National Cheng Kung University

Mechanical Engineering

Stochastic Dynamic Data - Analysis and Processing

Random Number Generator

Author:
Zhao Kai-Wen

Supervisor:
Chang Ren-Jung

October 4, 2012

Contents

1 Introduction

The assignment can divided into 2 parts.

- MATLAB simulation
- Algorithm implementation

In first section, I manipulate the MATLAB function to generate random numbers and observe the time-series of random data.
In the algorithm design part, I followed the linear recurrences method to write my program. I further verify its deterministic property and analyze the data generated from my code.

2 MATLAB Simulation

MATLAB is a powerful tool to do simulation and it is facile to generate random data by primitive function *rand*().

2.1 Time series random data

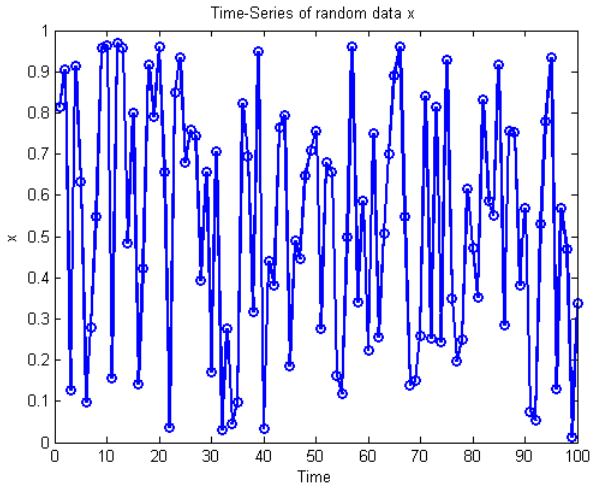

Figure 1: Generate a random vector with 100 elements.

2.2 Histogram

Histogram is a statistical tool to illustrate frequencies that data would show. It is widely used in image processing, which is one of fileds I am interested in, to visualize its intensity distribution. Now I plot the histogram with 100 random data. But it shows that data scattered around and the interval of 0.5 0.7 takes a large portion.

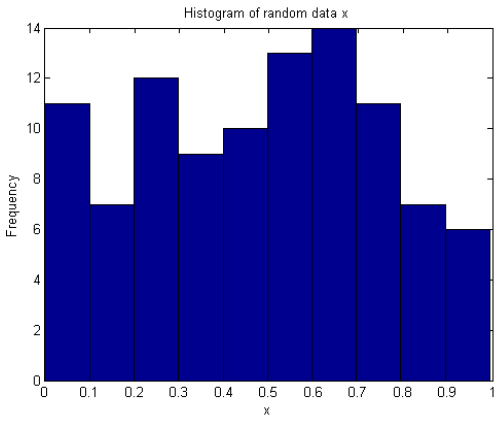

Figure 2: $N = 100$

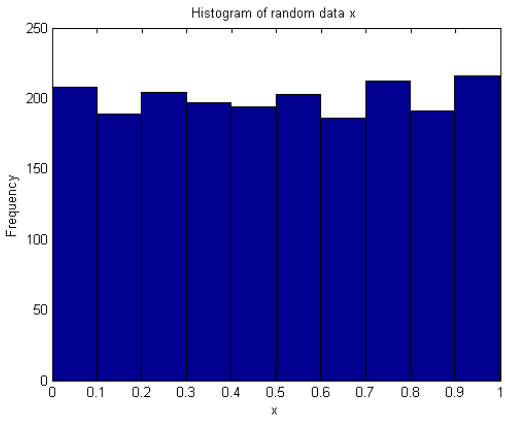

Figure 3: $N = 2000$

However, on the right hand side, I got 2000 data and the graph demostrates that different values corresponde to almost same frequency($\sim$ 200). It indicates that each value has approximatly same probability to show up. I think that is an important and essential factor of a good RNG.

2.3 'Tossing a coin' simulation

To support the idea that the true mean of random data is 0.5, I programmed a simulation to ' toss a coin '. I used MATLAB RNG to creat different length of vector. Exponentially increase the tossing number and plot it in semilog scale as next page.

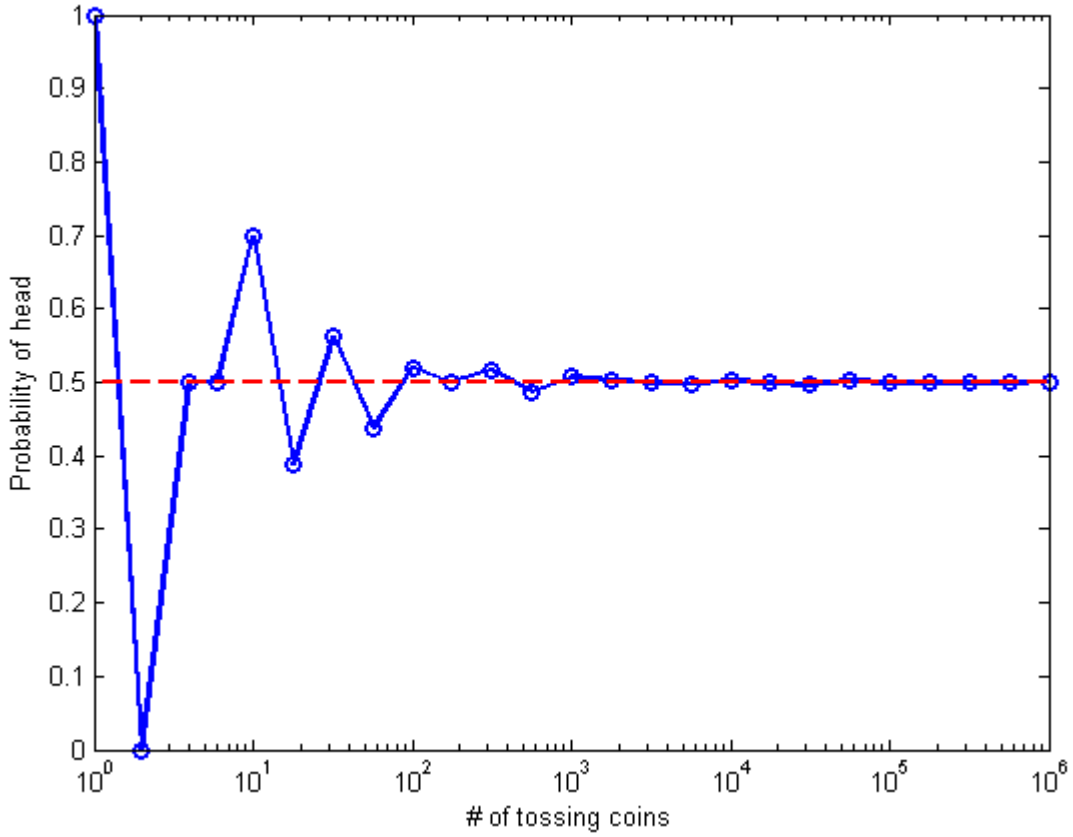

Figure 4: Tossing a coin simulation

The horizontal axis shows tossing times and and vertical axis is the probability of 'head'. When I just toss a coin once, it is a head, and the probability is 1. That's tricky but correct. The probability of head asymptotically approach 0.5, the thoretical mean, when I the tossing times increase.
The the number approaches $N = 10^6$ times, the probability is 0.5.

3 Algorithm Implementation

There are two kinds of random number. One is generated from physical device or hardware which lacks any pattern. Such appratus is based on microscopic phenomena that generate random noise signal, such as thermal noise or other quantum phenomena.
The other I focused on is RN generated by software. In computer science, random number generator is useful for cryptography and simulation. Applications of random number abounds, I recently use random number to test my program's **average performance**.
However, RNG designed by software is a challenge because algorithm itself is a set of **deterministic processes** and it must be finite and non-ambigious. So, the RNG algorithm actually contains many complex steps. We give it a **'seed'** as initial condition and algorithm does many calculations which is too complicated to find its pattern.

Figure 5: RNG algorithm

Following this block diagram, we find that if we send the same initial condition, the same result would come out because the algorithm is a series of certain steps. For programmers, the most commonly used method is **extracting system time as the seed** . (Since the system clock is strictly increasing and never repeat. On the other word, it is a one-way ticket.)

3.1 My linear recurrent RNG

I followed the concept mentioned above to write the RNG and compare to MATLAB version.

```
%% Random Number Generator

function u = lc_rand( n )
 m = 2^31 -1 ;

 k = 1 ;
a1 = 1385320287 ;
 x = zeros( 1 , n) ;

% the specific value for verifying the algorithm is deterministic
%  x(1)  = 1 ;

% Using system time as the seed
seed  = clock() ;
 x(1) = seed(6) ;

for i = k : n-k
   x( i +1) = mod(a1*x(i) , m) ;
end
u = x/m ;
end
```

Figure 6: Source Code

On left hand side is MATLAB primitive function, the opposite side is my function. I use green dash line to illustrate its mean value.

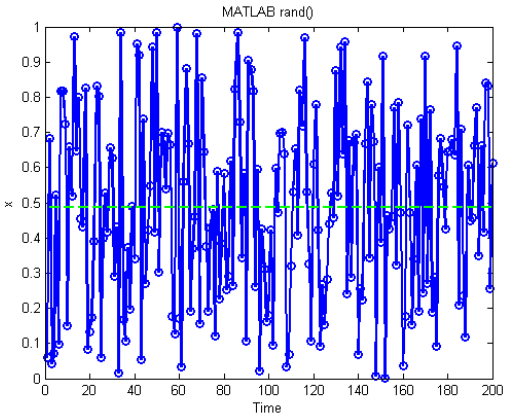

Figure 7: $MATLAB$

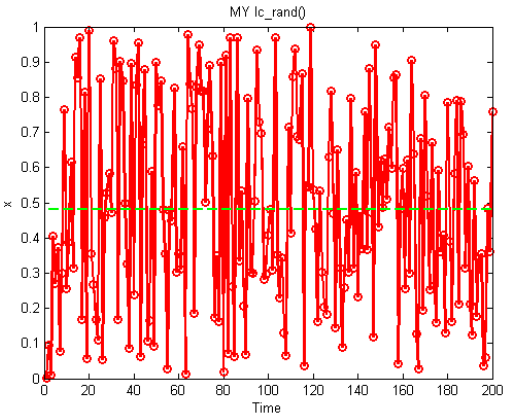

Figure 8: $MINE$

I think that they both have scattered data and not illustrate any pattern or periodicity. Analyze them with histogram and list different m, k, a_1, a_2

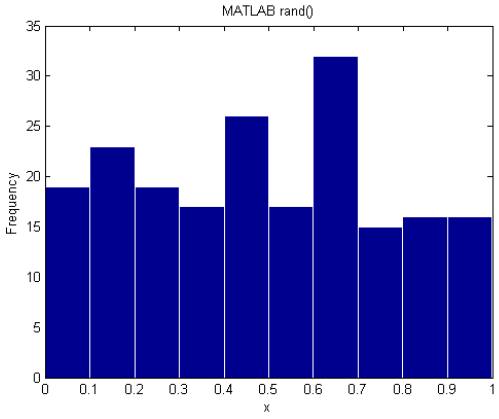

Figure 9: $MATLAB$

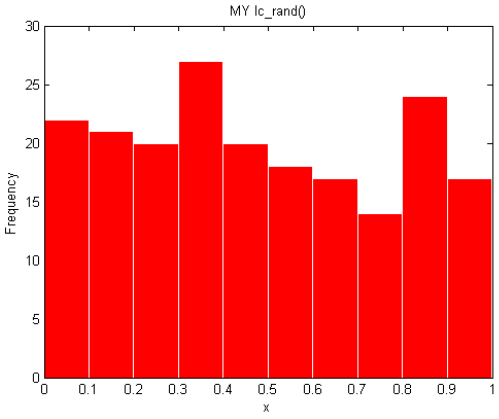

Figure 10: $MINE$

Let N approaches to large number.

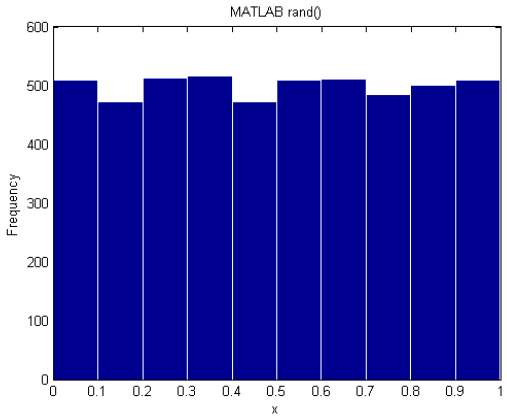

Figure 11: $MATLAB$

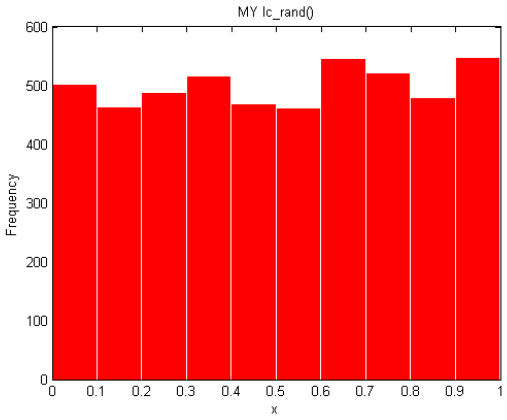

Figure 12: $MINE$

I can say that they are both uniformly distributed.

3.2 RNG with different multiplier

We have the table of algorithm setting parameters.

function()	k	m	a_1	a_2
$lcg\_1()$	1	$2^{31}-1$	1385320287	-
$lcg\_2()$	1	$2^{31}-1$	41358	-
$mrg\_1()$	2	$2^{31}-1$	1498809829	1160990996
$mrg\_2()$	2	$2^{31}-1$	46325	1084587

I seperately test these four sets of parameters and observe the histogram they produced.

3.2.1 Time series data

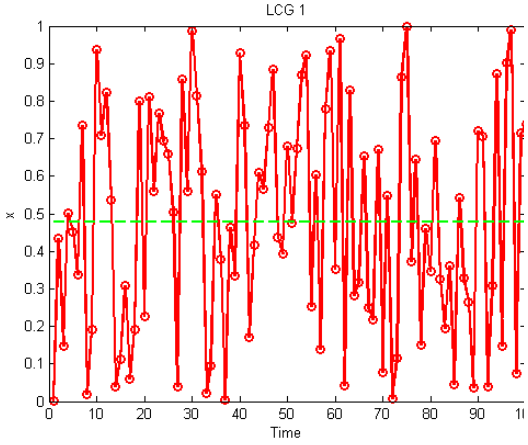

Figure 13: $LCG1$

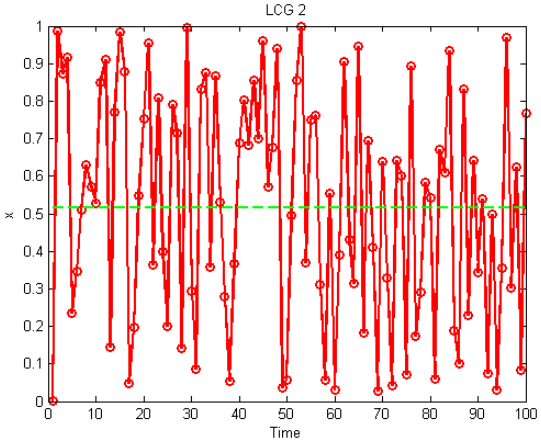

Figure 14: $LCG2$

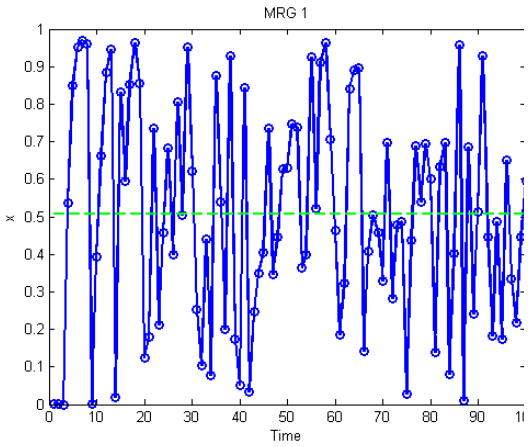

Figure 15: $MRG1$

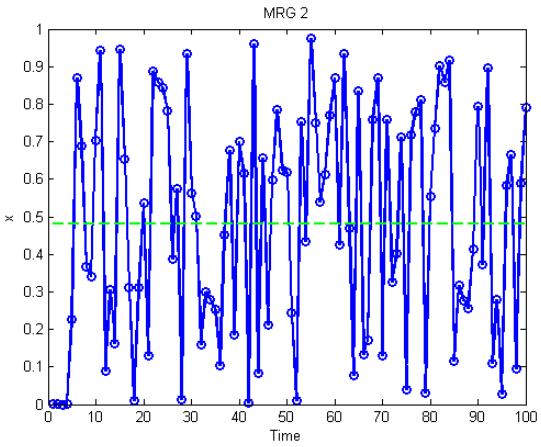

Figure 16: $MRG2$

3.2.2 Histogram

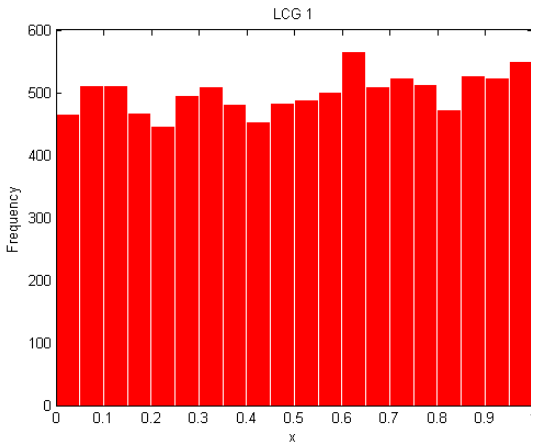

Figure 17: $LCG1$

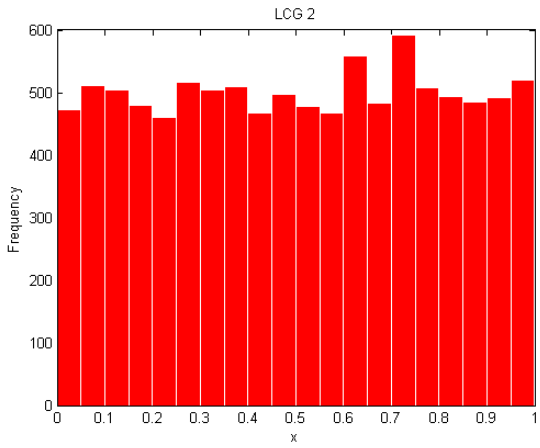

Figure 18: $LCG2$

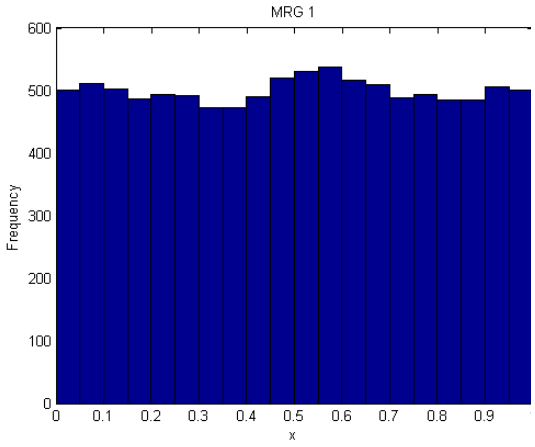

Figure 19: $MRG1$

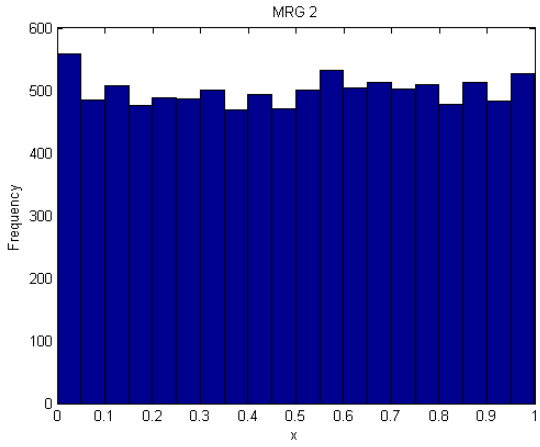

Figure 20: $MRG2$

By observing the diagram, I can conclude that no matter which multiplier we choose, data seem to be random and uniformly distributed.

3.3 Look into MATLAB *rand*()

The MATLAB default algorithm is called Mersenne twister designed by a Japaness named 松本 真. The algorithm is based on a matrix linear recurrence.

$$x_{k+n} = x_{k+m} \otimes (x_k^u | x_{k+1}^l) A$$

where $\otimes$ is the bitwise exclusive or.
MATLAB is very mighty. They provide various and efficient algorithm and we only need to pass an argument to select them.

- multiplicative congruential generator
- multiplicative lagged Fibonacci generator
- combined multiple recursive generator
- modified subtract with borrow generator

So, we can easily find out that MATLAB has a totally different algorithm with which we programmed.

3.4 "Pseudo" RNG?

As previous mentioned, the algorithm can only follow the precise steps, so every number produced by algorithm is deterministic. I can verify this phenomenon by passing a same initial condition(same seed $s_0 = 1$).

3.4.1 *lcg_1*()

Running times	x_1	x_2	x_3	x_4	x_5
$1st$	0.0000	0.6451	0.3925	0.2763	0.1888
$2nd$	0.0000	0.6451	0.3925	0.2763	0.1888
$3rd$	0.0000	0.6451	0.3925	0.2763	0.1888
...	0.0000	0.6451	0.3925	0.2763	0.1888
nth	0.0000	0.6451	0.3925	0.2763	0.1888

3.4.2 $lcg\_2()$

Running times	x_1	x_2	x_3	x_4	x_5
$1st$	0.0000	0.0000	0.7965	0.9058	0.0046
$2nd$	0.0000	0.0000	0.7965	0.9058	0.0046
$3rd$	0.0000	0.0000	0.7965	0.9058	0.0046
...	0.0000	0.0000	0.7965	0.9058	0.0046
nth	0.0000	0.0000	0.7965	0.9058	0.0046

3.4.3 $mrg\_1()$

Running times	x_1	x_2	x_3	x_4	x_5
$1st$	0.0000	0.0000	0.0000	0.3959	0.1329
$2nd$	0.0000	0.0000	0.0000	0.3959	0.1329
$3rd$	0.0000	0.0000	0.0000	0.3959	0.1329
...	0.0000	0.0000	0.0000	0.3959	0.1329
nth	0.0000	0.0000	0.0000	0.3959	0.1329

3.4.4 $mrg\_2()$

Running times	x_1	x_2	x_3	x_4	x_5
$1st$	0.0000	0.0000	0.0000	0.0004	0.9290
$2nd$	0.0000	0.0000	0.0000	0.0004	0.9290
$3rd$	0.0000	0.0000	0.0000	0.0004	0.9290
...	0.0000	0.0000	0.0000	0.0004	0.9290
nth	0.0000	0.0000	0.0000	0.0004	0.9290

The deterministic properties are shown by repeatedly numerical verification.

3.5 Discuss simulated results

Frankly to say, I found two significant problems embeded in the algorithm.

3.6 Zero initial values

This situation caused by the algorithm itself. Because I set small values (relative to $m = 2^{31} - 1$) which is system clock in second as seed. They would generate several zeros due to modulus operation. Hopefully, when samples are large enough numbers gernerated are considered 'legal' random data.

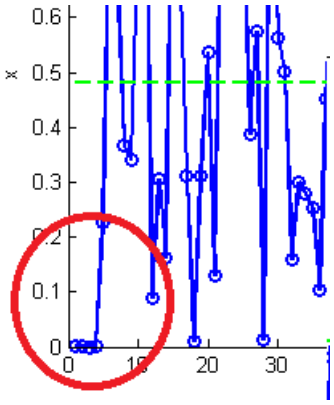

Figure 21: Zero initials

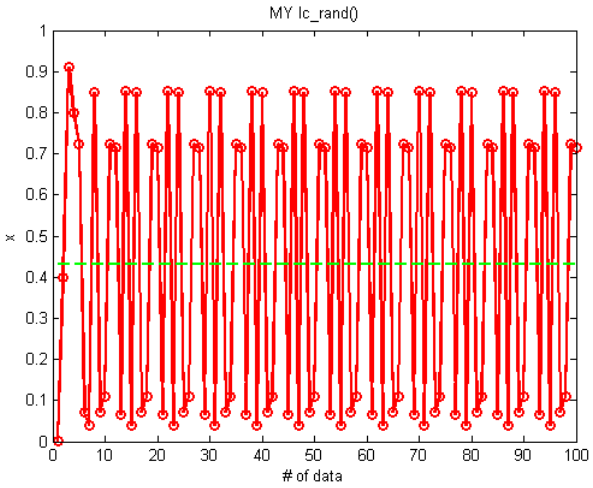

Figure 22: Periodic phenomenon

3.7 Periodicity

The problem is caused by the property of modulus operation. There is a chance that $mod(x_i, m)$ would repeat. The $\rho = m^k - 1$ is the index to prove periodicity occurs.

References

[1] MATLAB: User' s guide. Massachusetts: The Math Works Inc., 2009.

[2] Wikipedia : Random number generation, 2012

[3] Wikipedia : Pseudorandom number generator, 2012

[4] Wikipedia : Linear congruential generator., 2012