Practice 5: Pseudo Random Numbers

Mayra Cristina Berrones Reyes 6291

October 6, 2020

1 Introduction

A pseudo random number is the term used for the computer generated random numbers. The prefix "pseudo" is used be differentiate this type of number to a truly random number that is generated by a random physical process such as radioactive decay [3].

A pseudo random number generator (PRNG) is refereeing to an algorithm that uses mathematical formulas to create sequences that approximate as much as posible random numbers. This type of numbers are very important in the area of computational science, because more often than not, we find the need to use randomness in a computer program to be able to perform experiments. It is quite difficult however, to get a computer to do something by chance, because a computer follows instructions in a way that can be predicted one way or another [1].

Truly random numbers are not possible to generate from a deterministic method, so we use PRNG techniques to develop random numbers using a computer. A PRNG sequence is completely determined by an initial value called seed.

2 Generators based on linear recurrences

A great discovery in pseudo random generators was the introduction of techniques based on linear recurrences. They where used as standard in the second half of the 20th century. Their quality was known to be inadequate, but there were no better methods available.

A linear congruential generator (LCG) in an algorithm used to make pseudo randomized numbers with a discontinuous piecewise linear equation. This generator is defined by the recurrence relation represented in Equation 1:

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

where:

m $0 < m-$ the "modulus"

a $0 < a < m-$ the "multiplier"

c $0 \leq c < m-$ the "increment"

X_0 $0 \leq X_0 < m-$ the "seed"

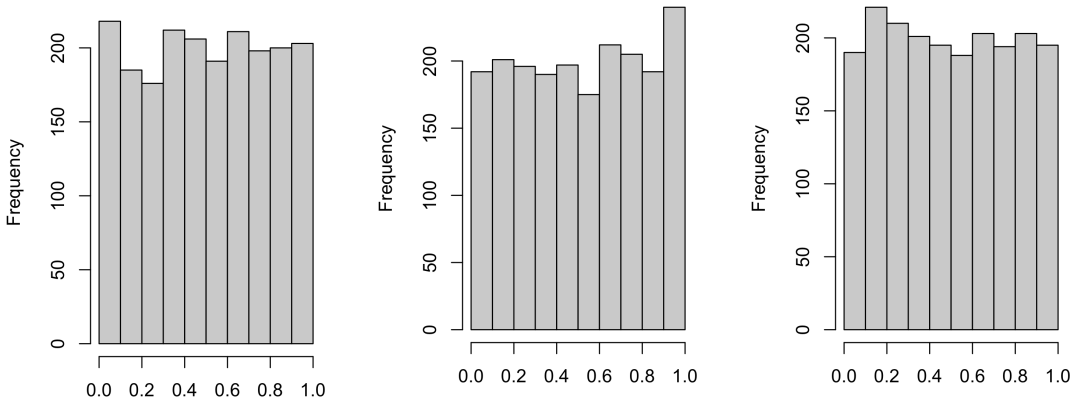
Given this generator, for this experimentation we are working with three different examples of LCG. First we have the standard generator of R for uniform random numbers `runif`. The other one is one generated by code shown in 1, where the variables `a, c,` and `m` in Equation 1 are prime numbers. The seed used is fixed to 27.

Listing 1: Code for uniform random numbers in R

```
> uniforme = function(n, semilla) {
>     a = 11551
>     c = 27077
>     m = 39709
>     datos = numeric()
>     x = semilla
>     while (length(datos) < n) {
+         x = (a * x + c) %% m
+         datos = c(datos, x)
>     }
>     return(datos / (m - 1))
>     }
```

The third generator uses the same structure as 1, but the values of `a, c,` and `m` are taken from the experimentation in the ANSI experimentation by Saucier [4], `m` $= 2^{32}$, `a` $= 1103515245$ and `c` $= 12345$. Another main difference is the quality of the seed used. In this case we set the seed using the current system time in microseconds.

On all three we perform a uniform test with a Chi - squared. In all three the `p value` gives us a correct uniformity, as shown in Figure 1.



(a) Histogram of `runif` (2000) (b) Histogram of function `uniforme`(2000, 27) (c) Histogram of function `lcg.rand`(2000)

Figure 1: Histograms showing the behaviours of each experimentation.

3 Gaussian distribution

The Gaussian distribution, also known as normal distribution, in probability theory is a type of continuous probability distribution for a real valued random variable. The form of its probability density is shown in Equation 2.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{2}$$

where:

μ is the mean or expectation of the distribution.

σ is the standard deviation.

σ^2 is the variance of the distribution.

A random variable with this Gaussian distribution is called normal deviate.

Using the standard Box – Muller transform we are experimenting using the three generators of uniform random numbers mentioned in Section 2 and moving the values of Z. If Z is a standard normal deviate then $X = Z\sigma + \mu$ will have a normal distribution [2].

Listing 2: Code for Gaussian distribution in R

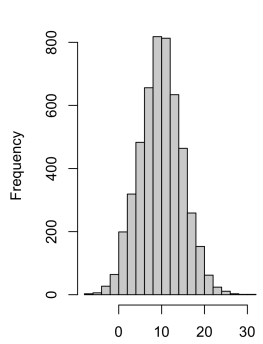
```
> gaussian = function(mu, sigma) {
>     u = runif(2);
>     z0 = sqrt(-2 * log(u[1])) * cos(2 * pi * u[2]);
>     z1 = sqrt(-2 * log(u[1])) * sin(2 * pi * u[2]);
>     datos = c(z0, z1);
>     return (sigma * datos + mu);
>  }

cat(gaussian(0, 1))
```

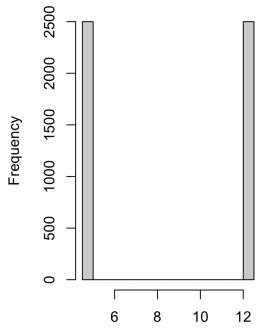
In Figure 2 we see all the distributions for all the experimentations with the different parameters using the code in Listing 2. On Figures 2a, 2b and 2c we use both parameters Z_0 and Z_1. As we can appreciate Figures 2a and 2c behave as a normal distribution. Changing the parameter used to only Z_0 made little difference in those two distributions, as we can see in Figures 2d and 2f, but in the case of Figure 2e it changes quite a bit.

In an effort to normalize whatever was happening on Figures 2b and 2e we used the two random generated numbers as separated variables `u1` and `u2` with the two Z variables. In Figure 2g the parameter for `u1` is `runif`(1) and `u2` we use the function `uniforme` (1, `u1` $*$ 1000).

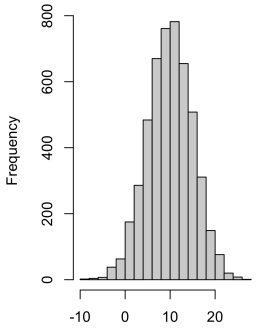
Then in Figure 2h we tried the same using now `u1` as the function `lcg.rand` (1) and `u2` `uniforme` (1, `u1` $*$ 1000). Both of this figures behave as normal distributions. Thinking that the solution to the `uniforme` function was the variables, we made a last attempt. In Figure 2i we used `u1` as the function `uniforme` (1, 27) and `u2` `uniforme` (1, `u1` $*$ 1000). This however yielded the same results as using the `uniforme` function by itself.



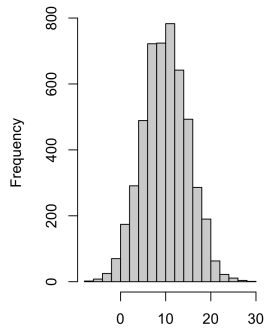
(a) With Z_0 and Z_1 using `runif`



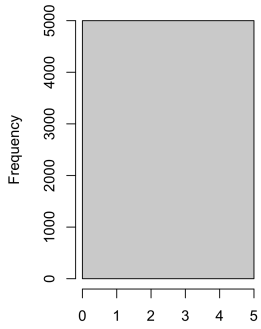
(b) With Z_0 and Z_1 using `uniforme` function



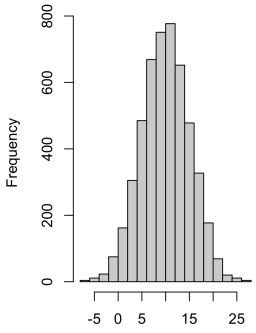
(c) With Z_0 and Z_1 using `lcg` function



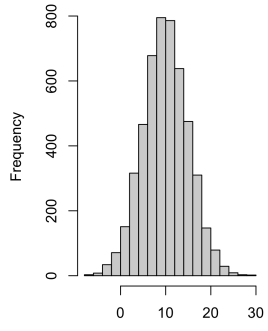
(d) With Z_0 using `runif`



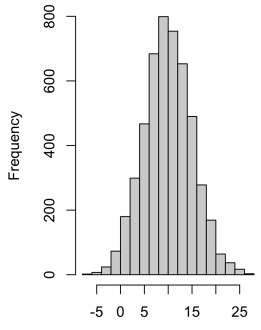
(e) With Z_0 using `uniforme` function



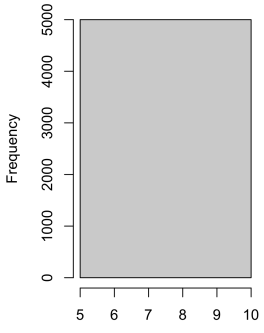
(f) With Z_0 using `lcg` function



(g) With Z_0 and Z_1 using `runif` as u1 and `uniforme` as u2

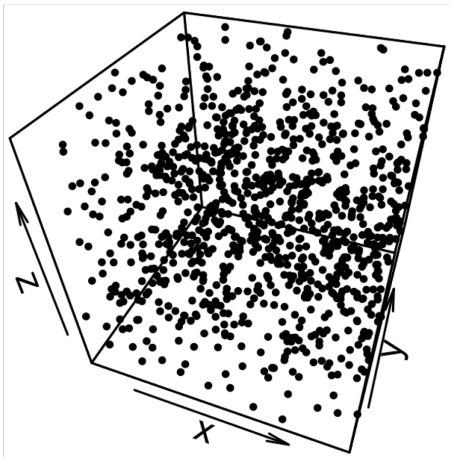


(h) With Z_0 and Z_1 using `lcg` as u1 and `uniforme` as u2

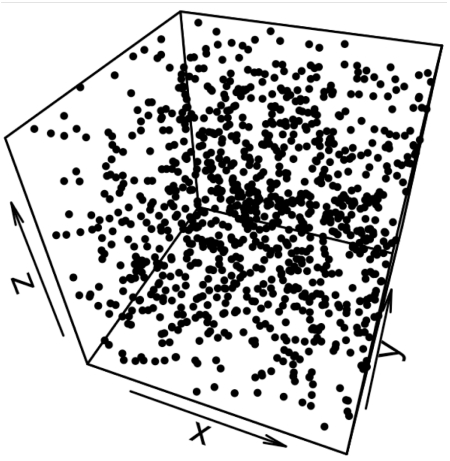


(i) With Z_0 and Z_1 using `uniforme` as u1 and `uniforme` as u2

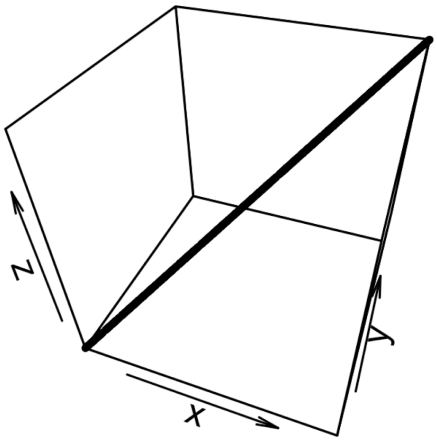
Figure 2: Histograms of all the distributions with different parameters.



(a) 3D of `runif` (2000)



(b) 3D of function `lcg.rand`(2000)



(c) 3D of function `uniforme`(2000, 27)

Figure 3: 3D depiction of each uniform random number generator

4 Other experiments

Looking for a better seed for the experimentation of the uniform distribution, we arrived to an example of a different way of seeing the randomness of each experimentation. In Figure 3 we can see in each point the behaviour of the `runif`, the `lcg` and the `uniforme` functions.

Same as with the distributions plot, in this figure we see an odd behaviour on the `uniforme` function.

5 Conclusions

It was really interesting learning more about random numbers, since we use them so much when doing experimentations. It was also compelling seeing the difference between using a good

seed and a bad one in the generation of uniform numbers, because in the first figures when we prove their uniformity there was no big sign that the results in the Gaussian experiment where going to behave that way.

There was also the expectation that in the experiments of changing the independent variables into dependent ones, that the histograms would look a lot more weird than the ones with independent variables, if only because in class there was the idea that taking away the independence of the variables would ruin the outcome of the Gaussian experiment.

References

[1] Pseudo random number generator. `https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/?ref=rp`. Accessed: 2020-10-05.

[2] Box – muller transform. `https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform`. Accessed: 2020-10-05.

[3] M. Luby. Pseudorandom number. *Pseudorandomness and Cryptographic Applications.*, page 266, 1992.

[4] Richard Saucier. *Computer Generation of Statistical Distributions.* Army Research Laboratory, 2000.