Reporte práctica 5

Método Monte-Carlo

A 12 de Septiembre de 2017

Método Monte-Carlo

El método Monte-Carlo es una herramienta de la estadística que se utiliza para la generación de muestras aleatorias de una distribución conocida. La importancia de este método se debe a la existencia de problemas que no se pueden resolver por métodos analíticos o numéricos, pero que dependen de factores aleatorios o se pueden asociar a modelos probabilísticos.[1]

Práctica

En la práctica sobre método Monte-Carlo se examinó el efecto del tamaño de una muestra en la precisión de un estimado, y se comparó con el resultado obtenido con Wolfram Alpha. Este efecto se examinó con el resultado de la integral de $\int_3^7 \boldsymbol{f(x)dx}$, donde la función ***f(x)*** es igual a

$$\boldsymbol{f(x)} = \frac{\mathbf{1}}{\boldsymbol{exp(x)+exp(-x)}}.$$

La función de ***f(x)*** tiene una función de distribución válida ***2f(x)/π***, lo que nos permite la generación de números pseudoaleatorios con la distribución ***g(x)=2f(x)/π***, y así estimar $\int_3^7 \boldsymbol{g(x)dx}$.

- Código ejemplo

```
inicio <- -6
final <- -inicio
paso <- 0.25
x <- seq(inicio, final, paso)
f <- function(x) { return(1 / (exp(x) + exp(-x))) }
png("p5f.png") # dibujamos f(x) para ver como es
plot(x,  (2/pi) * (1/(exp(x)+exp(-x))))
lines(x,  (2/pi) * (1/(exp(x)+exp(-x))), type="l")
graphics.off()
suppressMessages(library(distr))
g <- function(x) { return((2 / pi) * f(x)) }
generador  <- r(AbscontDistribution(d = g)) # creamos un generador
muestra <- generador(50000) # sacamos una muestra
png("p5m.png") # validamos con un dibujo
hist(muestra, freq=F, breaks=50,
    main="Histograma de g(x) comparado con g(x)",
    xlim=c(inicio, final), ylim=c(0, 0.4))
lines(x, g(x), col="red") # dibujamos g(x) encima del histograma
graphics.off()
desde <- 3
```

```
hasta <- 7
pedazo <- 50000
cuantos <- 500
parte <- function() {
   valores <- generador(pedazo)
   return(sum(valores >= desde & valores <= hasta))
}
suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectCores() - 1))
montecarlo <- foreach(i = 1:cuantos, .combine=c) %dopar% parte()
stopImplicitCluster()
integral <- sum(montecarlo) / (cuantos * pedazo)
print((pi / 2) * integral)
```

Este código genera en base las funciones de las integrales genera números pseudoaleatorios para obtener el resultado de la integral propuesta.

Este código se modificó para llevar a cabo la examinación del efecto del tamaño de la muestra en la precisión del estimado de la integral, esto se comparó con el resultado obtenido con Wolfram Alpha, el cual fue 0.031089.

- Código modificado

```
f <- function(x) { return(1 / (exp(x) + exp(-x))) }

suppressMessages(library(distr))

g <- function(x) { return((2 / pi) * f(x)) }

generador  <- r(AbscontDistribution(d = g)) # creamos un generador

cu <- seq(50, 300, 50)

resultados = data.frame()

desde <- 3

hasta <- 7

nucleos = 2

suppressMessages(library(doParallel))

registerDoParallel(makeCluster(nucleos))

parte <- function(x) {

  valores <- generador(x)

  return(sum(valores >= desde & valores <= hasta))
```

```
}
for (cuantos in cu) {
  res = numeric()
  for (replica in 1:5) {
    montecarlo <- foreach(i = 1:nucleos, .combine=c) %dopar% parte(cuantos)
    integral <- (sum(montecarlo) / (cuantos * nucleos))
    res = c(res, integral)
  }
  resultados = rbind(resultados, c(cuantos, res))
}
stopImplicitCluster()
n50=resultados[,2]
n100=resultados[,3]
n150=resultados[,4]
n200=resultados[,5]
n250=resultados[,6]
labels= c("50", "100", "150", "200", "250")
boxplot(n50, n100, n150, n200, n250, names=labels, ylab="Estimado", xlab="Muestra")
abline(h = 0.031089, col="red")
```

En el código se conservó activo el paquete “doParallel”, el cual permite hacer múltiples cálculos haciendo uso de los núcleos en la computadora. En el código se retiraron las líneas de comando que generaban los histogramas, y se retiraron las líneas que generaban varios clusters y sólo se dejó una línea de generación del cluster, donde se utilizaron dos núcleos para realizar los cálculos, ya que la computadora cuenta con 4 núcleos. También permaneció un stopCluster para detener los cálculos realizados por el cluster. Se generó un vector vacío “res=numeric()”, el cual se llenará con los resultados de la integral más los resultados del vector “res”. Se utilizó un rbind para recopilar los datos de la “data.frame” de resultados y finalmente se generó un boxplot en el cual se presentan el número de muestras contra el estimado.

Resultados

Como resultado de la examinación se obtuvo un diagrama-caja bigote de tamaño de muestra contra estimado de pi.

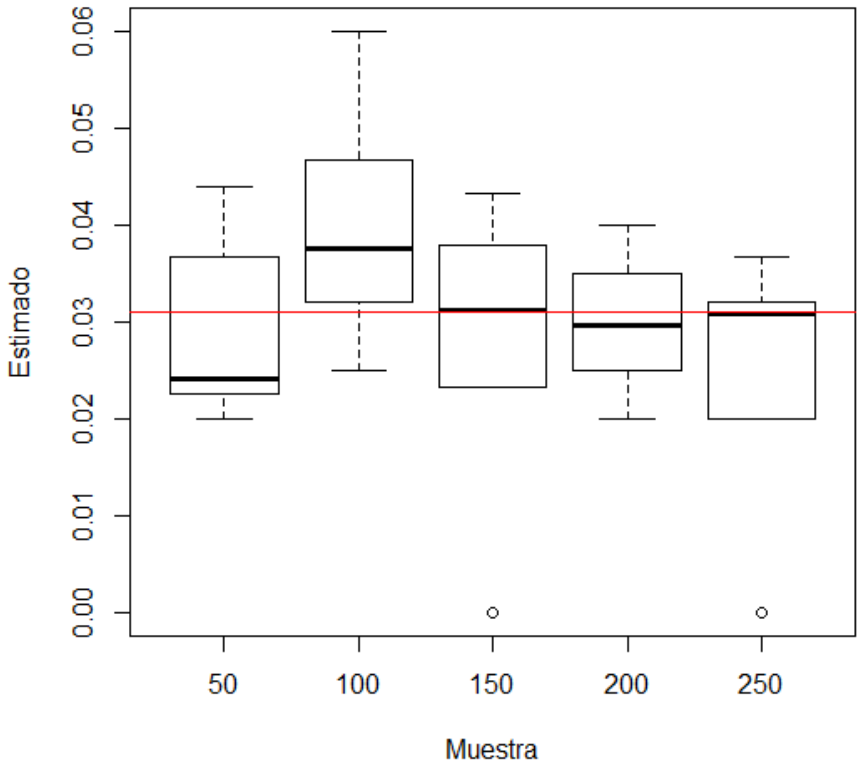

Fig. 1. Diagrama caja-bigote de muestra contra estimado.

En la figura 1 se muestra el diagrama caja-bigote de la muestra contra el estimado de pi, la línea roja que se observa es el valor de la integral obtenido con Wolfram Alpha, las cajas-bigote representan los resultados de repetir cinco veces, el número de muestra, se puede observar que la precisión se va aumentando cada vez más con el tamaño de la muestra.

Conclusiones

De esta práctica se puede concluir la importancia del método Monte-Carlo, el cual nos ayuda a generar muestras aleatorias en una distribución conocida. En este experimento se concluye que el tamaño de la muestra así como el número de repeticiones ayuda mejorar la precisión del estimado, esto se refiere a que la precisión aumenta con el número de repeticiones y así se obtienen resultados cercanos a los obtenidos por softwares especializados para la resolución de problemas matemáticos.

Bibliografía

1) Santos del Cerro, Jesús, (2006), *Historia de la probabilidad y la estadística (III)*, Ed. Delta Publicaciones.