# Pseudo-random number generators

G. Palafox

October 6, 2020

**Abstract**

Various pseudo-random number generators are exhibited and compared. Statistical tests are run to determine their performance. Variations are studied to analyze the sensitivity of the algorithms.

## 1 Introduction

In the present work, different pseudo-random number generators are studied. In particular, methods for generating uniform distributed pseudo-random numbers and normal distributed pseudo-random numbers are studied. Their efficacy is statistically tested with Frosini's uniformity test and Shapiro's Normality test, respectively.Additionally, the sensitivity of some of the methods is tested by varying their input parameters and measuring possible changes in the quality of the output. A time comparison is also carried out to compare the performance of some of the methods. This study was performed with R version 4.0.0 [4] on a Jupyter [3] notebook[1].

## 2 Methods for the uniform distribution

First, the simplest of the methods is studied: the linear congruential generator (LGC). Parameters are varied to look for differences in its output. Then, the Additive Congruential Random Number Generator is looked at. Frosini's test for uniformity [1] is performed on data outputted by both algorithms.

### 2.1 Linear Congruential Method

This method outputs a sequence of $n$ numbers uniformly distributed on $(0, 1)$. Pseudo-code for this method is shown in Algorithm 1. It is based on the recurrence relation

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

where $X_0, a, c$ and $m$ are integers chosen by the user. The algorithm outputs at most $m$ distinct numbers, but the possibility of patterns of length shorter than $m$ occurring exist. Necessary and sufficient conditions for achieving an $m$-length sequence are known [2]: $a, m$ must be coprime, $a-1$ must be divisible by all prime factors of $m$, and $a-1$ need be divisible by 4 if $m$ is divisible by 4. Figure 1a shows an histogram of the output of LGC with a seed of $X_0 = 101$, and $a = 1151, c = 27077, m = 2^{32}$. The data in this histogram gives us a $p$-value of $0.38 > 0.05$ when Frosini's Uniformity test [1] is performed on it. To further study this method, a hundred sequences of length one-thousand were generated, and for each, the $p$-value associated to Frosini's test was computed and stored. This was done twice: first, with parameters $a = 2^{10} + 1, c = 2^{16} + 1, m = 2^{32}$ and then with parameters $a = 2^{10}, c = 2^{16} + 1, m = 2^{32}$. Note the first set of parameters satisfy the aforementioned conditions for maximum length while the second set does not. In particular, $a = 2^{10}, m = 2^{32}$ are not coprime. In the first scenario, the $p$-value was greater than 0.05 in 93 out of 100 times. A boxplot of this $p$-values is shown in Figure 1b. For the second scenario, all hundred $p$-values where significantly less than 0.05.

### 2.2 Additive Congruential Random Number Generator

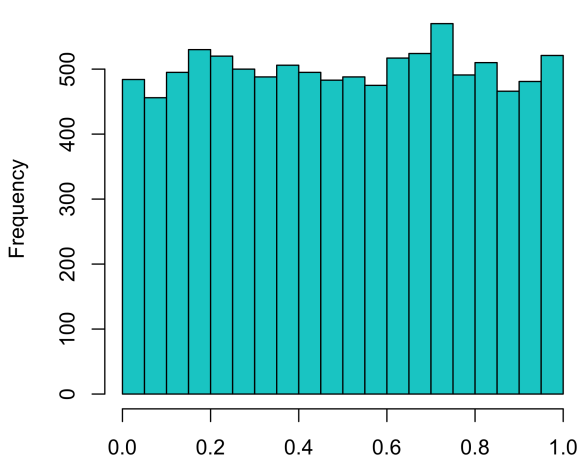
Similar as the LGC, the Additive Congruential Random Number Generator (ACORN) [7] is a method based on modular arithmetic and generates $n$ uniformly distributed pseudo-random numbers. It starts with $k$ initial values $Y_0^m, m = 1, 2, \ldots, k$ all being less than a modulus $M$. With these, the following are defined:

$$Y_n^0 = Y_{n-1}^0, n \geq 1, \tag{2}$$

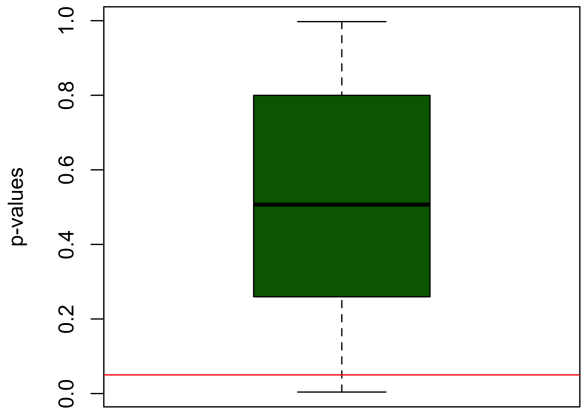
$$Y_n^m = (Y_n^{m-1} + Y_{n-1}^m) \mod M, n \geq 1, m = 1, 2, \ldots, k, \tag{3}$$

$$X_n^k = Y_n^k / M,\ n \geq 1. \tag{4}$$

[1] The notebook with the code containing our analysis, as well as this report, can be found in the Github Repository: `https://github.com/palafox794/AppliedProbabilityModels/tree/master/Assignment5`




(a) Histogram of a LCG sample of size ten-thousand, with parameters $a = 1151, c = 27077, m = 2^{32}$



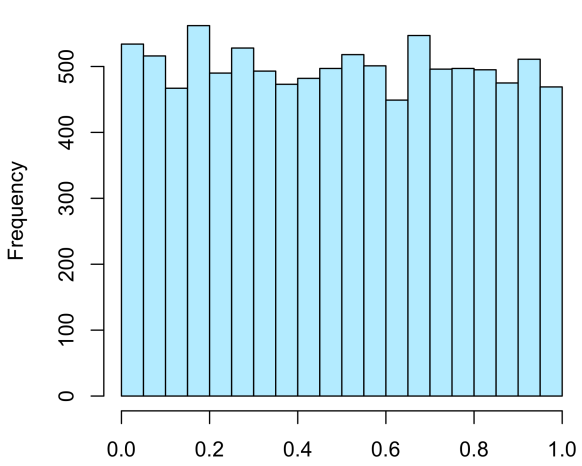

(b) Boxplot of the $p$-values of a hundred LCG samples, with $a = 2^{10} + 1, c = 2^{16} + 1, m = 2^{32}$. Red line is at 0.05

Figure 1: Linear congruential generator.

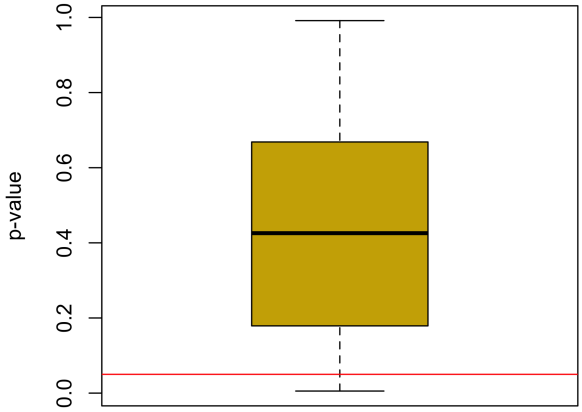
**Algorithm 1** Linear Congruential Generator

**Input:** Positive integers `n, a, c, m, seed`.
**Output:** Array of `n` numbers with Unif(0,1) distribution.

1: Make empty numeric vector `numbers`
2: Make `x` = `seed`
3: **while** length(`numbers`) < `n` **do**
4: Make `x = (a*x + c) mod m`
5: Append `x/(m-1)` to `numbers`
6: **end while**
7: **return** `numbers`




(a) Histogram of one ACORN sample.




(b) Boxplot of the $p$-values of a hundred ACORN samples.

Figure 2: ACORN generated numbers.

The sequence $X_n^k$ is the sequence of ACORN generated pseudo-random numbers. Pseudo-code for this method is found in Algorithm 2. In a similar fashion as with LGC, a hundred samples of ACORN outputs were tested with Frosini's test, and the respective $p$-values were stored. Here, 95 out of a 100 were greater than 0.05. These are displayed in Figure 2b. An histogram of ACORN generated numbers is found in Figure 2a.

**Algorithm 2** Additive Congruential Random Number Generator

**Input:** Positive integers `M, n, y11, y12, ..., y1k`.
**Output:** Array of `n` numbers with Unif(0,1) distribution.

```
Create n × k matrix A with first row equal to y11, y12, ..., y1k
for row = 2, 3, ..., n do
    for col = 1, 2, ... k do
        if col == 1 then
            A[row][col] = A[row-1][col]
        else
            x = A[row][col-1] + A[row-1][col]
            A[row][col] = x mod M
        end if
    end for
end for
Make vector numbers equal to the k-th column of A
return numbers / M
```

# 3 Methods for the normal distribution

Methods for generating normal-distributed pseudo random numbers are shown next. First, the Box-Muller method and its polar form variation are presented. The sensitivity of Box-Muller method is explored. Finally, an algorithm based on the rejection method [6] is given.

## 3.1 Box-Muller method

Algorithm 3 presents the Box-muller method for sampling a pair $z_0, z_1$ of normal distributed pseudo-random numbers. A proof of why the algorithm works is given by Ross [5]. The algorithm was used to generate one-hundred samples of

Table 1: Number of $p$-values smaller and larger than 0.05 for different normal samples

| Method | Less than 0.05 | Greater than 0.05 |
|---|---|---|
| Box-Muller, keeping only $z_0$ | 3 | 97 |
| Box-Muller, keeping only $z_1$ | 6 | 94 |
| Box-Muller, keeping both $z_0, z_1$ | 7 | 93 |
| Using `rnorm` | 4 | 96 |

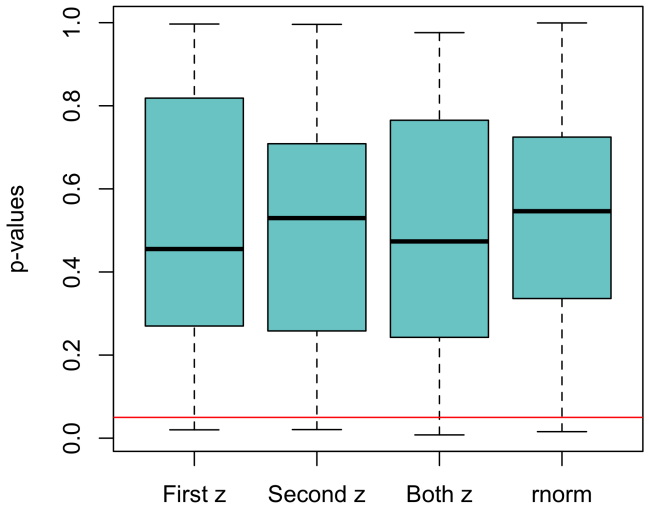



Figure 3: $p$-values obtained with different methods. Red line represents 0.05

five-thousand normally distributed values, where the $p$-values resulting of performing the Shapiro Test on each sample was stored. This was compared to the $p$-values of R's `rnorm` method, and to variations of the Box-Muller method were only the first ($z_0$) or second ($z_1$) values sampled are kept. The comparison of these $p$-values is shown in Figure 3. A summary is also shown in Table 1.

**Algorithm 3** Box-Muller

**Input:** Real numbers `mu, sigma`.
**Output:** Number with Normal(`mu, sigma`) distribution.

1: Generate numbers `u1, u2` $\sim$ Unif$(0, 1)$
2: Make `z0 = sqrt(-2 * log(u1)) * cos (2 * pi * u2)`
3: Make `z1 = sqrt(-2 * log(u1)) * sin (2 * pi * u2)`
4: **return** `sigma * z0 + mu, sigma * z1 + mu`

Next, to study the sensitivity of Algorithm 3, some variations were performed: first, instead of using R's `runif` to generate the uniform numbers, Algorithm 1 (LGC) was used instead. The other two variations consisted of using two *non-independent* uniform distributed numbers. The first one uses $u_1 \sim \text{Unif}(0, 1)$ and $u_2 = \frac{u_1}{2}$, while the second one uses $u_1 \sim \text{Unif}(0, 1)$ and $u_2 = \frac{u_1+1}{2}$. As before, we generated a hundred samples and stored the $p$-values from Shapiro's Test. The results are summarized in Table 2.

It is observed that using a different method for generating the uniform pseudo-random numbers did not have much effect on the output. Using non-independent uniform numbers, however, made caused all samples to fail Shapiro's test.

### 3.1.1 Box-Muller polar form

A known variation of the Box-Muller method [5] is given in Algorithm 4. It works with the same principle, but avoids the computation of trigonometric functions. The performance of both Box-Muller forms was compared, concluding that

Table 2: Number of $p$-values smaller and larger than 0.05 for different Normal samples

| Method | Less than 0.05 | Greater than 0.05 |
|---|---|---|
| Box-Muller, using LGC | 8 | 92 |
| Box-Muller, non-independent uniform values ($u_1 \sim \text{Unif}(0,1)$ and $u_2 = \frac{u_1}{2}$) | 100 | 0 |
| Box-Muller, non-independent uniform values ($u_1 \sim \text{Unif}(0,1)$ and $u_2 = \frac{u_1+1}{2}$) | 100 | 0 |

Algorithm 4 performs slower than Algorithm 3 77% of the time, with an average difference of 0.001 seconds.

**Algorithm 4** Box-Muller Transform. Polar version

**Input:** Real numbers `mu, sigma`.
**Output:** Number with Normal(`mu, sigma`) distribution.

1: **repeat**
2: Generate numbers `u1, u2` $\sim$ Unif$(0,1)$
3: Make `V1 = 2 * u1 - 1`
4: Make `V2 = 2 * u2 - 1`
5: Make `S = V1^2 + V2^2`
6: **until** `S` $\leq 1$
7: Make `Z1 = sqrt( (-2 * log(S) ) / S ) * V1`
8: Make `Z2 = sqrt( (-2 * log(S) ) / S ) * V2`
9: **return** `sigma * Z1 + mu`, `sigma * Z2 + mu`

### 3.2 Rejection method

Algorithm 5 is based on the Rejection Method, and is explained in Ross' Simulation book [6]. As in previous sections, a hundred samples were generated using Algorithm 5 and the $p$-values of this samples under Shapiro's test were stored, as can be seen in Figure 4c. A histogram of a five-thousand sample generated with Algorithm 5 is shown in Figure 4a, next to a histogram of five-thousand numbers generated by R's `rnorm` in Figure 4b.

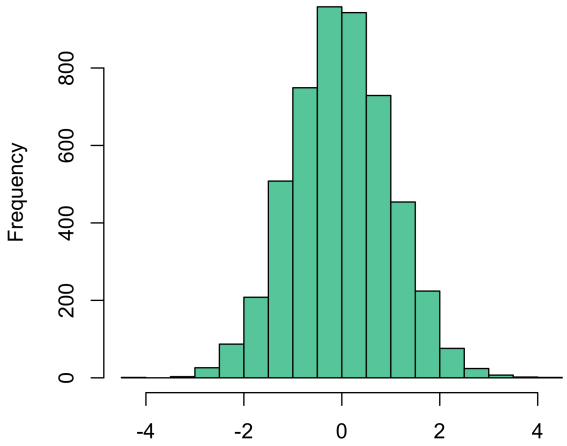
**Algorithm 5** Rejection method. Normal distribution

**Input:** Real numbers `mu, sigma`.
**Output:** Number with Normal(`mu, sigma`) distribution.

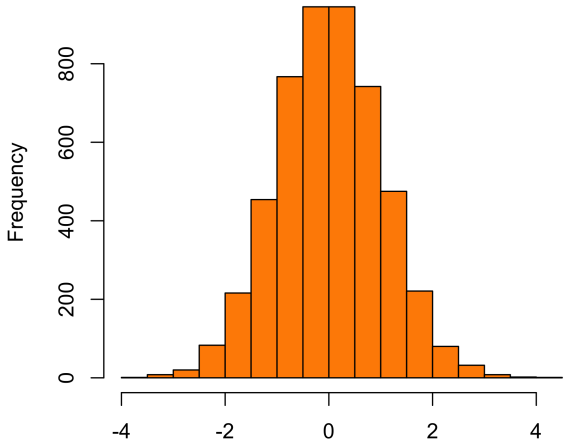
1: Generate `y1, y2` $\sim$ Exp$(1)$
2: **while** `y2 - (y1-1)^2 / 2 < 0` **do**
3: Generate `y1, y2` $\sim$ Exp$(1)$
4: **end while**
5: Generate `u` $\sim$ Unif$(0,1)$
6: **if** `u` $\leq 1/2$ **then**
7: Make `z = y1`
8: **else**
9: Make `z = - y1`
10: **end if**
11: **return** `sigma * z + mu`

# 4 Conclusions

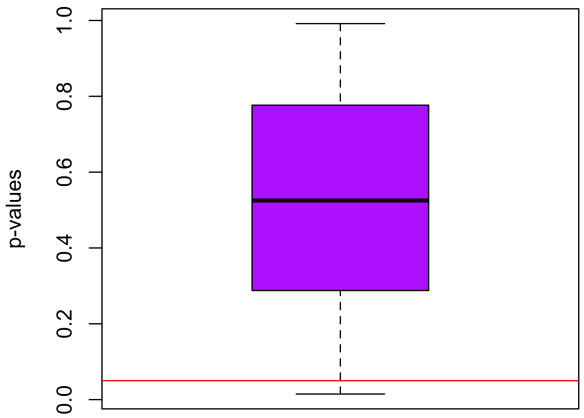
It is observed how methods for different generating non-uniform pseudo-random numbers commonly rely on the generation of uniform distributed pseudo-random numbers. The LGC and ACORN methods both seem to give good results, even when used with methods like Box-Muller's. Further study should include pseudo-random number generators for different distributions. Given how Frosini's and Shapiro's test are both distribution dependent (uniform, normal), the study of generators for new distributions should be accompanied with the study of new statistical tests.




(a) Histogram of one five-thousand number sample given by Algorithm 5.




(b) Histogram of one five-thousand number sample given by `rnorm`.




(c) Boxplot of the $p$-values of a hundred samples given by Algorithm 5.

Figure 4: Comparing Algorithm 5 to `rnorm`.

# 5 Acknowledgments

We thank Professor Elisa Schaeffer for providing code for the Box-Muller method in Algorithm 3 and the Linear Congruential Generator in Algorithm 1.

# References

[1] P. Blinov and B. Lemeshko, *A review of the properties of tests for uniformity*, in 2014 12th International Conference on Actual Problems of Electronics Instrument Engineering (APEIE), IEEE, Oct 2014, p. 540–547.

[2] T. E. Hull and A. R. Dobell, *Random number generators*, SIAM Review, 4 (1962), pp. 230–254.

[3] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, et al., *Jupyter notebooks—a publishing format for reproducible computational workflows*, in Positioning and Power in Academic Publishing: Players, Agents and Agendas: Proceedings of the 20th International Conference on Electronic Publishing, IOS Press, 2016, p. 87.

[4] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2020.

[5] S. M. Ross, *Introduction to probability models*, Harcourt/Academic Press, 7th ed., 2000.

[6] ——, *Simulation*, Elsevier Academic Press, 4th ed ed., 2006.

[7] R. Wikramaratna, *ACORN—a new method for generating sequences of uniformly distributed pseudo-random numbers*, Journal of Computational Physics, 83 (1989), p. 16–31.