(Pseudo)Random Number Generation

Krish

Outline

Introduction

Problem

How can computers, which are purely deterministic, generate random numbers?

Before discussing this, let's talk about randomness in nature.

Introduction

Problem

How can computers, which are purely deterministic, generate random numbers?

Before discussing this, let's talk about randomness in nature.

Nature is full of random fluctuations: rate of radioactive decay, the number of faces on a pebble, thermal noise, etc.

Introduction

Problem

How can computers, which are purely deterministic, generate random numbers?

Before discussing this, let's talk about randomness in nature.

Nature is full of random fluctuations: rate of radioactive decay, the number of faces on a pebble, thermal noise, etc.

It is possible, in theory, to measure one such random event and use that to generate a random number.

- This does pose some issues when it comes to generating massive numbers, as well as being very inefficient and slow.

Introduction

Problem

How can computers, which are purely deterministic, generate random numbers?

Before discussing this, let's talk about randomness in nature.

Nature is full of random fluctuations: rate of radioactive decay, the number of faces on a pebble, thermal noise, etc.

It is possible, in theory, to measure one such random event and use that to generate a random number.

- This does pose some issues when it comes to generating massive numbers, as well as being very inefficient and slow.

Thus, we must turn to algorithms to generate *pseudorandom* numbers.

What Does "Pseudorandom" Even Mean?

A pseudorandom number is defined as a number that "appears to be statistically random, despite having been produced by a completely deterministic and repeatable process" (Wikipedia).

What Does "Pseudorandom" Even Mean?

A pseudorandom number is defined as a number that "appears to be statistically random, despite having been produced by a completely deterministic and repeatable process" (Wikipedia).

Basically, at first glance, a pseudorandom number appears random, but after enough numbers have been generated, a clear pattern emerges.

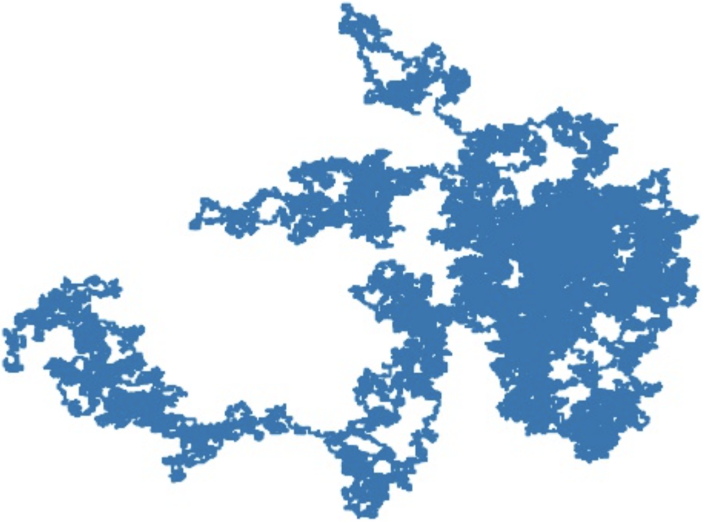
What Does "Pseudorandom" Even Mean?

A pseudorandom number is defined as a number that "appears to be statistically random, despite having been produced by a completely deterministic and repeatable process" (Wikipedia).

Basically, at first glance, a pseudorandom number appears random, but after enough numbers have been generated, a clear pattern emerges.

We can visualize this using random walks, which show the progression and relation between a set of generated numbers by plotting the distance between two consecutive numbers in a set of numbers.

Truly Random Visualization



Pseudorandom Visualization

Middle Squares Method

1. Start by choosing a seed. The last three digits of the Unix time as I'm making this are 186, so that's our seed number.

Middle Squares Method

1. Start by choosing a seed. The last three digits of the Unix time as I'm making this are 186, so that's our seed number.
2. Now square this number and select as many middle digits as needed (here we're doing 3). $186^2 = 34596$, so our next seed is 459.

Middle Squares Method

1. Start by choosing a seed. The last three digits of the Unix time as I'm making this are 186, so that's our seed number.
2. Now square this number and select as many middle digits as needed (here we're doing 3). $186^2 = 34596$, so our next seed is 459.
3. Repeat using this number as the new seed.

Middle Squares Method

1. Start by choosing a seed. The last three digits of the Unix time as I'm making this are 186, so that's our seed number.
2. Now square this number and select as many middle digits as needed (here we're doing 3). $186^2 = 34596$, so our next seed is 459.
3. Repeat using this number as the new seed.

Thus, to generate random 3-digit numbers with a seed of 186, we get: $186^2 = 34596 \rightarrow 459$, $459^2 = 210681 \rightarrow 068$, $68^2 = 4624 \rightarrow 624$, $624^2 = 389376 \rightarrow 937$, $937^2 = 877969 \rightarrow 796$, and so on.

Middle Squares Method

1. Start by choosing a seed. The last three digits of the Unix time as I'm making this are 186, so that's our seed number.
2. Now square this number and select as many middle digits as needed (here we're doing 3). $186^2 = 34596$, so our next seed is 459.
3. Repeat using this number as the new seed.

Thus, to generate random 3-digit numbers with a seed of 186, we get: $186^2 = 34596 \rightarrow 459$, $459^2 = 210681 \rightarrow 068$, $68^2 = 4624 \rightarrow 624$, $624^2 = 389376 \rightarrow 937$, $937^2 = 877969 \rightarrow 796$, and so on.

Once you get a repeat number, it's easy to see that the numbers will start to repeat from there. Some seeds will have a longer period than others.

Linear Congruential Generator (LCG)

The LCG is another simple pseudorandom number generator.

The general LCG formula is:

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

Linear Congruential Generator (LCG)

The LCG is another simple pseudorandom number generator.

The general LCG formula is:

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

Where:

- X_n is the current seed
- X_{n+1} is the next pseudorandom number
- a is the multiplier
- c is the increment
- m is the modulus
- Assume all are non-zero

Example LCG Computation

Given arbitrary parameters: $m = 79, \quad a = 43, \quad c = 15, \quad X_0 = 21$

1. $X_1 = (43 \cdot 21 + 15) \mod 79$

   $43 \cdot 21 = 903 \rightarrow 903 + 15 = 918 \rightarrow 918 \mod 79 = 63$

   $X_1 = 63$

2. $X_2 = (43 \cdot 63 + 15) \mod 79$
   $43 \cdot 63 = 2709 \rightarrow 2709 + 15 = 2724 \rightarrow 2724 \mod 79 = 9$

   $X_2 = 9$

3. $X_3 = (43 \cdot 9 + 15) \mod 79$

   $43 \cdot 9 = 387 \rightarrow 387 + 15 = 402 \rightarrow 402 \mod 79 = 7$

   $X_3 = 7$

We get a sequence of 21, 63, 9, 7, 0, 15, 34, and so on.

XorShift

General Process:

1. Pick a seed and convert it to binary.
2. Perform a bit shift either left or right and any amount.
3. Perform the Xor operation ($\oplus$) on the numbers in steps 1 and 2.
4. Repeat steps 1 – 3 as many times as you want, reversing the direction of the bit shift each time.
5. Convert the final number back to decimal.

Example with Arbitrary Seed 3146505

Step	**Result of Operation**
Initial	0000 0000 0011 0000 0000 0011 0000 1001
Left Shift $\ll 13$	0001 1000 0000 0000 0000 0000 0000 0000
Xor1: $12345 \oplus (12345 \ll 13)$	0001 1000 0011 0000 0000 0011 0000 1001
Right Shift $\gg 17$	0000 0000 0000 0001 1000 0000 0000 0000
Xor2: Previous $\oplus$ (Previous $\gg 17$)	0001 1000 0011 0001 1000 0011 0000 1001
Left Shift $\ll 5$	0011 0000 0110 0010 0000 0110 0010 0000
Xor3: Previous $\oplus$ (Previous $\ll 5$)	0010 1000 0101 0011 1000 0101 0010 1001

Results

Iteration	Initial Seed	Operations	Result
0	3146505	Initial seed	3146505
1	3146505	$3146505 \oplus (3146505 \ll 13)$	405799689
2	405799689	$405799689 \oplus (405799689 \gg 17)$	405897993
3	405897993	$405897993 \oplus (405897993 \ll 5)$	676562217

Pseudorandom number: 676562217

Summary

Middle Squares, LCG, and XorShift are three fairly simple pseudorandom number generation algorithms.

Summary

Middle Squares, LCG, and XorShift are three fairly simple pseudorandom number generation algorithms.

All three can be completed in constant time, making them suitable for real-time simulation number generation.

Summary

Middle Squares, LCG, and XorShift are three fairly simple pseudorandom number generation algorithms.

All three can be completed in constant time, making them suitable for real-time simulation number generation.

This does mean that neither is great for cryptographic purposes, which often use many complex processes like entropy pools, block ciphers, and sometimes even truly random numbers from physical processes as I mentioned earlier. I highly recommend reading up on these as they are very interesting!

Questions?