Random Number Generator Maps: A Custom Lehmer Generator for Drawing Random Cards From a Deck

Rachel Huang

Math 441

December 10, 2025

1 Abstract

Random number generators (RNG's) are a tool that generate sequences of numbers that cannot be predicted better than by random chance. RNG's are an essential component of advanced cryptographic algorithms and complex system models, and are often implemented in computer programming. The focus of this project is a specific type of pseudo-RNG, the Lehmer generator, that simulates random number generation by generating long sequences of numbers that appear to be random. A custom Lehmer RNG was created for the context of randomly picking cards from a standard deck, by selecting appropriate parameters and analyzing the quality of the generated sequences. Since the Lehmer generator's behavior was constrained to discrete integer outputs, the custom model was extended to a real number space by executing a change of variables. The resulting map exhibited chaotic and unpredictable behavior that serves as a source of inspiration for selecting the Lehmer generator's parameters to ensure that its output sequences can properly simulate the generation of random numbers.

2 Introduction

There are two types of RNG's that can be programmed: true RNG's and pseudo RNG's. True RNG's generate numbers by measuring physical processes that are inherently random, such as atmospheric noise and quantum phenomena. Pseudo RNG's are algorithms that generate long sequences of numbers that are statistically random and uniformly distributed. The pseudo RNG that was analyzed in this project was the Lehmer generator. The Lehmer generator, also called the Prime Modulus Multiplicative Linear Congruential Generator, is similar to the the Bernoulli shift map, but with integer restrictions placed on its parameters

and input (Park & Miller, 1988).

The parameters and input to the Lehmer generator are discrete integers, where the seed (initial input) equals z_0 and $f(z) = az \bmod m$. $z_{n+1} = f(z_n)$ for $n = 1, 2, ...$, and is defined for all integer input $z \in [1, m-1]$. The modulus m must be a large prime integer, and the multiplier a must be an integer in the range $[2, m-1]$.

$$f(z) = az \bmod m \tag{1}$$

The Lehmer generator satisfies the three goals of a good pseudo-random number generator if its parameters a and m are properly selected (Park & Miller, 1988):

1. The output sequence is statistically indistinguishable from a sequence drawn at random without replacement.
2. The sequence has a full period (i.e. the sequence repeats every $m-1$ elements).
3. The created map can be efficiently implemented on 32 or 64 bits of computer hardware.

The optimal Lehmer generator for 32-bit machines was determined to be $f(z) = 16807z \bmod 2147483645$, by (Park & Miller, 1988; Herring & Palmore, 1989). $a = 16807 = 7^5$, and $m = 2147483645 = 2^{31} - 1$, where $2^{31} - 1$ ensures that the longest possible sequence of the generator is within the maximum possible value that can be stored on 32-bit machines. The selected parameters essentially generate the longest possible output sequence without surpassing the machine's physical storage capabilities.

Note: To properly understand the model for this project, the terms fixed point, period, and limit cycles are applied to the context of the Lehmer generator map. Fixed points are output values of the map that don't change in value even after applying the map again. In essence, the map gets "stuck" at that value. The Lehmer generator map also produces ordered, fixed length sequences based on the selected parameters and input seed. The generated sequences repeat in a fixed cycle and the period is defined to be the length of the sequence that is repeated. A full period generator has output that has the greatest possible period (determined by the selected parameters). Finally, limit cycles refer to the sequences that are repeated periodically, as the output of the map will always approach the periodic sequence and keep cycling if the map is iterated enough times. Any valid seed circularly shifts a limit cycle if the generator is full period.

3 Model

To demonstrate the characteristics of the Lehmer generator in a tangible situation, a Lehmer generator for drawing random cards from a standard deck of

length 52 was created.

$$f(z) = 5z \bmod 53 \tag{2}$$

Note: The model assumes that the 52 cards are labeled from 1-52, so that the possible output values of the map can be manipulated to fall within the integer range $[1, m-1] = [1, 52]$. The cards are also assumed to be chosen at random without replacement.

The values of the two parameters for this model, a and m, were chosen to be 5 and 53, respectively. These parameters were chosen for their combined characteristics that satisfied the aforementioned three properties of a good psuedo random number generator.

The parameter m was chosen to be 53, as the modulus determines the maximum possible period of the sequence. The maximum possible period is equal to $m-1$, and had to be 52 in order to cover all of the cards that could be selected during an iteration of the map. The generated sequence should ideally be equal to its greatest potential value, because the Lehmer generator is deterministic and a fixed sequence is generated corresponding to each valid input seed. Since the order of elements in the generated sequences is fixed, and the sequences are periodic, a greater period ensures that it takes as long as possible to encounter a repeat in the output sequence. This could theoretically be accomplished for the chosen card-drawing scenario by taking the optimal Lehmer generator and modding the outputs by 52. However, for the purposes of demonstrating the behavior of the Lehmer generator, an output sequence with period of length 52 was ideal, as it was still applicable to the chosen scenario and exhibited certain patterns that we were interested in.

In addition to being the length of our desired period + 1, 53 was also prime. A prime integer was necessary in order to ensure that the output of the map could not collapse to 0 for any possible input. In addition, 53 did not result in the generation of any fixed points. For example, if $m = 10$, the map's output sequence would have collapsed to a fixed point immediately after the first iteration for $z_0 = 1$:

$$f(1) = (5 \cdot 1) \bmod 10 = 5$$

$$f(5) = (25 \cdot 1) \bmod 10 = 5$$

$$f(5) = (25 \cdot 1) \bmod 10 = 5$$

Next, the multiplier a was chosen be 5 because it ensured that the actual period of the sequence was the full period of 52. For example, if we had selected $a = 10$, the actual period of the generated sequence would have only been 13.

Human preference also plays a role in the selection of a. For instance, if $a = 2$, the generated sequence would have been the following: [1, 2, 4, 8, 16, 32,

11, 22, 44, 35, 17, 34, 15...]. This sequence also had a full period of 52, but the first 6 elements were in monotonically increasing order, making the sequence less preferable (Park & Miller, 1988).

For the created Lehmer generator model in equation (2), the first and third goals for a good pseudo random number generator were assumed to be a given because of the negligible length of the generated sequences. The statistically random and uniformly distributed characteristics of the Lehmer generator's output have been verified by multiple researchers (Park & Miller; Herring & Palmore). Rather than having to verify these properties for the created map, the conclusions of these results were assumed to hold since these properties were verified for any map of the Lehmer generator structure in equation (1). In addition, the concerns regarding implementation efficiency were largely dependent on the issue of overflow, which occurs when the value of the largest and smallest numbers that can be physically stored on a machine are surpassed. Older machines were 32-bit, which meant that the maximum possible integer that could be physically represented by the machine was $2^{31} - 1$, and the minimum possible integer was -2^{31} (Park & Miller, 1988). Thus, the full period of the optimal Lehmer generator was determined to be $2^{31} - 2$, and since 52 is significantly smaller than this value, it was assumed hat equation (2) had an efficient implementation on the modern 64-bit machine used in this project.

4 Results

Note: All experiments regarding the created model from equation (2) used an input seed of $z_0 = 1$.

Since the parameter m was defined to be a prime integer, there were only 15 prime numbers in the valid range of $[1, 53]$. Output sequences were generated for each prime modulus to determine what parameter values resulted in a fixed point.

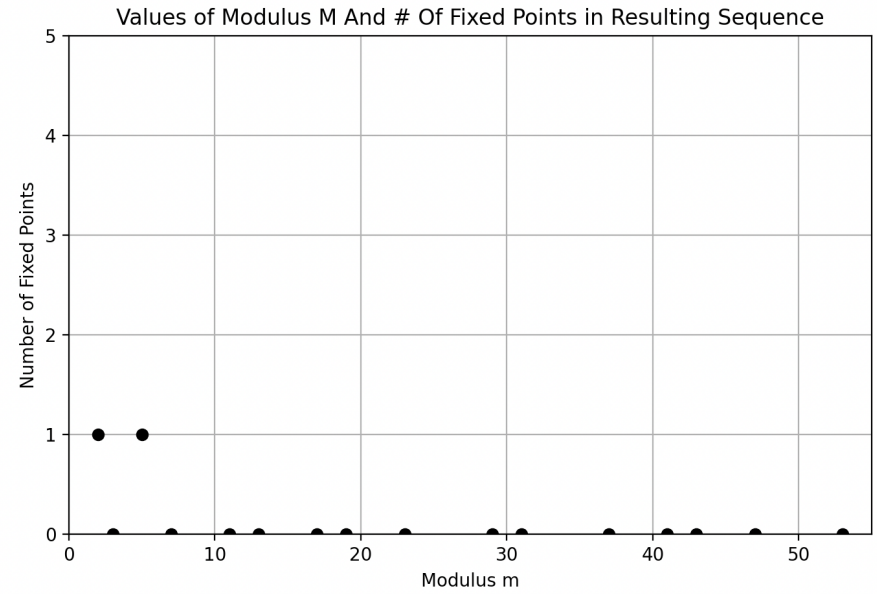

Figure 1: Prime moduli plotted against resulting number of fixed points in each generated output sequence.

Only two prime moduli, 2 and 5, resulted in a fixed point while generating an output sequence. For $m = 2$, the map $f(z) = 5z \bmod 2$ immediately approached the fixed point 1 after the first iteration. For $m = 5$, the map immediately approached the fixed point 0 on the first iteration. Since 5 is prime, this should have prevented the output of the map from reaching 0, but $a = m = 5$, which resulted in 0 for any input seed. Both of these fixed points were attracting, as the fixed point could only be approached from the left with subsequent iterations, and the output was unable to move past it. In the chosen context of randomly selecting a card, $m = 2$ and $m = 5$ were undesirable parameter values, as they would keep selecting the card indexed 1, and the card indexed 0, respectively.

There were no resulting fixed points for all possible values of the multiplier parameter a in the valid range $[2, m - 1]$ for equation (2).

Next, the output sequence of equation (2) was plotted to visualize its periodic behavior.

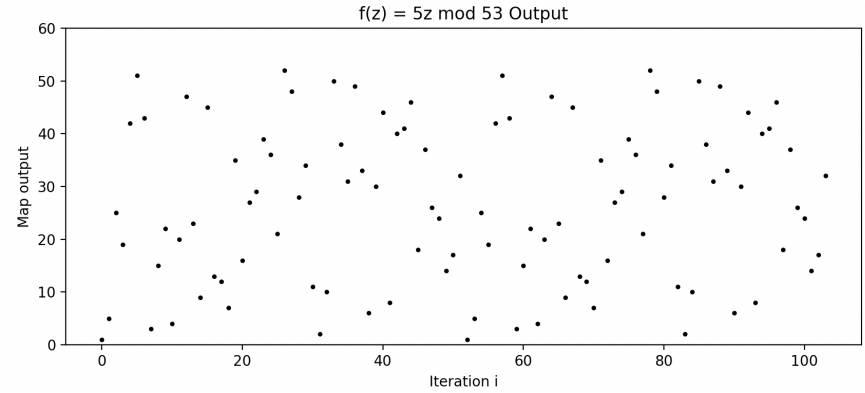

Figure 2: The discrete output sequence of f(z) = 5z mod 53 was plotted for two periods.

The integer constraints of the Lehmer generator were applicable to the chosen context of randomly picking numbered cards, yet made it difficult to visualize the model's behavior. To help visualize system behavior, the integer restrictions on the multiplier a were removed, since our integer sampling from equation (2) corresponded to a discrete sampling from the broader dynamics of a continuous map where a was a real parameter. If a became a continuous, real parameter, z had to also become a continuum of inputs if we wanted to observe system dynamics that were not limited to finite integer permutations. The model in equation (2) became the following, where a was defined to be a continuous, real parameter.

$$f(x) = (ax) \bmod 53, a \in \mathbb{R} > 0 \tag{3}$$

We then defined a normalized input $y = \frac{x}{53}$. Plugging this into equation (3), we saw that $\frac{f(x)=(ax) \bmod 53}{53} = f(y) = (ay) \bmod 1$. The updated model now had a continuous multiplier a and continuous input y.

$$f(y) = (ay) \bmod 1, a \in \mathbb{R} > 0 \tag{4}$$

The output of equation (4) was then plotted to visualize the system's dynamics.

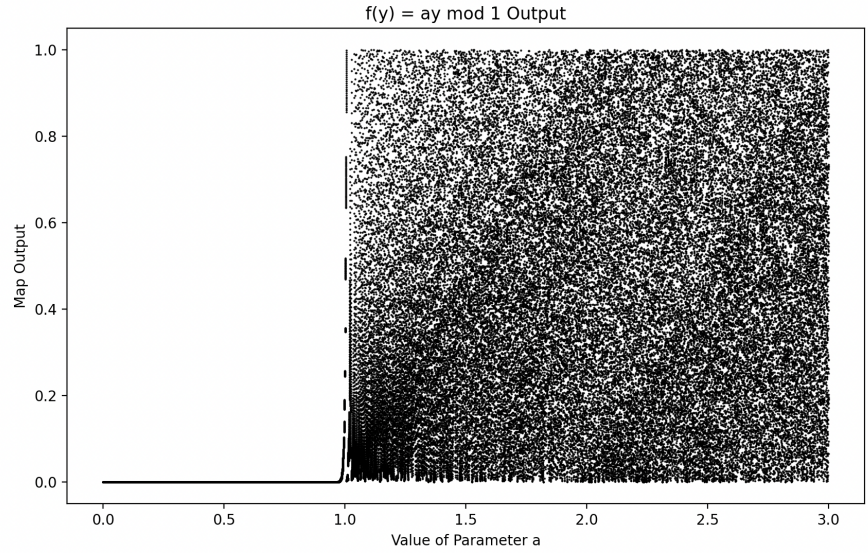

Figure 3: The long-term output generated by $f(y) = (ay) \bmod 1$ when $a \in [0, 3]$.

Figure 3 exhibited three distinct behaviors when $0 < a < 1$, $a = 1$, and $a > 1$. When $0 < a < 1$, all inputs eventually collapsed to 0. Thus, 0 was an attracting fixed point. When $a = 1$, $f(y) = y$, and thus all points were attracting fixed points. When $a > 1$, $f'(y) = a > 1$, which meant that all of the fixed points became unstable. The system was then equivalent to the Bernoulli shift map (with a generalized a), that exhibited chaotic behavior due to its aperiodic long-term behavior, deterministic nature, and sensitive dependence on initial conditions. For $a > 1$, the visual block of static appeared to be random and non-repeating. In addition, equation (4) was completely deterministic, as the initial value, parameter a, and number of iterations would produce the exact same output if the same values were used. The sensitive dependence on initial conditions was visualized by each vertical slice (corresponding to a value of a) containing points that spread across the entire output range even if the initial values differed by a minuscule amount. Two initial conditions that were arbitrarily close diverged rapidly and filled the output interval in an unpredictable manner.

A bifurcation occurs at $a = 1$, when the system transitions from having a single, stable fixed point to devolving into chaotic behavior. The transition occurs at the degenerate line of fixed points that appear when $a = 1$.

5 Discussion

The integer constraints of the created Lehmer generator map $f(z) = 5z \bmod 53$ produced a statistically random and uniformly distributed sequence of integers that corresponded to randomly selecting an indexed card from a standard 52-count deck. However, variations in the two parameters, the multiplier a and the modulus m were only able to result in finite permutations of integer sequences, lacking the system dynamics of nonlinear, continuous systems that we were interested in. By lifting the integer restrictions of equation (2), the continuous map $f(y) = (ay) \bmod 1$ was created. With this new map, output was no longer limited to fixed, ordered sequences. In fact, when the parameter a was increased past $a = 1$, the output of the map became chaotic and unpredictable. The modification of lifting the integer restrictions on the multiplier parameter a and input from equation (2) to equation (4) greatly improved the unpredictability of the generated sequences.

The continuous version of the model equation (4) no longer corresponded to the discrete scenario of selecting random cards. For future work, a transformation could be created to take the output from equation (4) and convert it back to the discrete range of integers in the range $[1, 52]$. Perhaps the $[0, 1]$ range of equation (4) could be divided into 52 segments, and output in a segment would correspond to selecting a card of that segment's index. In addition, the optimal Lehmer generator could be used to address the scenario of selecting random cards if its output was modded by 52. The significantly greater period of the optimal Lehmer generator could result in the appearance of additional behaviors that were not easily visualized in the limited sequence length of (2).

Furthermore, technological advancements have resulted in the machine standard to be 64-bit, rather than 32-bit. The optimal Lehmer generator could thus be improved to have a significantly greater period on 64-bit machines, since the optimal generator described in the analyzed literature was created for 32-bit hardware.

References

Herring, C., & Palmore, J. I. (1989). Random number generators are chaotic. *ACM SIGPLAN Notices*, *24*(11), 76–79. https://doi.org/10.1145/71605.71608

Park, S. K., & Miller, K. W. (1988). Random Number Generators: Good Ones Are Hard to Find. *Communications of the ACM*, *31*(10), 1192–1201. https://doi.org/10.1145/63039.63042

Strogatz, S. H. (2024). *Nonlinear Dynamics and Chaos*. CRC Press.