Práctica 3: Generación de Números Aleatorios

García Pérez Rodrigo Alan
No. de alumno: 58

Probabilidad aplicada y simulación estocástica
Profesora: Laura Clementina Eslava Fernández

02 de septiembre del 2021

Resumen

Como parte de esta práctica número tres se reviso el funcionamiento del generador lineal congruencial y se codifico una función en el lenguaje de programación R que usara este generador lineal para crear números aleatorios. Posteriormente se graficaron los diferentes valores que se obtuvieron de este generador y se analizaron los resultados.

1. Generador Lineal Congruencial

Los generadores congruentes lineales son una clase de algoritmos generadores de números pseudoaleatorios que se utilizan para generar secuencias de números similares al azar. La generación de números aleatorios juega un papel importante en muchas aplicaciones que van desde la criptografía hasta los métodos **Monte Carlo**, que utilizamos en la práctica anterior. Los generadores congruentes lineales se definen por la siguiente expresión:

$$X_{i+1} = (aX_i + c)(mod\ m)$$

Para la creación del código de la práctica a, c, m y x_0 serán constantes ingresadas por el usuario, es importante escoger estos valores adecuadamente para generar secuencias de dígitos aproximadamente aleatorios, con ciclos largos para que no se repitan con frecuencia. Como todos los generadores de números pseudoaleatorios, dado que este generador comienza con una semilla inicial y luego sigue un proceso determinista, es reproducible, lo que puede ser útil en algunos tipos de análisis.

2. Código en R

Se creó la siguiente función para generar números aleatorios. Como ya se menciono los argumentos que toma la función de parte del usuario son a, c, m y x_0 así como también una n que es el tamaño del vector de números generados.

```r
Generador_congruencial <- function(a,c,m,s,n){ #Generador congruencial si te pide ingresar una semilla (s)
  s <- (s%%m)  #Aqui nos aseguramos de poner a s en un valor adecuado
  numeros <- c(s)  #Agregamos la semilla como primer elemento del vector para poder utilizarla para X_1
  for (i in 2:n){
    numeros[i] <- ((a*numeros[i-1]+c)%%m)  #Aqui llamamos al elemento X_{i-1} para calcular X_i
  }
  numeros <- numeros/m
  return(numeros)
}
```

Figura 1: Función generador lineal congruencial

Con los siguientes parámetros mostrados se puede observar que el generador lineal congruencial produce un vector con números bastante aleatorios entre el rango de $[0, 1]$. Es importante notar que el valor un valor pequeño de m produce un vector de números se cicla muy pronto.

```
> a <- Generador_congruencial(3,4,17,3,5)
> print(a)
[1] 0.1764706 0.7647059 0.5294118 0.8235294 0.7058824
```

Figura 2: Números aleatorios generados

Ahora se puede apreciar la importancia del parámetro c pues en la siguiente imagen de la ejecución, el vector de números generados se cicla cuando $c = 0$. Si se desea obtener un vector con un ciclo más largo, se debe cumplir que $c > 0$ o de lo contrario se debe aumentar el valor de a y de m.

```
> a <- Generador_congruencial(3,1,2**4,1,8)
> print(a)
[1] 0.0625 0.2500 0.8125 0.5000 0.5625 0.7500 0.3125 0.0000
> a <- Generador_congruencial(3,0,2**4,1,8)
> print(a)
[1] 0.0625 0.1875 0.5625 0.6875 0.0625 0.1875 0.5625 0.6875
```

Figura 3: Parámetro $c = 0$

Como parte de la práctica se solicito proponer una forma para escoger la semilla de forma automática, por lo que el código mostrado anteriormente se adecuó para cumplir este

requisito. Se utilizó la función **sample ()** la cual genera un número aleatorio y esta será la semilla a utilizar.El rango de esta semilla debe estar entre $[0, m-1]$ para que funcione correctamente

```r
Generador_congruencial2 <- function(a,c,m,n){ #Generador congruencial no pide agregar una semilla
  #Fijamos una seed para obtener siempre los mismo valores
  set.seed(10)
  s <- sample(0:(m-1),1,replace=F) #Obtiene una semilla entre [0,m-1] con s entero
  s <- (s%%m)
  numeros <- c(s)    #Misma idea del codigo anterior
  for (i in 2:n){
    numeros[i] <- ((a*numeros[i-1]+c)%%m)
  }
  numeros <- numeros/m
  return(numeros)
}
```

Figura 4: Generación de números aleatorios con semilla aleatoria

3. Gráficas y Resultados

Siendo el Generador Lineal Congruencial un algoritmo utilizado para la creación de números aleatorios, el histograma que genere su vector de números debe de ser uniforme en el rango de $[0, 1]$ pues todos los valores en este rango deben de aparecer uniformemente y no debe de haber preferencia por cierto valor en particular. A continuación se muestra el histograma para 500 000 números generados por la función para valores de $a = 3^5$, $m = 3^{31} - 1$ y $c = 0$.

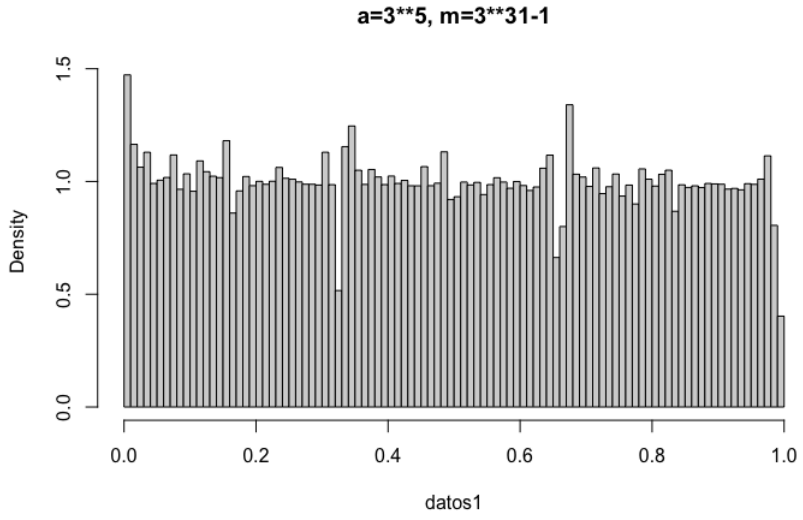

Figura 5: Histograma para 500 000 números

Del gráfico anterior podemos observar que efectivamente todos los valores en el en el rango de $[0, 1]$ sin embargo no aparecen con la misma frecuencia, pues hay mayor tendencia a aparecer de ciertos números, esto se puede se observar con las barras que tienen más altura que el promedio.

Para el siguiente gráfico se utilizaron tres valores distintos de a y tres valores distintos de m para generar 1000 valores los cuales se proyectarán en $\mathbf{R}^3$ mediante el uso de la función **scatterplot3d( )**. Podemos notar una nube de puntos bastante densa la cual demuestra que los números generados son suficientemente aleatorios.

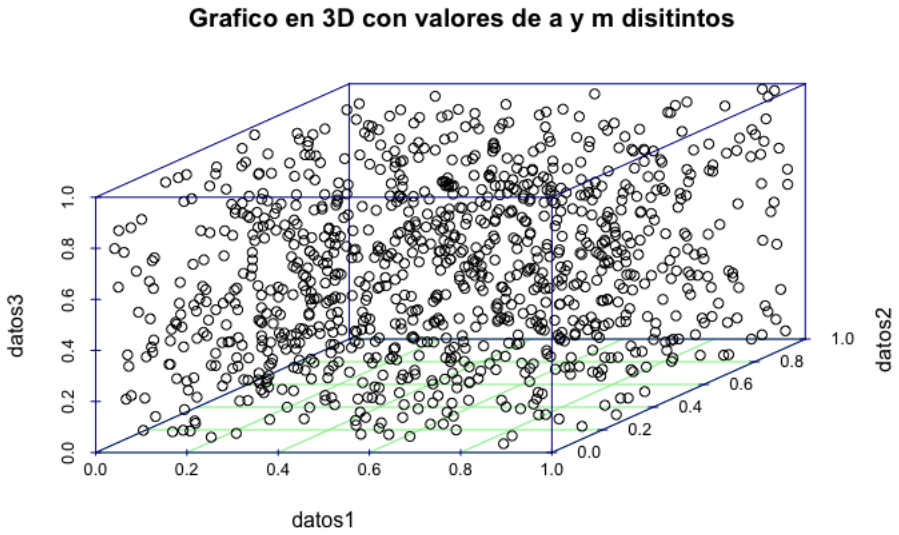

Figura 6: Gráfica para tres dimensiones

Se volvió a generar un histograma con 500 000 pero ahora con valores de $a = 7^5$, $m = 2^{31} - 1$ y $c = 0$, propuestos en la práctica. Se aprecia que este histograma es mucho más uniforme que el anterior y por lo tanto es más aleatorio porque ningún valor aparece con más frecuencia que otro.

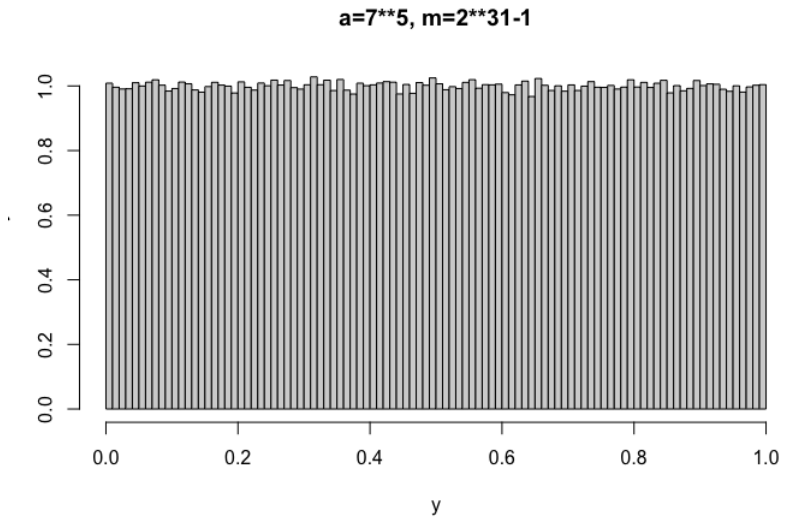

Figura 7: Histograma con parámetros propuestos en la práctica

Por último se propuso utilizar el Generador Lineal Congruencial programado en esta práctica para determinar el valor de π y comparar el tiempo que tarda con el tiempo del programa de la práctica anterior, es decir comparar los tiempos de nuestra función con la función runif( ) y determinar cual es más rápido. Para esto se ejecutaron ambas funciones para calcular π con un número de 400 000 y se midió el tiempo para cada una. El resultado de la ejecución se muestra a continuación.

```
> print(paste("Tiempo con generador lineal congruencial:",deltaTiempo))
[1] "Tiempo con generador lineal congruencial: 0.492589950561523"
> print(paste("Tiempo con runif( ) :",alfaTiempo))
[1] "Tiempo con runif( ) : 2.67504000663757"
>
```

Figura 8: Comparación de tiempos

Como podemos ver en la ejecución, el generador lineal congruencial es más rápido en su ejecución que el método runif( ) por hasta dos segundos, sin embargo runif( ) es más fácil de usar pues se puede generar un solo valor aleatorio y este será diferente cada vez y el generador lineal congruencial debe ser adaptado para que primero genere los números aleatorios y luego reparta uno por uno estos valores. Esta puede ser la razón por la que es más rápido LCG porque solo debe generar los números aleatorios una ocasión y runif( ) los genera dos veces en la función.