Generador Lineal Congruencial

Deben programar el generador lineal congruencial propuesto por Derrick Henry Lehmer en 1949 definido con la relación:

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

Para este ejercicio el valor de X_0 debe ser parte del *input* y se conoce como semilla. Por lo tanto la función debe tener como argumento a a,m,$semilla$,n donde n es el tamaño del vector de números generados.
Una vez hecho ese código, encontrar cuál es el conjunto de números que genera la sucesión en el caso $c = 0$. Es decir, que valores puede tomar la sucesión

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

Usar esta información para generar números uniformente distribuidos en el intervalo [0,1].

Luego propongan una manera automática de escoger la semilla, y con parámetros de su elección, deberán:

- Discutir cómo debería verse el histograma de n números para a y m dados. Con n grande.
- Hacer el histograma de 500 000 números con el comando *hist()* y el argumento *probability = TRUE.*
- Con tres valores distintos de a y tres valores distintos de m generar 1000 valores para cada par de parámetros y hacer una gráfica de 1000 puntos en $\mathbf{R}^3$ (pueden usar plotly: https://plotly.com/r/getting-started/)

Repetir lo anterior con $a = 7^5$, $m = 2^{31} - 1$ y comparar las gráficas.

Finalmente usar este generador para volver a estimar π y comparar los tiempos medios de ejecución para distintas n.

Para la entrega cada integrante del equipo deberá mandar el código y un archivo en PDF con: las gráficas, sus observaciones acerca sus diferencias o similitudes, tambien deben agregar las semillas que usaron para cada gráfica, además del método que propusieron para generar la semilla.

¿Cual generador de números es mejor? *runif* o el que programaron aquí.

Generador lineal congruencial

El generedor lineal congruencial es uno de varios algortimos para generar numeros pseudoaleatorios. Se basa en una relacion de recurrencia, es decir, para generar cada número aleatorio se debe utilizar el anterior y así sucesivamente.

```r
#' Genenerador lineal congruencial
#'
#' @param n es la cantidad de nmeros a generar.
#' @param semilla es nuestro x0 donde empezara nuestro algoritmo.
#' @param m es el modulo.
#' @param a es la constante multiplicativa.
#' @param c es la constante aditiva.
#'
#' @return un vector de nmeros pseudoaleatorios.
generador_congruencial <- function(n, semilla, m, a, c){
  x = c(semilla)
  for(i in 2:(n+1)){
    x[i] = (a*x[i-1]+c) %% m
  }
  return(x[2:(n+1)])
}
```

Procedemos a probar nuestra función:

```r
# Para n=3, semilla=15, mod=100, a=50, c=16
generador_congruencial(3, 15, 100, 50, 16)

generador_congruencial(10, 4, 100, 50, 16)
```

Vemos que el periodo de nuestros números aleatorios es muy pequeño ya que observamos que en el tercer numero se vuelve a repetir. Ahora trataremos de cambiar los valores de la semilla y de las constantes a y c;

```r
generador_congruencial(28, 23, 100, 32, 53)

generador_congruencial(28, 67, 100, 45, 23)
```

Ahora podemos observar que nuestra serie de número aleatorios todavía sigue teniendo un patrón con un periodo de 4, ya que cada 4 números se vuelve repetir la misma serie de números. Otra cosa que podemos ver es que el conjunto posible de números que pueden a parecer puede ser desde 0 hasta $m - 1$ debido que estamos usando el módulo para generar nuestros números aleatorios. Con estas primeras observaciones podemos concluir que nuestros números no cumplen con ser números psudoaleatorios ya que tienen una cierta estructura visible.

Ahora procederemos generar un sucesión de números aleatorios cuando $c = 0$:

```r
# Cuando c = 0
generador_congruencial(50, 23, 100, 77, 0)
```

```
3
4 generador_congruencial(50, 45, 100, 67, 0)
```

Podemos observar al igual que el anterior que sigue viéndose un patrón, pero nos percatamos que dependiendo de los valores de la semilla y la constante multiplicativa puede tener un periodo más grande o pequeño, tal que no sea tan visible encontrarlo. Dado que el conjunto de valores que puede generar la sucesión esta entre 0 a $m-1$,la serie se volverá a repetir cuando en la operación de módulo se vuelva repetir. Podemos ver hasta el momento que si queremos generar mejores números aleatorios el módulo en la operación no debe repetirse, es decir, tratar de que cada vez sea un módulo distinto. Por lo tanto, si tenemos una m muy grande tendremos una menor probabilidad de volver a repetir el mismo módulo, ya que el total de casos que puede tomar serian $m-1$:

```
1 # Pruebas para un m muy grande
2 generador_congruencial(101, 45, 1000, 67, 0)
3
4 generador_congruencial(100, 45, 10000, 67, 0)
```

Vemos que el patrón de los números suele ser cada vez más difícil de encontrar, debido que la operación del modulo puede salir un conjunto de valores más grande, también otro factor a considerar son las constantes multiplicativa y aditiva ya que son más operaciones a realizar y puede alterar el patrón para evitar repetir el módulo.

Para generar números aleatorios uniformemente distribuidos en el intervalo de [0,1] tendríamos que dividir nuestro numero aleatorio:

$$\frac{xi}{m-1} \qquad i = 0,1,2,3,\ldots,n$$

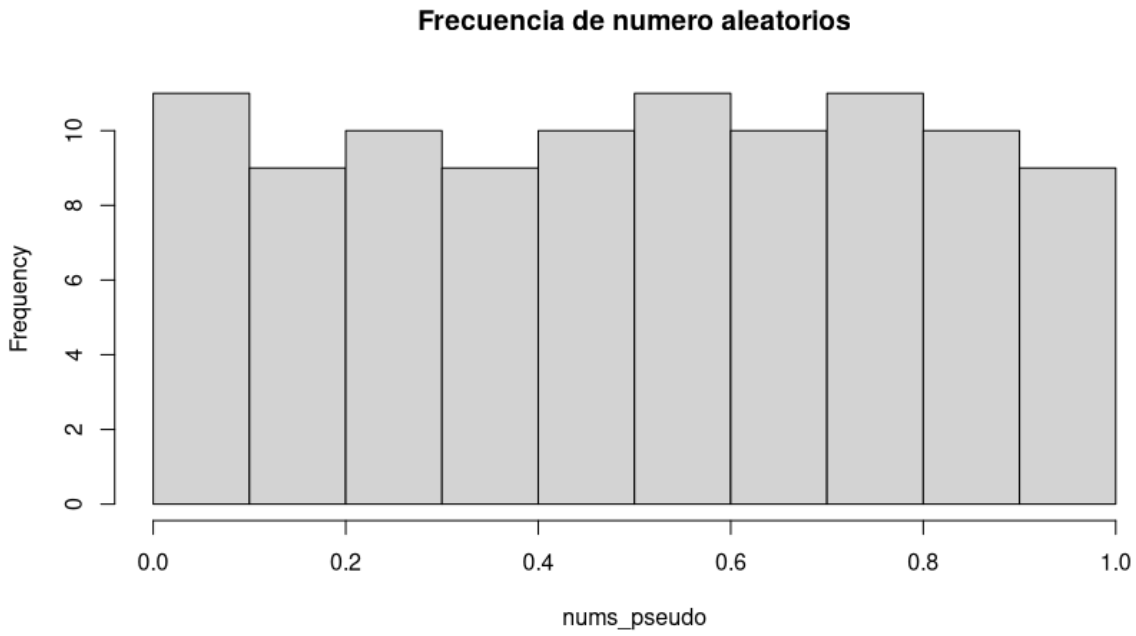
Y vamos también a proponer una semilla automática a nuestra función, la cual será ‘Sys.time()‘ que nos dará el tiempo actual, así cada vez que se llame la semilla sera diferente para caja ejecución.

```
1  #' Genenerador lineal congruencial uniforme distribuido entre [0,1]
2  #'
3  #' @param n es la cantidad de nmeros a generar.
4  #' @param m es el modulo.
5  #' @param a es la constante multiplicativa.
6  #' @param c es la constante aditiva.
7  #'
8  #' @return un vector de nmeros pseudoaleatorios.
9  generador_congruencial_u <- function(n, m = 1000, a = 32, c = 0){
10   semilla <- as.numeric(Sys.time())
11   x <-  c(semilla)
12   norm <- c(x)
13   for(i in 2:(n+1)){
14     x[i] <- (a*x[i-1]+c) %% m
15     norm[i]<- x[i] / (m-1)
16   }
17   return(norm[2:(n+1)])
```

```
}
print(generador_congruencial_u(20, 1000, 67, 0))
```

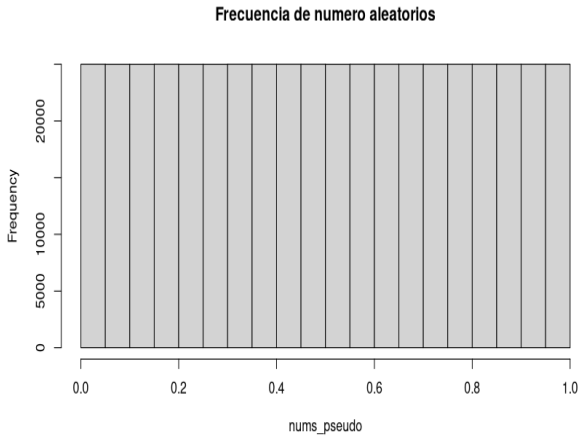
Haremos nuestro histograma con n y m muy grande, lo que podemos esperar es que tendríamos cada uno de nuestras barras con diferentes alturas ya que como se menciono anteriormente con m muy grande existe una probabilidad pequeña de que se repita el modulo y que se empiezan a repetir.

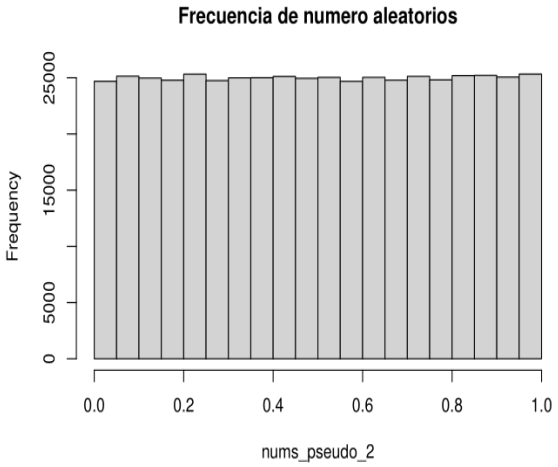
```
# Para esta prueba tenemos
# n = 100, m = 1000, a = 342
nums_pseudo = generador_congruencial_u(100, 1000, 342)
hist(nums_pseudo, main="Frecuencia de numeros aleatorios")
```



Como podemos apreciar que nuestro histograma con $n = 100$, no hay una estructura bien definida, las alturas van variando, por lo tanto, vemos que la aleatoriedad de nuestro números hasta el momento se ve bien.

```
# Prueba con n = 500000, m = 1000, a = 342
nums_pseudo = generador_congruencial_u(500000, 1000, 342)
hist(nums_pseudo, main="Frecuencia de numero aleatorios")

# n = 500000, m = 2^31 - 1, a = 7^5
nums_pseudo_2 = generador_congruencial_u(500000, 2^31 - 1, 7^5)
hist(nums_pseudo_2, main="Frecuencia de numero aleatorios")
```



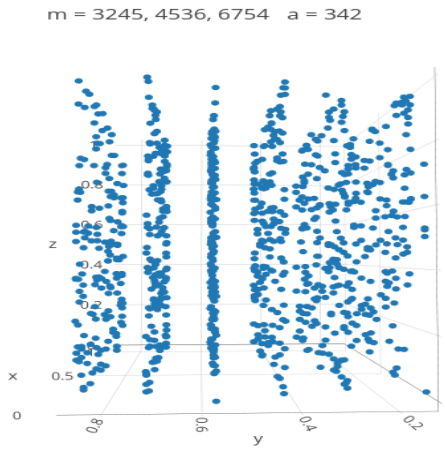


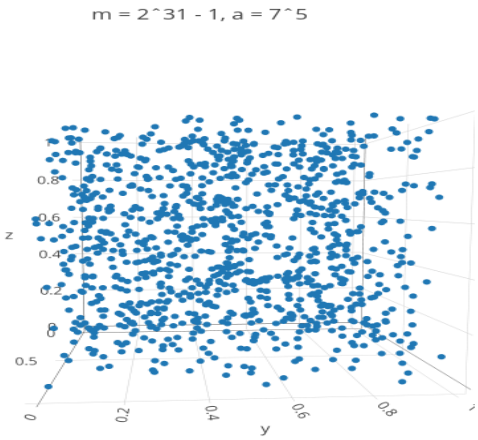
Ahora haciendo nuestro experimento con valores más grandes de m y a, podemos ver que el comportamiento cambia ligeramente, en el lado izquierdo vemos como el histograma tiene una cierta forma, dado que se ve como un rectángulo grande bien formado, es decir, cada número tiene la misma frecuencia de salir, por otro lado, vemos que el histograma de lado derecho no se ve uniforme, sino más bien la altura de cada rectángulo varia ligeramente.

```
# Prueba con n = 500000, m = 1000, a = 342
r1 <- generador_congruencial_u(1000, 3245, 234)
r2 <- generador_congruencial_u(1000, 4536, 432)
r3 <- generador_congruencial_u(1000, 6754, 543)
# plot_ly(x=r1,y=r2,z=r3,type = "scatter3d", marker = list(size = 2))

# n = 500000, m = 2^31 - 1, a = 7^5
r1_1 <- generador_congruencial_u(1000, 2^31-1, 7^5)
r2_2 <- generador_congruencial_u(1000, 2^31-1, 7^5)
r3_3 <- generador_congruencial_u(1000, 2^31-1, 7^5)
# plot_ly(x=r1_1,y=r2_2,z=r3_3,type = "scatter3d", marker = list(size = 2))
```

Ahora graficando en R^3 se ve más claramente que el conjunto de números del lado izquierdo tiene una estructura o forma, ya que se pueden apreciar lineas verticales por cada número aleatorio, por lo tanto, podemos decir que no tenemos un buen generador de número aleatorios con los parámetros que propusimos,ya que esta formando un patrón visible y que se puede predecir,en cambio, en el lado derecho la gráfica no se ve como una estructura visible o un patrón que se pueda predecir, por consiguiente podemos afirmar que cuando m y a toman valores extremadamente grandes podemos tener buen generador de números pseudoaleatorios.

Ahora usaremos nuestro generador para calcular π con el experimento de gotas de lluvia. Dado que el código será el mismo de la practica 2 solo cambiaría el generador por





el nuestro 'generador$_c ongruencial_u$'conm=$2^{31}-1$ y $a = 7^5$, omitiré esa parte de código y pondré sólo lo que nos interesa. Primeramente, el tiempo de ejecución para calcular π de cada uno de los generadores.

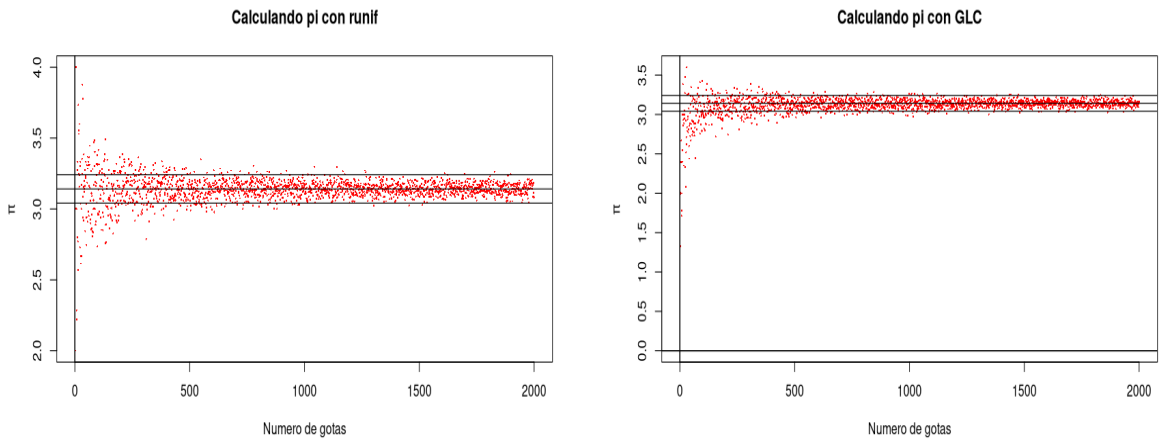
```
# Generador runif
gotas_dentro1 <- function(n){
  x <- runif(n, -1, 1)
  y <- runif(n, -1, 1)
  contador_dentro <- 0
  for(i in 1:n){
    if(x[i]^2+y[i]^2 <= 1){
      contador_dentro = contador_dentro + 1
    }
  }
  return(contador_dentro)
}

# Generador lineal congruencial
gotas_dentro2 <- function(n){
  x <- generador_congruencial_u(n, 2^31-1, 7^5)
  y <- generador_congruencial_u(n, 2^31-1, 7^5)
  contador_dentro <- 0
  for(i in 1:n){
    if(x[i]^2+y[i]^2 <= 1){
      contador_dentro = contador_dentro + 1
    }
  }
  return(contador_dentro)
}


simulacion_lluvia <- function(num_gotas, f, msg="", graficar=FALSE) {
  x <- 1:num_gotas
  aprox_pi <- c()
  for(i in 1:num_gotas){
    aprox_pi[i]= (4*f(i))/i
```

```
33    }
34    if (graficar) {
35      plot(x , aprox_pi,
36           main = paste("Calculando pi",msg),
37           ylab = expression(pi),
38           xlab = "Numero de gotas",
39           pch=16, cex=.3, col="red")
40      abline(0,0,pi-0.1,0)
41      abline(0,0,pi,0)
42      abline(0,0,pi+0.1,0)
43
44    }
45  }
```

```
1  time_fun <- function(num_gotas, f, f2) {
2    t_ini = as.numeric(Sys.time())
3    f(num_gotas, f2)
4    return(as.numeric(Sys.time()) - t_ini)
5  }
6  paste("Tiempo usando runif: ", time_fun(2000, simulacion_lluvia, gotas_dentro1))
7  paste("Tiempo usando GLC: ", time_fun(2000, simulacion_lluvia, gotas_dentro2))
```

Podemos observar claramente que el tiempo de ejecuación de nuestro generador es mucho mayor que el implementado por R, puede deberse por el tipo de operaciones que realiza nuestro generador, tanto haciendo el módulo como la multiplicación, además de los valores de m y a que deben tomar valores muy grandes para generar números realmente pseudoaleatorios, como consecuencia las operaciones van hacer más tardadas.

```
1  simulacion_lluvia(2000, gotas_dentro1, "con runif", graficar = TRUE)
2  simulacion_lluvia(2000, gotas_dentro2, "con GLC", graficar = TRUE)
```



Vemos que en tanto la gráfica los dos métodos se aproximan π, la única diferencia con el método 'runif' son los puntos que están más dispersos, a comparación del nuestro, debe ser por la manera en que esta implementada, podemos conjeturar que sus números son más aleatorios que los nuestro ya que si vemos la gráfica para cada valor de *pi* esta tanto cerca como lejos del valor real, en cambio, con el nuestro suele estar un poco más cercano a las dos linea de error que dibujamos, dándome una intuición de que en el experimento existen casos que las gotas están cayendo más dentro del circulo y por lo tanto hay cierto sesgo o tendencia a salir números más pequeños que valores grandes.

Conlusión

Pudimos entender que la generación de números aleatorias depende de los parámetros para generarlos, siendo un factor muy importante. Si no se eligen adecuadamente podrían encotrarse un patrón y poder predecirlos. Vimos que si se gráfican los números aleatorios de un generador y encontramos una estructura o forma en la gráfica, indica que no es un buen generador, por otro lado, cuando los puntos de una gráfica no tienen una forma o estructura y es difícil encontrar un patrón, quiere decir que nuestro generador si genera números pseudoaleatorios.