Práctica 2: Calculando Pi con gotas de lluvia

García Pérez Rodrigo Alan
No. de alumno: 58
Probabilidad aplicada y simulación estocástica
Profesora: Laura Clementina Eslava Fernández
02 de septiembre del 2021

Resumen

Para esta segunda práctica se revisó el Método Montecarlo usado para la estimación de áreas y se aplicó para encontrar el valor de π, se realizó un código en lenguaje de programación R y se creó una gráfica que muestra los diferentes valores de π a los que se llego a través de este método.

1. Método Montecarlo

La simulación de Monte Carlo se utiliza principalmente para predecir la consecuencia final de una serie de sucesos, cada uno con su propia probabilidad. Un requisito importante en la simulación de Monte Carlo es que el evento o fenómeno se describa mediante funciones de densidad de probabilidad. Una vez que se conocen las funciones de densidad, la simulación de Monte Carlo puede proceder mediante un muestreo aleatorio de estas funciones de densidad para su uso dentro de la simulación.

En este caso el muestreo aleatorio se pensará de la siguiente manera: se dibuja un cuadrilátero, y dentro de él un círculo. Una vez dibujado, lo colocamos bajo la lluvia de modo que le caiga una buena cantidad de gotas. Como las gotas de lluvia se reparten al azar sobre la superficie del cuadrado, es de esperar que la probabilidad de que una gota caiga dentro del círculo sea proporcional al área del mismo, y que la probabilidad de que caiga en el cuadrado sea, también, proporcional al área del a cuadrado, por lo que se podrá despejar a π de la siguiente forma:

$$\pi \approx 4\frac{Gotas_{círculo}}{Gotas_{cuadrado}}$$

2. Código en R

Toda la idea de las gotas de lluvia y la cartulina con el circulo se abstrae en el siguiente código de R.

```r
calcularPi<-function(n){

  gotas<-n #Variable para  el número de gotas que van a caer
  gotas_cirulo<- 0 #Contador para gotas que caen dentro del círculo
  gotas_cartulina<-0 #Contador para gotas que caen fuera del círculo

  for (i in 1:n){

    #Generamos números aleatorios con runif
    #Estos se encontrarán entre el rango de -1 a 1 en x,y
    x <-runif(1, min=-1, max=1)
    y<-runif(1, min=-1, max=1)

    #Obtenemos la distancia al origen del vector x,y
    distancia_origen<- (x**2) + (y**2)

    #Si la distancia es menor o igual a 1 significa que
    #está dentro del circulo y el contador aumenta
    if (distancia_origen <= 1){
      gotas_cirulo <- gotas_cirulo +1
    }

    gotas_cartulina <- gotas_cartulina +1
  }
  gotas_cirulo
  gotas_cartulina

  #Se obtiene finalmente el valor de pi
  valorpi <- 4*gotas_cirulo/ gotas_cartulina
  return(valorpi)
```

Figura 1: Función para calcular valor π

En la función anterior se recibe como un parámetro a n, que es la variable que recibirá el número de gotas a contar y para cada gota expresada como el vector (x, y) se obtendrá su distancia al origen para poder decidir si la gota cayó dentro del círculo de $radio = 1$. Por último se aplica la fórmula expresada anteriormente y obtenemos el valor de π para ese número de gotas. A continuación se necesita ejecutar esta función en múltiples ocasiones para valores diferentes de n y así se podrán graficar los diferentes resultados obtenidos. Para esto se creó un ciclo for que aumenta el valor de n progresivamente y almacena el valor de π encontrado en un vector. A continuación se muestra esta parte del código.

```r
  #Se obtendrá pi para valores de n desde 1 a 20000
  for (j in 1:20000){
    z<-calcularPi(j)
    vector_y<-c(vector_y,z)
  }
```

Figura 2: Ciclo para encontrar π multiples veces

Es importante mencionar que para reproducir la aleatoriedad de las gotas de lluvia cayendo, se tuvo que encontrar una función que devuelva números realmente aleatorios. Para esto se utilizó la función **runif( )** la cual permite obtener n observaciones aleatorias de una distribución uniforme. Los argumentos de la función se describen a continuación:

runif syntax

```
runif(n        # Número de observaciones a ser generadas
      min = 0, # Límite inferior de la distribución (a)
      max = 0) # Límite superior de la distribución (b)
```

Figura 3: Función para generar números aleatorios

Sin embargo, cada vez que se ejecute el código anterior se obtienen números distintos. Para que el resultado sea consistente se utiliza una semilla para generar los números con la función **set.seed( )**

```
> set.seed(10)
> runif(3, min=-1, max=1)
[1]  0.01495641 -0.38646299 -0.14618467
> set.seed(10)
> runif(3, min=-1, max=1)
[1]  0.01495641 -0.38646299 -0.14618467
```

Figura 4: Generación de números aleatorios con $semilla = 10$

3. Resultados y Gráfica

Ahora se muestran los diferentes resultados que devuelve el programa para distintos valores de n. Se puede notar como el valor de π se va haciendo cada vez más exacto conforme aumentan el número de gotas contabilizadas.

```
> set.seed(10)
> print(calcularPi(10))
[1] 3.6
> print(calcularPi(100))
[1] 3.24
> print(calcularPi(1000))
[1] 3.132
> print(calcularPi(10000))
[1] 3.1504
> print(calcularPi(100000))
[1] 3.1448
```

Figura 5: Resultados del programa

Por último, creamos la gráfica de número de gotas por valor de π, calculado para valores de de 1 hasta 3000 gotas. Se eligió este número porque apartir de las 1500 gotas el valor de π ya no varia significativamente.

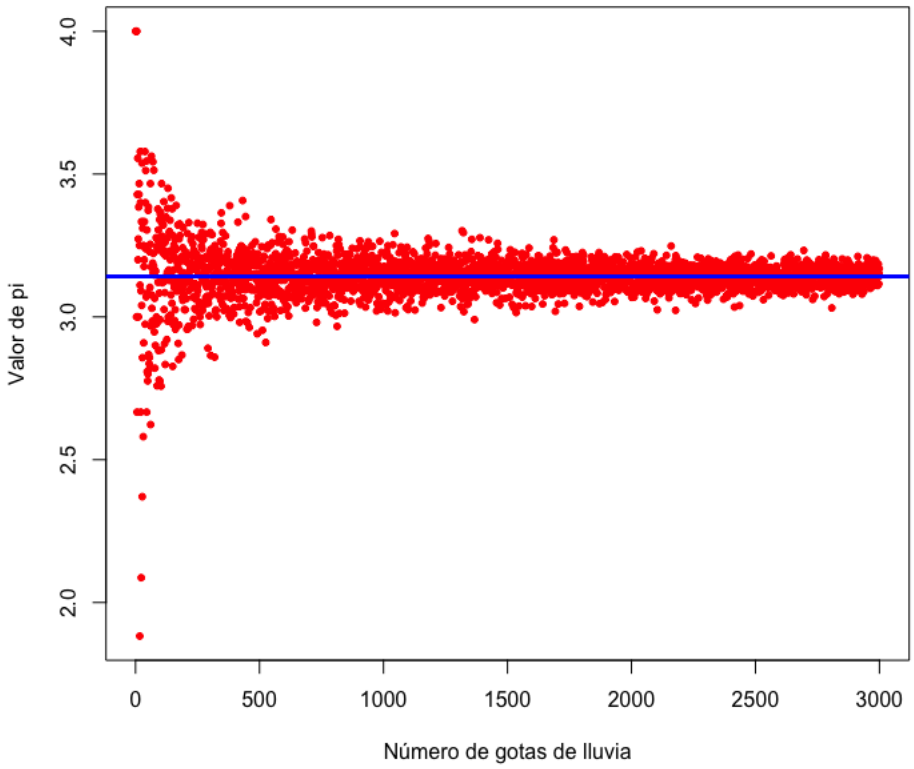

Figura 6: Gráfica π por *gotas*

De la gráfica anterior se puede observar que para los valores más pequeños de número de gotas, el valor de π varia por hasta ± 2 unidades y conforme n aumenta, su resultado se va haciendo cada vez más constante. La línea de color azul es el valor real que se tomo como $\pi = 3{,}1415926535$. Se necesitaría llegar hasta $n = 1,000,000$ para acertar apenas con tres decimales a π, la gráfica quedaría muy grande y el cambio sería imperceptible

```
> print(calcularPi(1000000))
[1] 3.14176
```

Figura 7: Valor de π para un millón de gotas