International J. of Math. Sci. & Engg. Appls. (IJMSEA)
ISSN 0973-9424, Vol. 11 No. I (April, 2017), pp. 185-193

A NEW RANDOM NUMBER GENERATOR USING FIBONACCI SERIES

KOTTA NAGALAKSHMI RACHANA[1] AND SOUBHIK CHAKRABORTY[2]
[1,2] Department of Mathematics,
BIT Mesra, Ranchi-835215, Jharkhand, India

Abstract

Extensive research is going on for developing good random number generators (RNG). No RNG is full proof. One problem is cycle. Another, perhaps bigger problem is that numbers generated from an arbitrary seed may not be random. In this paper we are trying to develop a new random number generator using Fibonacci series. In doing so an application of Fibonacci series combined with linear congruential method would be explored. Our model which uses Fibonacci series requires a division by the seed which eliminates the problem of cycle in random numbers which is otherwise present in linear congruential method. Hence the technique so proposed can be considered as an improvement. Our proposed method is also efficient in that numbers of arbitrary length generated from arbitrary positions were tested by run test for randomness and found to be random. This is achieved through programs run in Dev C++ software. Hence the second problem of improper seed selection stands resolved.

Key Words : *Random number generator, Linear congruential method, Fibonacci series, seed, Cycle, Run test for randomness.*

AMS Subject Classification : 62P99.

The paper is organised as follows. Section 1 is the introduction; section 2 gives the state of the art. The proposed model and the testing algorithm are explained in section 3 followed by experimental results in section 4. Sections 5 and 6 are reserved for discussion and concluding remarks respectively.

1. Introduction

Random number may be defined as any number whose occurrence of digits cannot be predicted basing on past value. Random number generation is a method to generate numbers in a random manner to acquire random characteristics. Unpredictable character of a random number has uncountable applications. Random processes are the raw materials of classical statistical inferences. In statistical methods in research most of the applications require a "random sample" to study the whole population, since the law of statistical regularity states that only a random sample represents a population. In the development of statistical methods also random processes' observations may be used. Not only in statistical field, but also in other areas like Cryptography (for encryption, key generation) in the medical field for the randomized control trails (RCT) in scientific modelling in Video poker machines which require to deal with random card sequences using a virtual deck of cards and for developing different game, we would be requiring the exploration of efficient random number generators. Some of the important applications in secured systems would require efficient random numbers to evade from breaking of a given designed algorithm. Different necessities employ different types of random numbers.

In order to terminate the cycle of pseudo-random numbers, i.e., in the case of linear congruential generator it has combined with Fibonacci numbers followed by a division by the seed. Mastumoto (2007) et.al. suggested that most RNG fail due to improper seed selection and this led us to chose a random seed for its initialization. To test randomness of the sequence generated by our developed RNG, it would undergo the Wald-Wolfowitz runs test. A run of a sequence is a maximal non empty segment of the sequence consisting of adjacent equal elements. Or we may say a run is a sequence of letters of the one kind surrounded by a sequence of letters of the other kind (except the first and last position where it is only followed by or preceded by respectively). The program developed in Dev C++ of the pseudo code given in section 3 would give the generated sequence as well as a percentage pass of the run test for sub strings of

arbitrary length taken from arbitrary positions.

2. State of the Art

Lehmer (1951) proposed the linear congruential method which is a pseudo-random number generator (PRNG) with additive constant (**C**) as zero. Usage of additive constant as non zero was done in further research. Thomson (1958) developed a modified congruence method of generating pseudo-random Numbers, Franklin (1958) researched on the equidistribution of pseudo-random Numbers, Rotenberg (1960) proposed a new RNG, and Greenbereger (1961) gave notes on a new PRNG. Tausworthe (1965) proposed a generator that produces numbers which are generated by modulo 2 linear recurrence techniques long used to generate binary codes for communications. Panneton, et.al (2006) proposed a fast uniform RNG with extremely long periods have been defined and implemented based on linear recurrences modulo 2. Mastumoto et.al (2007) have experimented with 58 RNG and found 40 to be defective, unwanted pattern caused by non systematic choice of seed rather than recurrence as mentioned earlier. Further literature can be found in Kennedy and Gentle (1980). For applications of pseudorandom numbers in cryptography, we refer the reader to Luby (1996). A more recent book on computational statistics is by Gentle (2009).

3. Methodology

a. Our New Random Number Generator :

Our new RNG has the following equations:

$$X_{i+1} = \left(a * X_i + c + int\left(\frac{f_i}{X_0}\right)\right) \ (mod\ m) \ \text{ for } i = 0, 1, 2, \cdots$$

where X : sequence of random numbers, X_0 : seed value /or arbitrary position.

X_i, a, c, m are integers, int : takes integral part.

f_i : i-th element of the Fibonacci series ($f_i = f_{i-1} + f_{i-2}$) we are choosing $a = 5$, $c = 1$, $m = 2^{32}$ (see the remark below).

Remark, : The values of a and c are to be selected so that the period (length of a cycle) of a RNG is maximized. In our case there is no cycle and so we are dropping the constraint in choosing the values of a, c strictly. For illustrative purpose, we take $a = 5$ and $c = 1$.

b. Verifying the Goodness of our New RNG Through Run Test of Randomness :

We re-emphasize that the proposed RNG would be better in the absence of a cycle and the ability of the RNG to generate numbers of arbitrary length even from arbitrary seed passing the randomness test successfully. We would take samples of different sizes from the arbitrary position of the sequence generated by the random number generator. We would take null hypothesis H_O as the sequence of observations is random and alternating hypothesis H_1 as the sequence of observations is not random.

Run Test of Randomness :

Let R is a random number which takes the number of runs in a given sequence, For each sequence of digits, runs of the sequence would be counted and expected number of runs and its variance of a given sequence would be counted using formulae given below

$$E(R) = \frac{n+2}{2} \quad \text{and} \quad Var(R) = \frac{n}{4}\left(\frac{n-2}{n-1}\right)$$

where n is the size of the sequence generated (including seed).

For large $n(> 25)$, R may be regarded as asymptotically normal and we may use the normal test using statistic $Z = \frac{R-E(R)}{\sqrt{Var(R)}} \sim N(0,1)$, asymptotically.

If $|Z|$ calulated < 1.96, we may accept the null hypothesis at 5% level of significance otherwise reject null hypothesis.

This test would be repeated for different samples sizes and from arbitrary positions.

c. Algorithm to Test the Randomness of Sequence Generated for a Given length n and from an Arbitrary Position (seed) :

Step 1 : Initialize the values as $a = 5, c = 1, m = 2^{32}$.

Step 2 : Input size of the sample and store in n.

Step 3 : Generate Fibonacci series and store it in an array $f[n]$.

Step 4 : $X[0]$ is initialized with random number using rand ( ) for arbitrary position.

Step 5 : Generate sequence using

$$X[i+1] = (a * X[i] + c + f[n]/X[0])\ mod m,$$

store it in an array $X[n]$.

Step 6 : From $i = 0$ to n

Print all the elements of $X[i]$.

Step 7 : To find the median

- sort the elements from $i = 0$ to n in ascending order
- if $n + 1$ is even (including seed value odd n will be even) then median = mean of $\left[\frac{(n+1)}{2}\right]^{\text{th}}$ and $\left[\left[\frac{(n+1)}{2}\right] + 1\right]^{\text{th}}$ term
- if n+1 is odd (including seed value even n will be odd) then median = $\left[\frac{((n+1)+1)}{2}\right]^{\text{th}}$ term.

Step 8 : From the first digit to the unsorted array,

- Write L', if the digit is less than the median, or
- Write 'U', if the digit is greater than or equal to the median.

Step 9 : Store all characters in a string $str[n]$, which is obtained from previous step.

Step 10 : Initialize a new character variable i with the first character of $str[n]$ and set $R = 1$.

Step 11 : From $i = 1$ to n

Check If $(str[i]! = str[i - 1])$ increment R by 1 ("!=" implies not equal to) value of R gives the number of runs in the string.

Step 12 : R is asymptotically normal with

Mean $E(R) = ((n + 1) + 2)/2$

Variance $Var(R) = (n + 1)((n + 1) - 2)/4((n + 1) - 1)$

Calculate $Z = [R - E(R)]/\sqrt{Var(R)}$ under H_0

$Z \sim N(0, 1)$ for $n(> 25)$.

Step 13 : If $|Z| < 1.96$, subsequence may be taken as random at 5% level of significance.

4. Experimental Results

a. To test the non cyclic nature of our RNG:

Figures 1 and 2 give some screenshots to display and compare the nature of linear congruential generator with that of our developed RNG. In fig. 1, the presence of cycle has been highlighted. There is no cycle in our developed RNG (fig. 2).

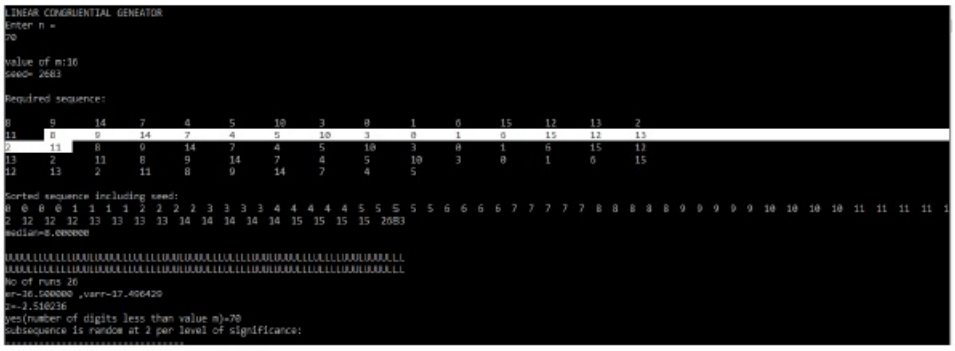

Fig 1: shows that linear congruential generator produces cycle after certain period.

Fig 2:The above output shows that our RNG doesn't produce cycle

To test the randomness of substrings of arbitrary lengths taken from arbitrary positions, table 1 is of interest.

Table 1 : Results of run test on arbitrary substrings and number of passes (score) out of 10 trails of the substring of given length.

Length (n)	Score out of 10
40	8
50	7
60	8
70	7
80	8
90	8
100	7

Length (n)	Score out of 10
110	9
120	7
130	9
140	9
150	8
160	10
170	7
180	9
190	7
200	10
210	10
220	10
230	10
240	8
250	7
260	8
270	9
280	9
290	8
300	10
310	10
320	8
330	9
340	8
350	9
360	10
370	10
380	9
390	9
400	10
410	8
420	9
430	10

Length (n)	Score out of 10
440	7
450	10
460	10
470	8
480	9
490	10
500	9

5. Discussion

Our model which uses Fibonacci series requires a division by the seed which eliminates the problem of cycle in random numbers which is otherwise present in linear congruential method. Hence the technique so proposed can be considered as an improvement. Our proposed method is also efficient in that numbers of arbitrary length generated from arbitrary positions were tested by run test for randomness and found to be random in general. **Let us call this new method FLRNG with the letters F and L standing for Fibonacci and Lehmer respectively.**

Supplementing only Fibonacci element to LCG gives cycle for different m values. In order to terminate the cycle Fibonacci element has to be divided with seed and integral part of the division should be equipped to the LCG. Experimenting with other values of a and c is reserved as a rewarding future work.

6. Concluding Remarks

A great deal of research has been done in generating a sequence which looks like random as well as truly random (a truly random sequence is one whose Kolmogorov complexity equals its own length or in simpler words to generate which we would require a program at least as long as the sequence itself). True random numbers have lots of efficient roles in weighty applications. The study of our RNG and its analysis, done upto sample size of 64,000 in this paper to develop a RNG with a devise of equipping linear congruental generator with integral part of Fibonacci number divided with a given seed, confirms that the sequence generated does not produce cycle. In order to avoid the complexity of culling the inappropriate seed, we are picking a seed randomly (substrings, of arbitrary lengths were taken from arbitrary positions). With these results we could eventually conclude that the developed RNG is a good generator.

References

[1] Lehmer D. H., Mathematical Methods in Large-scale Computing Units, Proceedings of the Second Symposium on Large Scale Digital Computing Machinery, Harvard University Press, Cambridge, (1951), 141-146.

[2] Thomson W. E., A Modified Congruence Method Of Generating Pseudo-Random Numbers, Comp. J., 1 (1958), 83, 86.

[3] Franklin J. N., On the Equidistribution of Pseudo-Random Numbers, Quart. Appl. Math., 16 (1958), 183-188.

[4] Rotenberg A., A New Random Number Generator, JACM, 7 (1960), 75-77.

[5] Greenbereger M., Notes on a New Pseudo-Random Number Generator, JACM, 8 (1961), 163-167.

[6] Tausworthe R. C., Random Numbers Generated by Linear Recurrence Modulo Two, Math. Comp., 19 (1965), 201-209.

[7] Kennedy W. and Gentle J., Statistical Computing, Marcel Dekker Inc. (1980).

[8] Panneton F. P., L'Ecuyer and Matsumoto M., Improved Long- Period Generators Based on Linear Reccurences Modulo 2, ACM Transactions on Mathematical Software, 32(1) (March 2006), 1-16.

[9] Mastumoto M., Wada I., Kuramoto A., Ashihara H., Common defects in Intialization of Pseudo-Random Number Generator, ACM Transcations on Modelling & Computing Simulation, 17(4) (2007).

[10] Luby M., Pseudorandomness and Cryptographic Applications, Princeton Univ Press, (1996).

[11] Gentle J., Computational Statistics, Springer Pub. Co. Inc. N.Y., (2009).