\cdots+k\binom{p^e}{1}.\end{aligned}$$

We want this to be congruent to 0 $\pmod{p^e}$, or, in other words, be divisible by p^e. Fortunately, the $k\binom{p^e}{i}(kp^f)^{i-1}$ terms for $i<p^e$ are divisible by p^e: If some factor $0<j<i$ in the denominator of $\frac{(p^e)(p^e-1)\cdots(p^e-i+1)}{i!}(kp^f)^{i-1}$ is divisible by some maximal p^b where $b \leq e$, then p^e-j is also divisible by p^b and cancels that factor with j. And when i is divisible by some $p^b \leq p^{i-1}$ (as $i \leq p^{i-1}$), its p^b factor can cancel with $(kp^f)^{i-1}=k^{i-1}p^{f(i-1)}$ (because $f \geq 1$). Since the p^e factor of the binomial is not affected, the whole term is divisible by p^e. Thus, the only special term is the $k(kp^f)^{p^e-1}$, but because $p^e \geq e+1$ for $e>0$, $k(kp^f)^{p^e-1}=k^{p^e}p^{f(p^e-1)}$ is in fact divisible by p^e. So the whole sum written above is divisible by p^e, as desired.

So $n'=p^e$ satisfies $\frac{a^n-1}{a-1} \equiv 0 \pmod{p^e}$. Now we must show that no smaller n' works. First, this smaller n' must be a factor of m. Otherwise, for this n', we have $\frac{a^{n'}-1}{a-1} \equiv 0 \pmod{p^e}$, or $a^{n'} \equiv 1 \pmod{p^e(a-1)}$. Let kn' be the smallest multiple of n' that is larger than m. Then $a^{kn'} \equiv 1 \pmod{m'(a-1)}$, just like $a^{p^e} \equiv 1 \pmod{p^e(a-1)}$. If $a^{kn'-p^e} \not\equiv 1 \pmod{p^e(a-1)}$, then when we multiply this by a^{p^e}, we find that $a^{kn'} \not\equiv 1 \pmod{p^e(a-1)}$. So, due to this contradiction, we must have $a^{kn'-p^e} \equiv 1 \pmod{p^e(a-1)}$. Since n' is not a factor of m, $m-kn'$ is smaller than n', so n' is not the smallest solution to $a^n \equiv 1 \pmod{p^e(a-1)}$. Hence, by proof by contradiction, n' must be a factor of $m'=p^e$. Since all multiples of the smallest n' must work too, we will show that $n'=p^{e-1}$ fails (implying that no other smaller n' works).

Continuing, we substitute $n'=p^{e-1}$ into $\frac{a^{n'}-1}{a-1}$, where $a=kp^f+1$ again. We obtain:

$$\frac{(kp^f+1)^{p^{e-1}}-1}{kp^f+1-1} = \frac{(kp^f)^{p^{e-1}} + \binom{p^{e-1}}{1}(kp^f)^{p^{e-1}-1} + \cdots + \binom{p^{e-1}}{1}(kp^f) + 1 - 1}{kp^f}$$
$$= k(kp^f)^{p^{e-1}-1} + k\binom{p^{e-1}}{p^{e-1}-1}(kp^f)^{p^{e-1}-2} + \cdots + k\binom{p^{e-1}}{1}.$$

The final binomial is clearly not divisible by p^e (as k is relatively prime to p). We reuse the reasoning above to prove that $k\binom{p^{e-1}}{i}(kp^f)^{i-1}$ is divisible by p^{e-1}. Then, there is an extra factor of p: i cannot be divisible by p^{i-1} because, for *odd* p, $p^{i-1} > i$, so i must be divisible by at most p^{i-2}. Thus, the $(kp^f)^{i-1}$ that cancels with i's factor has at least one p remaining (as $f \geq 1$, and $(kp^f)^{i-1} = k^{i-1}p^{f(i-1)} \geq k^{i-1}p^{i-1} = p(k^{i-1}p^{i-2})$). So the whole term is divisible by p^{e-1} and an extra p factor, so it is also divisible by p^e. Finally, because $p^{e-1} - 1 \geq e$ for *odd* p, $k(kp^f)^{p^{e-1}-1} = k^{p^{e-1}}p^{f(p^{e-1}-1)}$ has a factor of p^e as well. Since all terms except the lonely $k\binom{p^{e-1}}{1}$ are divisible by p^e, the whole sum is not divisible by p^e, as desired.

However, note that p must be odd so that the reasoning above works. Fortunately, little must be changed for $p = 2$: When $e = 1$, the whole sum is just $k(kp^f)^{p^{1-1}-1} = k$, which is not divisible by p^1. And when $e > 1$, the two statements requiring p's odd parity actually hold when $f > 1$: $(kp^f)^{i-1}$ has a remaining p factor because $(kp^f)^{i-1} = k^{i-1}p^{f(i-1)} \geq k^{i-1}p^{2(i-1)} \geq k^{i-1}p^i$ (as $2(i-1) \geq i$ for positive integer i), and hence its division with the (at most) p^{i-1} factor of i results in an extra factor of p. Similarly, when $f > 1$, $k(kp^f)^{p^{e-1}-1} = k^{p^{e-1}}p^{f(p^{e-1}-1)} \geq k^{p^{e-1}}p^{2(p^{e-1}-1)} \geq k^{p^{e-1}}p^e$ since $2p^{e-1} - 2 \geq e$ (when $e > 1$) for all prime p. So whenever $m = 2^e$ for $e > 1$, we need $a = k2^f + 1$ with $f > 1$, so $a \equiv 1 \pmod 4$.

Having considered all cases, we combine them to show that the conditions in Theorem 1 are indeed sufficient, completing half of the proof. □

Corollary 1. *When c is relatively prime to m, the period of that LCG must be a proper divisor of m.*

Proof. This follows from our proof that n' must divide $m' = p^e$. □

Next, we must prove that the conditions of Theorem 1 are necessary, which will fortunately require a much shorter proof.

Proof. **(Necessity)** Since in a period of m, the LCG must repeat each possible remainder mod m, we can assume that $x_i = 0$. Using equation (1), we want to have $0 \equiv \frac{a^n-1}{a-1}c \pmod m$ only when $n = m$ (i.e. only when the period is m). Note that if c shares a factor with m, $\frac{a^j-1}{a-1}c$ $(= x_{i+j})$ can never equal 1 (as it shares the same factor as c with m), contradicting the fact that we must cycle through all remainders mod m as j increases. Thus, c must indeed be relatively prime to m.

Hence, we can divide by c in $0 \equiv \frac{a^n-1}{a-1}c \pmod m$ to get $\frac{a^n-1}{a-1} \equiv 0 \pmod m$. Using the same reasoning as in the sufficiency proof, we only consider $m' = p^e$. We wish to prove that if n must be p^e, then $a \equiv 1 \pmod p$. Let's prove the contrapositive: Suppose $a \not\equiv 1 \pmod p$. Then, when $\frac{a^n-1}{a-1} \equiv 0 \pmod{p^e}$, we also have $a^n - 1 \equiv 0 \pmod{p^e}$ as $a - 1$ cannot cancel with any potential factor of p in $a^n - 1$. When $n = p^e$ (which must satisfy this congruence),

we have $a^{p^e} \equiv 1 \pmod{p^e} \equiv 1 \pmod{p}$. By Fermat's Little Theorem, $a^p \equiv a \pmod{p}$, so $a^{p^e} \equiv a^{p^{e-1}} \equiv a^{p^{e-2}} \equiv \cdots \equiv a^p \equiv a \pmod{p}$. Thus, a^{p^e} is congruent to both a and 1 mod p, but a is not congruent to 1 $\pmod{p}$, forming a contradiction. Hence, a is actually equivalent to 1 $\pmod{p}$.

However, there is an extra subtlety that occurs when $p = 2$. We do not want $n = 2^{e-1}$ to satisfy $\frac{a^n-1}{a-1} \equiv 0 \pmod{2^e}$. Yet, when $a \equiv 3 \pmod{4}$ (even a would not work regardless), $\frac{a^{2^{e-1}}-1}{a-1} = a^{2^{e-1}-1} + a^{2^{e-1}-2} + \cdots + a + 1 \equiv 3 + 1 + 3 \cdots 3 + 1 \equiv 3(2^{e-2}) + (2^{e-2}) \equiv 0 \pmod{2^e}$. Thus, we cannot have $a \equiv 3 \pmod{4}$, so we require $a \equiv 1 \pmod{4}$ when 2^e has a factor of 4.

Combining these cases again, we conclude that the conditions of Theorem 1 are indeed necessary. □

4 Limitations

While an LCG with a period of m is more difficult to predict, there may still be some issues. If our period is too short, even if it is equal to m, we'd be able to notice the repetition and obtain the entire sequence. So we want m to be sufficiently large, and have a and c satisfy Theorem 1 (so that we have a length-m period).

However, some problems arise in this situation as well. When m is quite large, we can ignore the modulo in the recurrence relation $x_n \equiv ax_{n-1} + c \pmod{m}$ for some smaller n since $ax_{n-1} + c$ is not yet greater than m. So initially $x_n = ax_{n-1} + c$, and also $x_{n-1} = ax_{n-2} + c$. Subtracting one equation from the other, we have $x_n - x_{n-1} = a(x_{n-1} - x_{n-2}$, so $a = \frac{x_n - x_{n-1}}{x_{n-1} - x_{n-2}}$. If we know some smaller terms, we are able to deduce a and then also deduce c (as $c = x_n - ax_{n-1}$)! A solution would be to select a large a or c so that $ax_{n-1} + c$ is immediately greater than m and so that the equivalence does not become an equality.

There may also be noticeable patterns in the units digits of the terms. For instance, suppose that $a = 5$. Then, if m is large enough, we notice that the first few terms of the generated sequence all have the same two units digits (that have a difference of 5). Then, we'd be able to guess that a is a multiple of 5. A solution to this, as well as the previous, problem is that, while we do first generate the terms using an LCG, we *then* remove the first few digits of each term (and if we do not have enough digits in a term, we simply replace the term with 0). Now, significant information is lost, and we can no longer deduce a or c. For sufficiently large m, we still have a large variety of terms in the new sequence. And even if some terms repeat, future terms will not necessarily repeat, unlike in a standard LCG. Informally, this new pseudo-random sequence can be considered more "random" because we can have repeated terms without consequences — a sequence without any repetition may be suspicious.

5 Generalizations

A simple generalization of our recurrence relation is to square the previous term, or, in other words, define the recurrence as $x_n \equiv ax_{n-1}^2 + c \pmod{m}$. This is known as a Blum-Blum-Shub (BBS) random number generator when $a = c = 1$. We can also use an "exponent" b

and obtain a different recurrence, $x_n \equiv ax_{n-1}^b + c \pmod{m}$.

However, these generators are less useful than an LCG because they may repeat more frequently for certain b. For instance, when $b = 2$, we know that x_{n-1}^2 can only be 0 or 1 mod 4. So $ax_{n-1}^2 + c$ can only be certain remainders mod 4, and if m is divisible by 4, we cannot obtain all possible remainders mod m and we can never have a period of length m. If m is *not* divisible by 4, a similar issue occurs: x^2 and $(m-x)^2$ have the same remainders mod m, so we only have $\lceil \frac{m}{2} \rceil$ possible remainders mod m for x_{n-1}^2. So $ax_{n-1}^2 + c$ can only have about half of all remainders mod m, and we cannot have a full length-m period. And since x^b and $(m-x)^b$ are congruent mod m for all even b, the same reasoning applies for all even b. Of course, for odd b, or simply for large m, this generator is still useful. We may not be able to reach a period of length m at all times, but we may still be able to reach a period length of $\lceil \frac{m}{2} \rceil$ (which is still large enough if m is).

Nevertheless, a BBS is considered cryptographically secure, while an LCG is not. This is mostly because we must factor the terms of the BBS to crack the seed and modulus, and factoring is a famously difficult problem, while an LCG, as its name suggests, is linear and does not require factoring. There are different conditions for a BBS to have a maximal period $\lceil \frac{m}{2} \rceil$ (m must be a product of two approximately-equal primes that satisfy certain congruences and gcd equations) but we do not focus on them in this paper.

LCGs and this generalization generate numbers that are remainders mod m. Often, we'd prefer to generate random numbers from the interval $[0, 1]$. We can use an LCG to create a uniform pseudo-random number generator, or, in other words, generate "random" numbers between 0 and 1, inclusive, that are uniformly distributed: We simply divide each term generated by the LCG by $m - 1$. The possible outputs are $\frac{0}{m-1}, \frac{1}{m-1}, \cdots, \frac{m-1}{m-1}$, so our terms are indeed between 0 and 1, inclusive. They are also spaced at equal intervals and are therefore uniformly distributed.

6 Examples

The following is a simple code for an LCG in Python. Note that we must only repeat m times as the period is at most m.

```
a=5 #multiplier
c=7 #increment
m=16 #modulus
x=3 #seed

for i in range(m):
    x=(a*x+c)%m
    print(x)
```

Running the program for the values of a, c, m, x_0 shown, we obtain the sequence $6, 5, 0, 7, 10, 9, 4, 11, 14, 13, 8, 15, 2, 1, 12, 3$, which is indeed a full period of length $m = 16$. If we use $a = 5$, $c = 7$, $m = 18$, and $x_0 = 3$ (a sequence that fails conditions 2 and 3 of Theorem 1), the sequence becomes $4, 9, 16, 15, 10, 3$ and then it repeats. So, as we'd expect from Theorem 1,

the sequence does not have a full period $m = 18$. These examples help validate the theorem, in addition to its (rather long) proof.

Also, notice that the period for this second LCG is 6, which divides $m = 18$, as implied by Corollary 1.

7 Conclusion

Sequences of random numbers have uses in various fields, and LCGs allow for simple and versatile random number generation. Theorem 1 in particular means that LCGs with certain values for a, c, m can have full periods of length m, making their repetition difficult to notice for large m. The Blum-Blum-Shub generalization lacks this key property. Any noticeable patterns in the terms of an LCG can be removed by eliminating the first few digits, making this random number generator useful to this day.

References

[1] T. E. Hull and A. R. Dobell. Random number generators. *SIAM Review*, 4:230–54, 1962.

[2] Donald Knuth. *Art of Computer Programming*, volume 2. Addison-Wesley, Reading, Massachusetts, 1997.