Método Monte-Carlo

Edson Raúl Cepeda Márquez

26 de febrero de 2019

1. Objetivo

El objetivo principal de esta práctica es examinar y analizar los resultados de aplicar el método Monte-Carlo para estimar el resultado de una integral definida. Se compara los resultados obtenidos con el método Monte-Carlo y Wolfram Alpha con el fin de encontrar el tamaño de muestra más conveniente para que los resultados sean lo más parecido posible. Esto en medida de cuantos decimales acierta el método. Se gráfica el resultado para así visualizar como afecta el tamaño de la muestra en la cantidad de decimales acertados.

Se hace uso del material de apoyo disponible en la pagina [2] de la Dra. Elisa Schaeffer, también se hace uso del codigo para el calculo de la integral usando el método Monte-Carlo escrito en el lenguaje de programación R [1].

2. Desarrollo

Se empieza por seleccionar las partes más importantes del código modificarlas para cumplir el objetivo. Establecemos un numero de réplicas del experimento y una variación en el tamaño de la muestra que estaremos probando en cada réplica.

```
replicas <- 10 #Replicas del experimento

for(i in 1:replicas) {
  for(j in c(1000,2500,5000,10000)){ #Variacion de las muestras
    montecarlo <- foreach(i = 1:j, .combine=c) %dopar% parte()
    stopImplicitCluster()
    integral <- sum(montecarlo) / (j * pedazo)
    resultado <- (pi / 2) * integral
    datos <- c(datos,resultado)

  }

}
```

Dentro de ella se ejecutara la función que calcula la integral. Los resultados se guardaran en el vector *datos*

Procedemos a establecer el valor esperado para su posterior comparación con los resultados.

```
resultadoEsp <- (pi/2) * 0.031089 #Se establece el resultado esperado
resultadoEsp_str <- strsplit(paste(resultadoEsp),split = "") #String del reusltado esperado
datos <- c()
datos_str <- c()
```

Se usa la comparación de cadenas de caracteres para comprobar la precisión de las decimales. Posteriormente se convierten los resultados a cadenas de caracteres para comparar y se guarda en el vector *resultadosstr*.

```
resultados <- c()
resultados_str <- c()
res_contadores <- c()

for(i in 1:length(data.matrix(datos))) {
  resultados <- c(resultados,data.matrix(datos)[i])
  resultados_str <- c(resultados_str, strsplit(paste(resultados[i]), split=""))
}
```

Ahora se calcula la precisión de las decimales.

```
for(i in 1:length(resultados)) {
  contador <- 0
  for(j in 1:7) {
    b <- (unlist(resultados_str[i])[j]) == (unlist(resultadoEsp_str)[j])
    if(b) {
      contador <- contador + 1
    } else {
      break;
    }

  }

  res_contadores <- c(res_contadores,contador-2)
}
```

.

Se compara los decimales y se incrementa la variable *contador* en uno cada vez que los resultados aciertan en una decimal. El ciclo se detiene cuando no acierta pues deja de ser preciso en las siguientes decimales.

3. Resultados y conclusiones

Ahora es posible visualizar los resultados.

Antes de dar paso a la conclusión es necesario recalcar que el calculo de la integral en este caso, es solo una aplicación del método Monte-Carlo.

Como podemos observar en la figura 1 la precisión de los decimales se ve directamente afecta por el tamaño de muestra, es decir, el numero de valores que se generen. Se alcanza los siete decimales de precisión en las cuatro de las variaciones del tamaño de muestra y tiene una precisión promedio de cinco decimales de exactitud.

Comparando que tan cercanos son los valores obtenidos en el cálculo y con respecto al valor esperado, podemos observar que en promedio los datos se acercaban considerablemente al valor esperado y la menor cantidad de los resultados se alejaban del valor esperado.

Se debe de establecer tamaños de muestra optimo para tener una buena precisión en los cálculos hechos por el método Monte-Carlo. También es una buena práctica replicar el calculo una cantidad de veces considerable siempre y cuando sé este trabajando con valores aleatorios.

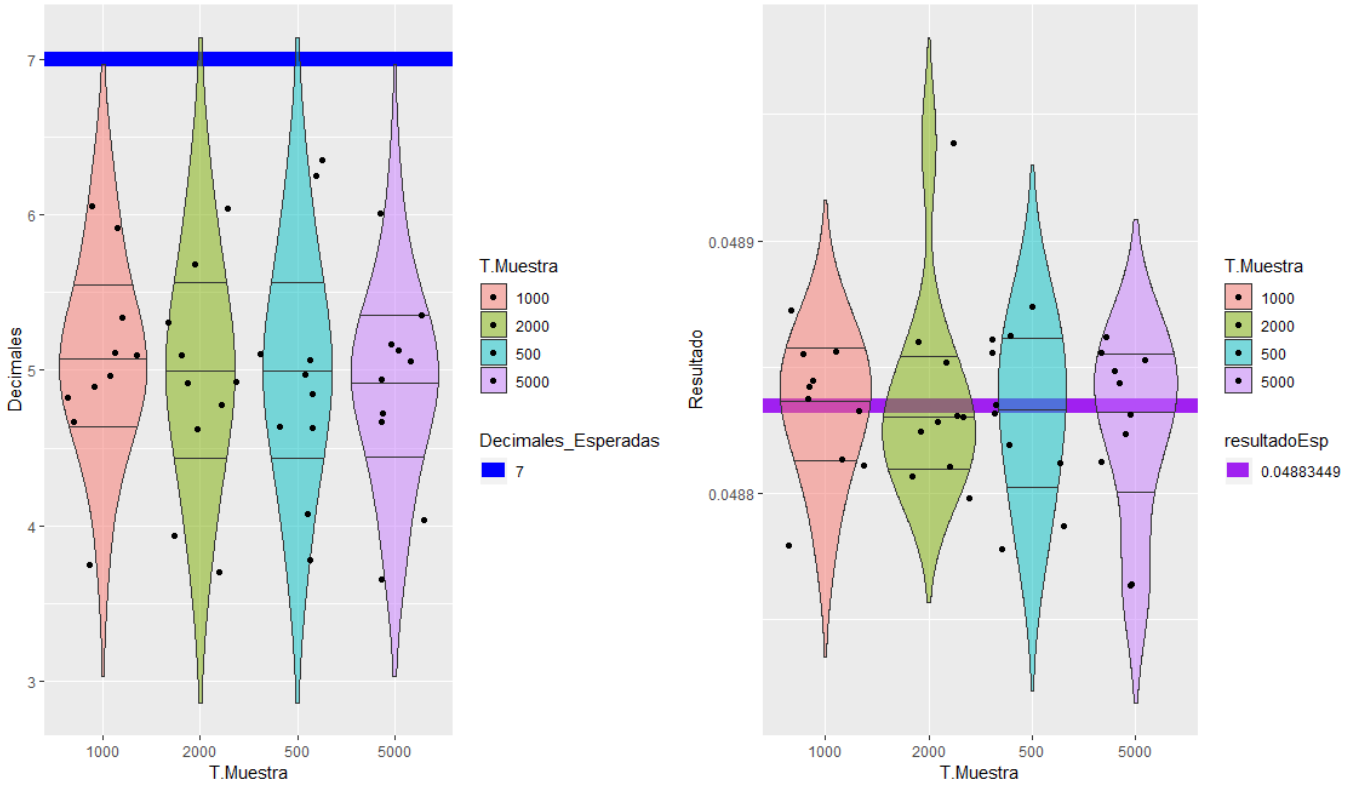

Figura 1: Comparación del tamaño de muestra dependiendo del valor esperado y la cantidad de decimales acertadas en el cálculo de la integral.

4. Primer reto

El primer reto consiste en diseñar un experimento para estimar el valor de π usando el paralelismo, también se determina la relación matemática entre el número de muestras obtenidas y la precisión de los decimales de los resultados obtenidos.

Se supone un cuadrado el cual contiene un círculo de radio r en su interior. De esta manera se puede afirmar que el radio del círculo (r) sera la mitad de los lados del cuadrado, por lo cual los lados del cuadrado sera dos veces el radio del círculo $(2r)$. Para determinar que área del cuadrado pertenece al círculo se establece la relación en la ecuación 1.

$$R = \frac{\pi r^2}{4r^2} \tag{1}$$

En donde R es la relación del área del cuadrado y el área del círculo y es el radio , podemos cancelar y despejar π para de esta manera obtener la ecuación 2 que estaremos utilizando para estimar el valor de π.

$$\pi = 4 * R \tag{2}$$

Ahora es posible estimar π a partir de esta relación.

```
1 calculo <- function(generar) {
```

```
  r <- 1 #Radio
  contcirc <- 0 #Contador dentro del circulo
  contcuad <- 0 #Contador fuera del circulo
  num <- generar #Muestra
  for(i in 1:num){
    x <- runif(1) #valor x dentro del cuadrado
    y <- runif(1) #valor y dentro del cuadrado
    d <- (x*x) + (y*y) #Calcular distancia
    if (d < r*r) { #No sale del circulo
      contcirc <- contcirc + 1
    }
  }

  pi <- as.double(4 * (contcirc/num)) #Relacion
  return(pi)
}
```

Se establece un valor para el radio y se generan números aleatorios entre cero y uno para coordenadas imaginarias de X y Y dentro del cuadrado. Se procede a generar el numero de valores aleatorios especificado en la variable *num*, esto equivale al tamaño de la muestra. Se calcula la distancia pitagórica para así aislar los casos en los que los valores aleatorios estén fuera del área del círculo. Si un valor coincide dentro del área del círculo entonces la variable *contcirc* incrementa en uno.

Se calcula π con la ecuación 1. Se toma R como la relación entre el número de veces que los valores aleatorios cayeron dentro del círculo y el total de valores en la muestra. Se hacen diez réplicas del experimento y se usa una técnica de comparación de cadena de caracteres para encontrar la precisión de decimales de los resultados.

```

replicas <- 10 #Numero de replicas del experimento
datos <- data.frame()
variacion <- c(1000,5000,10000) #Variacion de las muestras
for(var in variacion) {
  calcularPi <- foreach(i=1:replicas, .combine = c) %dopar% calculo(var)
  datos <- rbind(datos,calcularPi)
}

resultadoEsp <- 3.1415926 #Resultado esperado
resultadoEsp_str <- strsplit(paste(resultadoEsp),split = "") #Se convierte a cadena de caracteres
resultados <- c()
resultados_str <- c()
res_contadores <- c()

for(i in 1:length(data.matrix(datos))) {
  resultados <- c(resultados,data.matrix(datos)[i])
  resultados_str <- c(resultados_str, strsplit(paste(resultados[i]), split=""))
}

for(i in 1:length(resultados)) { #Se comparan los decimales acertados
  contador <- 0
  for(j in 1:7) {
    b <- unlist(resultados_str[i])[j] == unlist(resultadoEsp_str)[j]
    if(b) {
      contador <- contador + 1
    }
    else {
      break
    }

  }

  res_contadores <- c(res_contadores,contador)
```

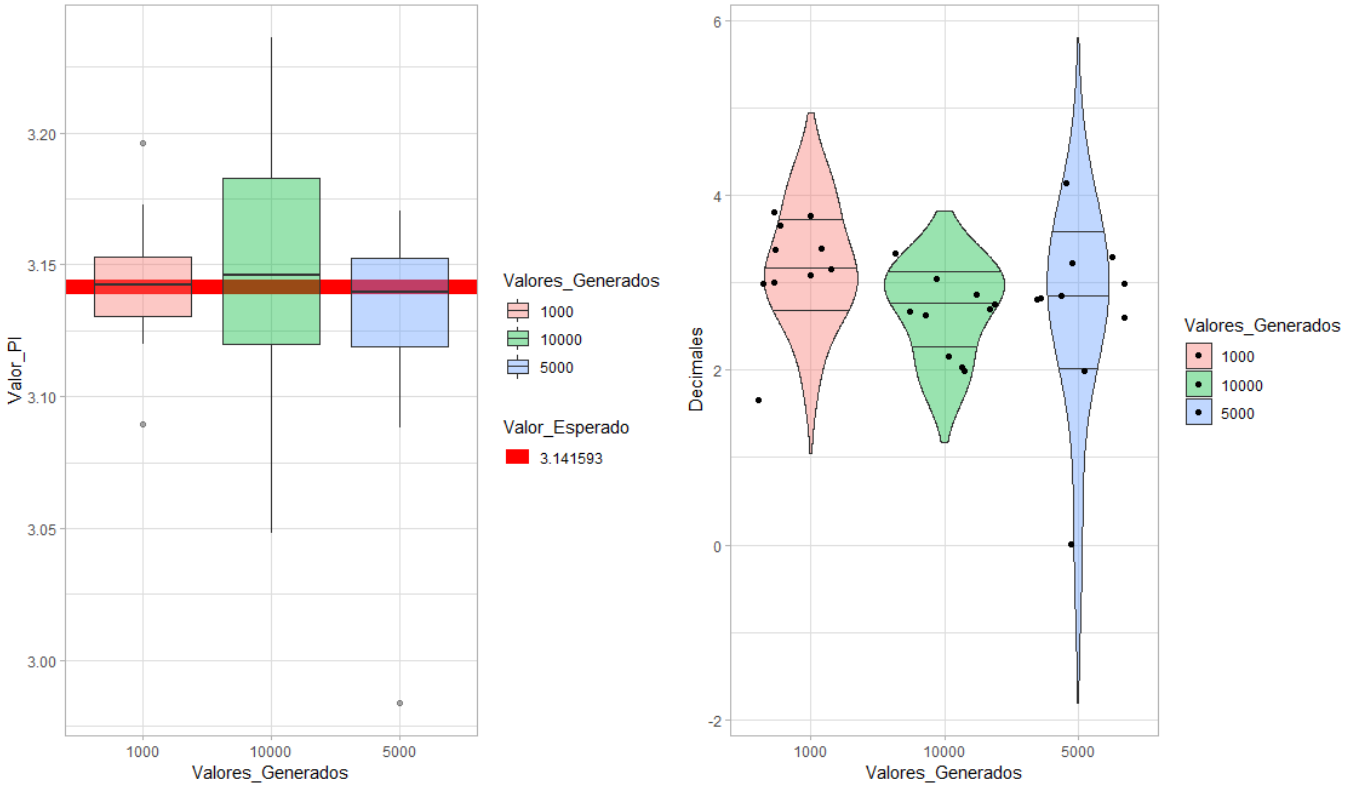

Figura 2: Comparación del tamaño de muestra dependiendo del valor esperado y la cantidad de decimales acertadas en la estimación de π.

Como se puede observar en la figura 2 los valores seleccionados son adecuados para estimar el valor de π. Esto se comprueba viendo que los valores promedio de todas las replicas y el tamaño de muestra, es decir, los valores generados son muy cercanos al valor esperado de π.Así mismo estos resultados alcanzan hasta 6 decimales de precisión a la hora de estimar el valor de π.

En la mayoría de los casos usando este método se necesita probar con distintos tamaños de muestra, no siempre usar un tamaño de muestra elevado asegura más precisión en el cálculo.

Referencias

[1] R: R Project, 2019
`https://www.r-project.org/`

[2] Satu Elisa Schaeffer: Práctica 5: Método Monte-Carlo, 2019
`https://elisa.dyndns-web.com/teaching/comp/par/p5.html`

[3] TheCodingTrain: Coding Challenge 95, Approximating the Value of PI, 2018
`https://www.youtube.com/watch?v=5cNnf7e92Ql`