Distribución normal y uniforme

Fabiola Vázquez

5 de octubre de 2020

---

1. Introducción

En este estudio, que se realiza con el software estadístico R versión 4.0.2 [3] en un cuaderno de Jupyter [2], se trabaja con distintos algoritmos para la generación de valores con distrubución normal y uniforme.

2. Generación de variables con distribución uniforme

Uno de los métodos que existen para la generación de valores uniformemente distribuidos es el método congruencial, que sigue la siguiente relación

$$x_{n+1} \equiv ax_n + c \mod m$$

Para este método necesitamos los parámetros enteros `a`, `c`, `m` y una `semilla`.

Como un primer ejemplo, se considera un generador congruencial con `m`=8, `a`=5, `c`=4. Si en la función, que se muestra en el código 1, se hace `semilla` = 5 y se pide `uniforme(10,5)`, el generador solo nos regresa el mismo valor diez veces, es decir, la elección de parámetros no es la indicada.

Código 1: Generador congruencial.

```
uniforme = function(n, semilla) {
    a = 5
    c = 4
    m = 8

    datos = numeric()
    x = semilla

    while (length(datos)<n){
        x=(a*x + c) %% m
        datos=c(datos,x)
    }
```

```
    return(datos/(m-1))
}
```

Se debe garantizar que el ciclo del generador congruencial sea de ciclo máximo o muy grande. El siguiente teorema muestra las propiedades que deben cumplir los parámetros para que se tenga periodo máximo [1].

Teorema 1. *Las siguientes condiciones son necesarias y suficientes para que un generador congruencial con parámetros m, a y c, tengan período máximo.*

1. *c y m son primos relativos (i.e. m.c.d. (c,m)=1).*
2. *$a-1$ es múltiplo de todos los factores primos de m (i.e. $a \equiv 1 \mod g$, para todo g factor primo de m).*
3. *Si m es múltiplo de cuatro, entonces $a-1$ también lo ha de ser (i.e. $m \equiv 0 \mod 4 \Rightarrow a \equiv 1 \mod 4$).*

Con lo anterior, es fácil ver que, si tenemos a `m` como una potencia de dos, basta con que `c` y `a` sean números impares, con esto se modifican los parámetros de la función `uniforme` y se tiene como en el código 2, el cuál es un generador de ciclo máximo.

Código 2: Generador congruencial.

```
uniforme = function(n, semilla) {
    a = 16385
    c = 8191
    m = 65536

    datos = numeric()
    x = semilla

    while (length(datos)<n){
        x=(a*x + c) %% m
        datos=c(datos,x)
    }
    return(datos/(m-1))
}
```

Se llama la librería `uniftest` y se utiliza la función `sherman.unif.test` para realizar un prueba de uniformidad de Sherman a una muestra de los números que se generan. La prueba arroja un `valor p` de 0.9925 y al ser mayor que 0.05, se acepta la hipótesis de que los números se distribuyen uniformemente.

3. Generación de variables con distribución normal

Se estudian tres diferentes algoritmos que generan valores con distribución normal.

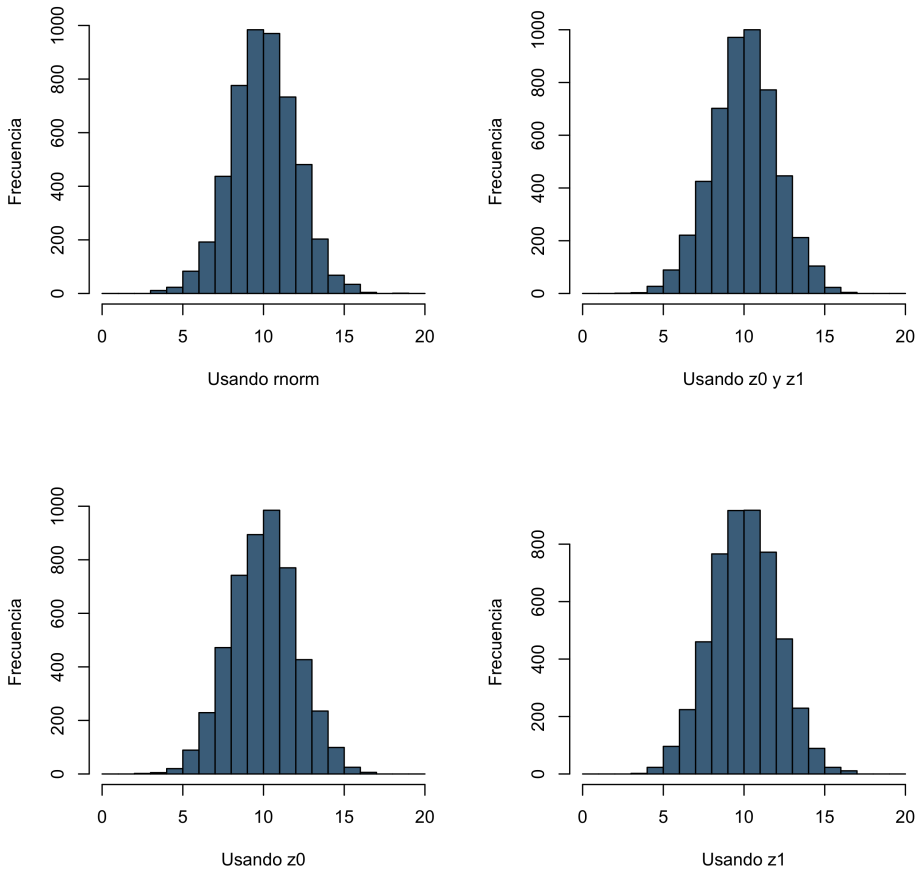

Figura 1: Histogramas de una réplica con las variaciones del algoritmo de Box-Muller.

3.1. Algoritmo Box-Müller

Este algoritmo, usa dos variables uniformemente distribuidas independientes para la generación de dos valores independientes con distribución normal [1].

Algoritmo 1 Algoritmo Box-Müller

Entrada: μ, σ
Salida: z_0, z_1 variables con distribución normal
1: Generar $u_1, u_2 \sim U(0,1)$
2: Hacer $x_1 = \sqrt{-2\ln(u_1)}\cos(2\pi u_2)$ y $x_2 = \sqrt{-2\ln(u_1)}\,\text{sen}(2\pi u_2)$
3: Devolver $z_1 = \mu + \sigma \cdot z_1$ y $z_2 = \mu + \sigma \cdot z_2$

Se realizan tres experimentos distintos, uno con ambas variables que retorna el algoritmo, y otros dos eligiendo solo una de las dos. Se quiere ver si hay alguna diferencia significativa entre los experimentos. En cada uno de estos, se generan 500 replicas de 5000 , donde a cada una de las replicas se le realiza una prueba de normalidad de Shapiro y 469 obtuvieron un `valor p` mayo de 0.05. En la figura 1 se tiene los histogramas de una de las 500 réplicas considerando las variaciones del algoritmo y se compara con los generados usando la función `rnorm`.

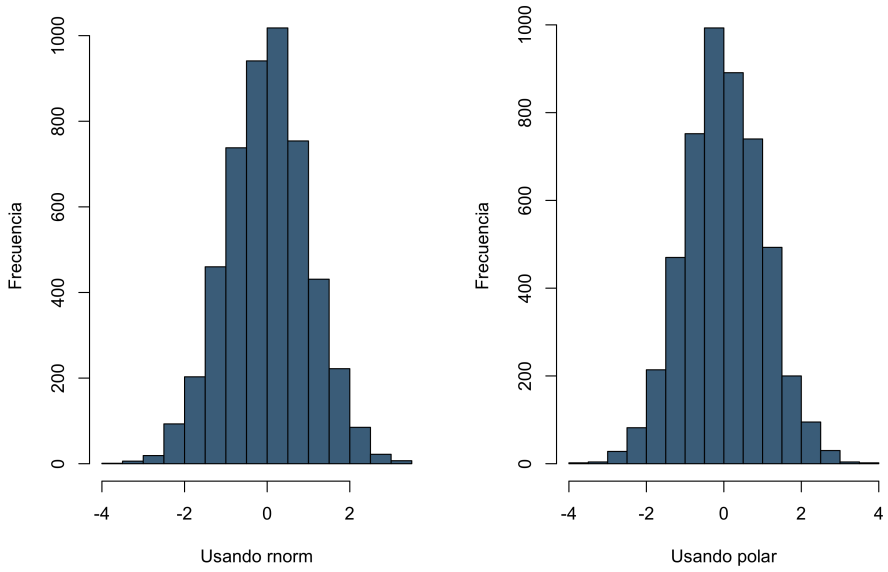

Figura 2: Comparación de una réplica usando el `algoritmo polar` con `rnorm`.

3.2. Algoritmo polar

Otro algoritmo para la generación de variables normalmente distribuidas es el *Algoritmo polar* [1].

Algoritmo 2 Algoritmo polar

Salida: z_0, z_1 variables con distribución normal

1: Generar $u_1, u_2 \sim U(0,1)$
2: Hacer $v_1 = 2u_1 - 1$, $v_2 = 2u_2 - 1$ y $w = v_1^2 + v_2^2$
3: Si $w > 1$ entonces, volver a 1
4: Hacer $y = \sqrt{\frac{-2\ln w}{w}}$
5: Devolver $x_1 = v_1 y$, $x_2 = v_2 y$

Se realiza un experimento, realizando 500 réplicas donde en cada una se generan 5000 variables aleatorias y se les somete a una prueba de normalidad de Shapiro, de las cuáles 474 pasaron la prueba.

3.3. Teorema del límite central

Este algoritmo se basa, en el *teorema del límite central* [1].

Teorema 2. *Dadas variables aleatorias $T_1, T_2, \ldots, T_n$, independientes y con distribución cualquiera, se tiene que*

$$\frac{\bar{T} - \mu_T}{\frac{\sigma_T}{\sqrt{n}}} = \frac{\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n}(T_i - E(T_1)\right)}{\sqrt{Var(T_1)}} \sim N(0,1)$$

si n es suficientemente grande.

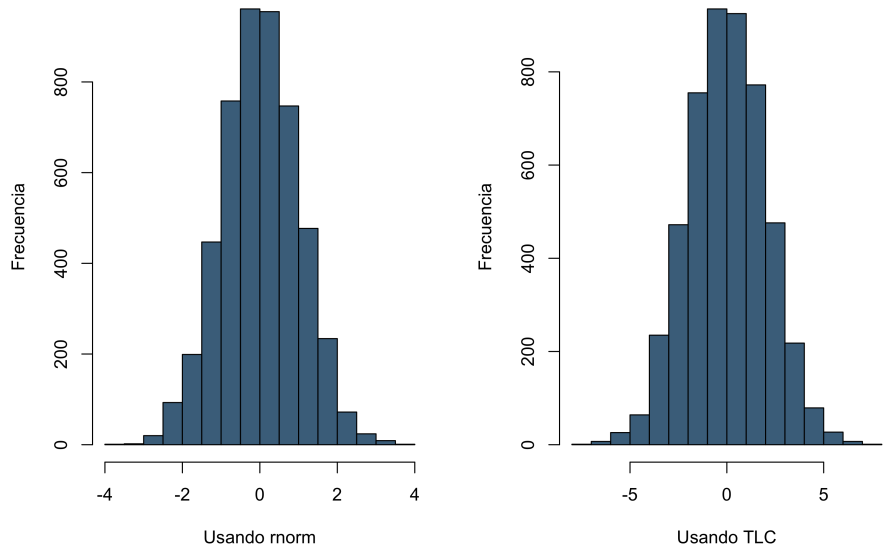

Figura 3: Comparación de una réplica usando el `algoritmo basado en TLC` con `rnorm`.

Usando una distribución que es fácil determinar, como la distribución uniforme, y $n = 50$, tenemos el *algoritmo basado en el TLC*.

Algoritmo 3 Algoritmo basado en el TLC

Salida: z_0 variable con distribución normal

1: Generar $u_1, u_2, \ldots, u_{50} \sim U(0, 1)$

2: Devolver $x = u_1 + u_2 + \ldots + u_{50} - 25$

Se realiza un experimento, realizando 500 réplicas donde en cada una se generan 5000 variables aleatorias y se les somete a una prueba de normalidad de Shapiro, de las cuáles 482 pasaron la prueba.

4. Conclusiones

En la figura 4 tenemos gráficos de caja que muestran los valores p resultantes de aplicar la prueba de Shapiro a muestras generadas por las tres variaciones del *algoritmo Box-Muller*, el *algoritmo polar* y *el basado en el TLC*, en dicha figura, la línea roja está ubicada en el `valor p` $= 0.05$. Se aprecia que para los tres algoritmos, las muestras en su mayoría pasan las pruebas de normalidad normalidad.

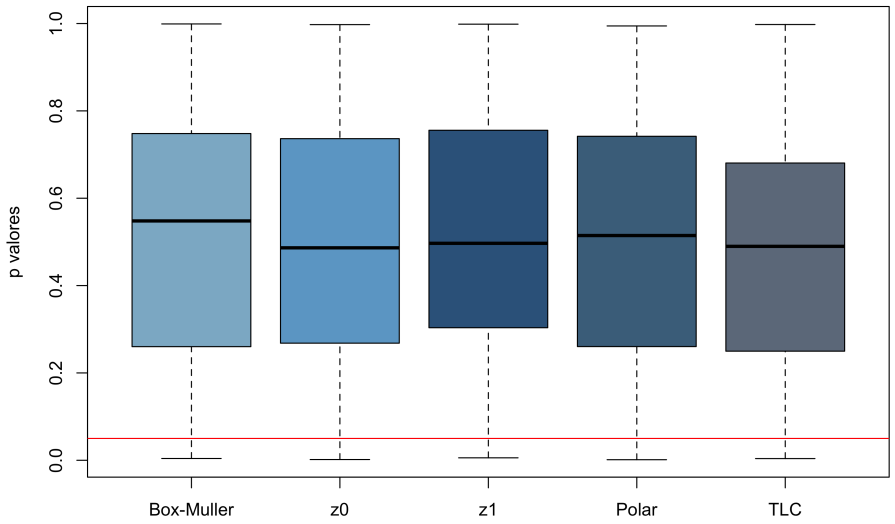

Figura 4: Gráficos de caja con los valores p obtenidos en los experimentos.

Referencias

[1] Ricardo Cao Abad. *Introducción a la simulación y a la teoría de colas*. Thomson Learning, Coruña, España, primera edición edition, 2002.

[2] Thomas Kluyver, Benjamin Ragan-Kelley, Pérez, 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*, page 87. IOS Press, 2016.

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