Linear Congruential Generators

25 September, 2021

1 Linear Congruential Sequences

Definition 1 (Linear Congruential Sequence)**.** *Let* $m, a, c, X_0 \in \mathbb{Z}$ *such that* $0 < m$, $0 \leqslant a < m$, $0 \leqslant c < m$, *and* $0 \leqslant X_0 < m$. *We call* m *the* **modulus**, a *the* **multiplier**, c *the* **increment**, *and* X_0 *the* **starting value**. *Starting from* X_0, *define a sequence*

$$X_{n+1} = (aX_n + c) \mod m, \ n \geqslant 0. \tag{1}$$

We call the this a **linear congruential sequence** *(or* `LCS`*).*

Consider the case where $m = 10$ and $X_0 = a = c = 3$. Performing the calculations reveals the following sequence:

$$\begin{aligned}
X_0 &= 3, \\
X_1 &= (3 \cdot 3 + 3) \mod 10 = 2, \\
X_2 &= (3 \cdot 2 + 3) \mod 10 = 9, \\
X_3 &= (3 \cdot 9 + 3) \mod 10 = 0, \\
X_4 &= (3 \cdot 0 + 3) \mod 10 = 3, \\
X_5 &= (3 \cdot 3 + 3) \mod 10 = 2, \ldots
\end{aligned}$$

Notice that $X_4 = X_0$, and so the sequence repeats. The number of terms in the repeating part of a sequence is called the **period**, and so our example has a period of 4.

The period need not include X_0. Taking $m = 10$ again but with $X_0 = 1, a = c = 2$, we get

$$\begin{aligned}
X_0 &= 1, \\
X_1 &= (2 \cdot 1 + 2) \mod 10 = 4, \\
X_2 &= (2 \cdot 4 + 2) \mod 10 = 0, \\
X_3 &= (2 \cdot 0 + 2) \mod 10 = 2, \\
X_4 &= (2 \cdot 2 + 2) \mod 10 = 6, \\
X_5 &= (2 \cdot 6 + 2) \mod 10 = 4, \ldots
\end{aligned}$$

Although the repeating doesn't occur until X_5, we say that the period is 4 because it started over at X_1.

However, this only happens when $\gcd(a, m) \neq 1$.

Theorem 2. *Consider a* `LCS` *with parameters* m, a, c, X_0 *such that* $\gcd(a, m) = 1$. *Then,* X_0 *will always occur in the period.*

Proof. Suppose the period is p. Then, $X_{p+k} = X_k$ for some $0 \leqslant k < p$ and X_k is the first number in the period. If $k > 0$, we can construct X_{k-1} from X_k by taking $X_{k-1} = a^{-1}(X_k - c) \mod m$. This construction works because a has a multiplicative inverse modulo m as a result of them being relatively prime. But then we can get $X_{p+k-1} = X_{k-1}$ from X_{p+k} by a similar construction, which means X_k is not the first number in the period. Therefore, $k = 0$. □

With that said, our goal is to study `LCS` for use as a (pseudo-)random number generator, so our next step will be choosing good parameters.

2 Choosing Parameters

2.1 The Modulus, m

When choosing the modulus, we want m to be very large. This is because the period of a `LCS` can be *at most* m due to the cyclic nature of modular arithmetic.

On the other hand, for computing, we also need to worry about speed. Because of this, typical values of m tend to be (near) the *word size* of the system it is implemented on. For example, in a 64-bit system, $m = 2^{64} \pm 1$.

Why not just $m = 2^{64}$? Because the lower-order bits tend to be less random than the higher-order bits.

Choose m and d such that $d \mid m$. If $Y_n = X_n \mod d$, then $Y_{n+1} = (aY_n + c) \mod d$. In particular, if $m = 2^e$, then we see that the lower-order ε bits will have periods of at most 2^ε – the last 4 bits will have a period of at most 16, or even worse, the final bit will either be constant or strictly alternating. This doesn't happen if we work with $m = 2^e \pm 1$.

For our implementation, we will actually take $m = 2^{31} - 1$. This is because we will be implementing `MINSTD` [4]. Note that we will be using a different multiplier than in the cited article, as the authors later advocated for the one we will use here.

2.2 The Multiplier, a

We want to maximize our period, because we want to have enough terms to suit our application. But not only that, we want enough "randomness" to not make the next output too obvious. For instance, taking $a = c = 1$ certainly gives a sequence of period m, but it's merely a counter from 0 to $m - 1$ that wraps back around to 0 on the mth iteration.

So, can we achieve a maximum period length of m without losing the random nature of the output?

Theorem 3 (Period m)**.** *The* `LCS` *with parameters* m, a, c, X_0 *has a period of length* m *iff*

1. c *is relatively prime to* m,

2. $b = a - 1$ *is a multiple of* p *for all primes* p *dividing* m,

3. b *is a multiple of* 4 *if* m *is a multiple of* 4.

For a proof of this theorem, check out Volume 2 of Knuth's Art of Computer Programming [3].

Since we are implementing `MINSTD`, we will be choosing $a = 48271$, $c = 0$, and let $X_0 \neq 0$ be defined when constructing the generator.

3 Code in Scheme

We're going to use what we've learned about what a `LCS` is to implement the following algorithm in Scheme, which uses my favorite theorem from number theory – that (colloquially) the probability that two randomly selected integers are relatively prime is $\frac{6}{\pi^2}$. We use `Rand` to denote an idealized[1] random number generator in specifying this algorithm.

Algorithm 1 Cesàro's method for approximating π

Require: $trials \geqslant 1$
  $passes \leftarrow 0$
  $N \leftarrow trials$
  while $N \neq 0$ **do**
    $a \leftarrow$ `Rand`
    $b \leftarrow$ `Rand`
    if $\gcd(a, b) = 1$ **then**
      $passes \leftarrow passes + 1$
    end if
    $N \leftarrow N - 1$
  end while
    return $\sqrt{\frac{6}{passes/trials}}$ ▷ This will be our approximation of π.

Unfortunately, in the real world, we don't have immediate access to "true" randomness for use in our computations, so we have to approximate the randomness used in our approximation of π. This is where the `LCG` comes in.

A **linear congruential generator** (`LCG`) is just a `LCS` being used as a random generator. Below is an implementation of `MINSTD`, a particular `LCG`, in Scheme which takes a user-provided seed.

```
(define minstd
  (lambda(X)
    (lambda()
      (define (gen n) (modulo (* n 48271) 2147483647))
      (set! X (gen X)) X)))
```

Using this, we may instantiate the `LCG` with a seed, in this case 1.

```
(define rand (minstd 1))
```

[1] We take this random generator to be *actually* random and to allow for any integer value.

Now we implement `Algorithm 1`. This application of random number generation comes from the SICP book [1].

```
(define (estimate-pi n)
  (sqrt (/ 6 (monte-carlo n cesaro))))
```

And this is the Cesàro experiment.

```
(define (cesaro)
  (= (gcd (rand) (rand)) 1))
```

To perform and tally up the results of the experiment is the Monte-Carlo procedure.

```
(define (monte-carlo trials experiment)
  (define (iter remaining passed)
    (cond ((= remaining 0)
           (/ passed trials))
          ((experiment)
           (iter (-1+ remaining)
                 (1+ passed)))
          (else
            (iter (-1+ remaining)
                  passed))))
    (iter trials 0))
```

Finally, in order to display our results, we will use a little procedure to display a line nicely (with a newline character at the end).

```
(define (display-line t)
  (display t)
  (newline)
  t)
```

And now we estimate π using 50000 experiments.

```
(display-line (estimate-pi 50000))
```

We then run in `mit-scheme`.

```
$ mit-scheme --quiet < cesaro.scm
3.1421796166243956
```

Appendix: Proof of the Correctness

In this section, we will justify the correctness of the method implemented in the code above.

Number Theory

First of all, we need to correctly formulate the notion of selecting "random" integers. This is because it isn't technically correct to say that we're selecting "random" integers, because there is no uniform distribution on the integers. Instead, we can "approximate" a uniform distribution on $\mathbb{N}$ by the notion of *natural density*.

Definition 4 (Natural Density)**.** *Let $A \subseteq \mathbb{N}^k$ for some $k \geqslant 1$. Let $A(x) = \#\{(a_1, \ldots a_k) \in A : a_i \leqslant x\}$, where $\#S$ denotes the number of elements in the set S. We say that A has* **natural density** *α if*

$$\alpha = d(A) = \lim_{x \to \infty} \frac{A(x)}{x^k}.$$

In other words, we're not looking at a uniform distribution on $\mathbb{N}^k$, we're looking at the *limit* of uniform distributions on finite intervals in $\mathbb{N}^k$. This is why, though technically inaccurate, one will hear of selecting "random" integers – we're really selecting random integers in the finite intervals $I_n^k := \mathbb{N}^k \cap [1, n]^k$ and taking the limit of the resulting probabilities that those integers are in A as $n \to \infty$.

Will will now take a few results of number theory as given, or else we'll end up writing a book on analytic number theory which happens to have a discussion of linear congruential generators as a preface. The theorems that will be used below can be found in any good book on number theory. For instance, Apostol [2].

I assume that the reader already knows about big-O notation, as well as big-Ω and big-Θ, but I will go ahead and recall a notation that's not as often studied in a typical algorithms course.

Definition 5 (Asymptotic Notation)**.** *We write $f(n) \sim g(n)$ if*

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1.$$

Note that $f(n) \sim g(n)$ automatically gives $f(n) = O(g(n))$.

We'll start with a lemma that's well-known in both mathematics and computer science.

Lemma 6 (Asymptotics of the Harmonic Numbers)**.**

$$H_n := \sum_{1 \leqslant d \leqslant n} \frac{1}{d} \sim \log(n).$$

Proof. First, note that

$$H_n \leqslant 1 + \int_1^n \frac{dt}{t} = 1 + \log(n).$$

On the other hand,

$$\log(n+1) = \int_1^{n+1} \frac{dt}{t} \leqslant H_n.$$

Since we may write $\log(n+1) = \log\left(n\left(1+\frac{1}{n}\right)\right) = \log(n) + \log\left(1+\frac{1}{n}\right)$, we have that

$$1 + \frac{\log\left(1+\frac{1}{n}\right)}{\log(n)} \leqslant \frac{H_n}{\log(n)} \leqslant 1 + \frac{1}{\log(n)}.$$

Taking $n \to \infty$ gives

$$1 \leqslant \lim_{n\to\infty} \frac{H_n}{\log(n)} \leqslant 1,$$

which, by the squeeze theorem, gives

$$\lim_{n\to\infty} \frac{H_n}{\log(n)} = 1,$$

as required. □

Let us now prove the following theorem.

Theorem 7 (Number of Relatively Prime k-tuples)**.** *Let $R_k \subseteq \mathbb{N}^k$ be the number of all relatively prime k-tuples in $\mathbb{N}^k$. Then,*

$$R_k(x) = \begin{cases} \frac{x^k}{\zeta(k)} + O\left(x^{k-1}\right) & \text{if } k \geqslant 3 \\ \frac{x^2}{\zeta(2)} + O(x \log x) & \text{if } k = 2 \end{cases}.$$

Before we prove this theorem, let's first prove what we're really after as a corollary.

Corollary 8 (Probability of Relative Primality)**.**

$$d(R_k) = \frac{1}{\zeta(k)}.$$

Proof. Theorem (7) gives

$$d(R_k) = \lim_{x\to\infty} \frac{R_k(x)}{x^k} = \lim_{x\to\infty} \frac{1}{\zeta(k)} + o(1) = \frac{1}{\zeta(k)}. \quad \square$$

The case $k = 2$ is of specific importance (both to our application and in historical significance), and we will later see how to evaluate $\zeta(2)$.

Proof of Theorem 7. First, note that

$$R_x(x) = \sum_{\substack{\gcd(a_1,\ldots,a_k)=1 \\ 1\leqslant a_i \leqslant x}} 1 \tag{2}$$

and

$$\lfloor x \rfloor^k = \sum_{\substack{1\leqslant a_i \leqslant x \\ (i=1,2,\ldots,k)}} 1 = \sum_{1\leqslant d\leqslant x} \sum_{\substack{\gcd(a_1,\ldots,a_k)=d \\ 1\leqslant a_i \leqslant x}} 1. \tag{3}$$

If $\gcd(a_1,\ldots,a_k)=d$, then $\gcd(a_1/d,\ldots,a_k/d)=1$, and so the inner sum of (3) is just $R_k(x/d)$. Using Möbius Inversion, we turn

$$\lfloor x\rfloor^k=\sum_{1\leqslant d\leqslant x}R_k(x/d) \tag{4}$$

into

$$R_k(x)=\sum_{1\leqslant d\leqslant x}\mu(d)\lfloor x/d\rfloor^k=\sum_{1\leqslant d\leqslant x}\mu(d)\left[x/d+O(1)\right]^k. \tag{5}$$

Expanding this out yields

$$R_k(x)=x^k\sum_{1\leqslant d\leqslant x}\mu(d)/d^k+x^{k-1}O\left(\sum_{1\leqslant d\leqslant x}\mu(d)/d^{k-1}\right)+\cdots+xO\left(\sum_{1\leqslant d\leqslant x}\mu(d)/d\right)+O\left(\sum_{1\leqslant d\leqslant x}1\right). \tag{6}$$

The first term, sans x^k, may be written as

$$\sum_{1\leqslant d\leqslant x}\mu(d)/d^k=\sum_{d=1}^{\infty}\mu(d)/d^k-\sum_{d=\lfloor x\rfloor+1}^{\infty}\mu(d)/d^k.$$

As any number theorist knows, $\displaystyle\sum_{d=1}^{\infty}\mu(d)/d^k=\frac{1}{\zeta(k)}$.

As for the second term in this difference,

$$\left|\sum_{d=\lfloor x\rfloor+1}^{\infty}\mu(d)/d^k\right|\leqslant\sum_{\lfloor x\rfloor+1}^{\infty}\frac{1}{d^k}<\int_{\lfloor x\rfloor}^{\infty}\frac{dt}{t^k}=O\left(1/x^{k-1}\right).$$

Therefore, the first term of (6) is $x^k/\zeta(k)+O(x)$. The other terms of the form

$$\sum_{1\leqslant d\leqslant x}\mu(d)/d^{\ell},$$

where $\ell>1$, are all $O(1)$. The final term is $O(x)$, since it's just a sum of $\lfloor x\rfloor$ copies of 1.

Finally, by Lemma 6, we have

$$\left|\sum_{1\leqslant d\leqslant x}\mu(d)/d\right|\leqslant\sum_{1\leqslant d\leqslant x}\frac{1}{d}\sim\log(x).$$

Combining all of these expressions together gives the desired result. □

Fourier Analysis – Evaluating $\zeta(2)$

As previously, in order to not write a book on Fourier Analysis, we will keep it short and take some notions as given. Of particular importance is Parseval's identity.

Theorem 9 (Parseval's Identity)**.** *For a function $f \in L^2[-\pi, \pi]$, with Fourier coefficients*

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx,$$

we have that

$$\int_{-\pi}^{\pi} |f(x)|^2 dx = 2\pi \sum_{n=-\infty}^{\infty} |c_n|^2.$$

A proof of this can be found in Rudin [5].

We now evaluate $\zeta(2)$, which is a famous problem known as the *Basel problem.*

Theorem 10 (Basel Problem)**.**

$$\zeta(2) = \frac{\pi^2}{6}.$$

Proof. Note that the function given by $f(x) = x$ is in $L^2[-\pi, \pi]$. Therefore, we may use Parseval's identity to get

$$2\pi \sum_{n=-\infty}^{\infty} |c_n|^2 = \int_{-\pi}^{\pi} x^2 dx = \frac{2\pi^3}{3}.$$

The Fourier coefficients are

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} x e^{-inx} dx = \frac{(-1)^n}{n} i,$$

and so $|c_n|^2 = \frac{1}{n^2}$ for $n \neq 0$ and 0 for $n = 0$. This means that

$$2\pi \sum_{n=-\infty}^{\infty} |c_n|^2 = 4\pi \sum_{n=1}^{\infty} \frac{1}{n^2} = 4\pi\zeta(2).$$

Dividing both sides of

$$4\pi\zeta(2) = \frac{2\pi^3}{3}$$

by 4π gives

$$\zeta(2) = \frac{\pi^2}{6},$$

as required. □

Corollary 11. `Algorithm 1` *is correct.* □

References

[1] Harold Abelson, Gerald Jay. Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1996.

[2] Tom M. Apostol. *Introduction to Analytic Number Theory.* Springer, 2011.

[3] Donald Ervin Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1997.

[4] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, 1988.

[5] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 1976.