Práctica 1. Movimiento Browniano

Ricardo Rosas Macías

21 de mayo de 2019

Resumen

Con fines educativos se hizó la práctica con el fin de observar la trayectoria que puede tener el movimiento de una nanopartícula o mejor conocido como movimiento Browniano. El software R permitió emular el movimiento de manera aleatoria, asimismo realizar la medición de manera cuantitativa, de su distancia Euclidiana y Manhattan.

1. Introducción

El movimiento Browniano es el desplazamiento aleatorio que una partícula o nanopartícula efectúa en un medio. En el experimento que se llevó acabo, la distancia que recorre la nanopartícula se encuentra cuantificada en discretos, de tal forma que cada paso tenga el mismo tamaño. La ejecución del código tiene el interés práctico de conocer la posición de la partícula respecto al origen.

2. Objetivo

La finalidad de la simulación del movimiento Browniano es discernir si la nanopartícula regresa al origen, además con el procesamiento de datos permitirá determinar si los parámetros afectan en el comportamiento.

2.1. Descripción

Lo que se debe hacer es [5]:

> "Examinar de manera sistemática los efectos de la dimensión en la probabilidad de regreso al origen del movimiento Browniano para dimensiones 1 a 8 en incrementos lineales de uno, variando el número de pasos de la caminata como potencias de dos con exponente de 6 a 12 en incrementos lineales de uno, con 30 repeticiones del experimento para cada combinación.

El primer reto es estudiar de forma sistemática y automatizada el tiempo de ejecución de una caminata (en milisegundos) en términos del largo de la caminata (en pasos) y la dimensión.

El segundo reto es realizar una comparación entre una implementación paralela y otra versión que no aproveche paralelismo en términos del tiempo de ejecución, aplicando una prueba estadística adecuada para determinar si la diferencia es significativa."

3. Resultados y conclusiones

Para obtener los resultados se le realizaron modificaciones al código anteriormente reportado [2][4], como se muestra en la parte inferior. En las primeras líneas se indica los incrementos de pasos de 2^6 a 2^{12}; que esta representada por la variable pasos, así como las 30 repeticiones que efectuara. Además el código esta paralelizado de modo que permita una ejecución más rápida, en las dimensiones 1 a 8.

```
rep <- 30
pasos <- sapply(6:12, function(x) {2**x})
eucl <- FALSE

cluster <- makeCluster(detectCores() - 1)
clusterExport(cluster, "dur")
clusterExport(cluster, "eucl")
clusterExport(cluster, "pasos")
datos <- data.frame()
for (dur in pasos) {
  for (dimension in 1:8) {
    counter <- 0
    clusterExport(cluster, "dimension")
    resultado <- parSapply(cluster, 1:rep,
                           function(r) {
                             pos <- rep(0, dimension)
                             test <- rep(0, dimension)
                             mayor <- 0
                             counter <- 0
                             for (t in 1:dur) {
                               cambiar <- sample(1:dimension, 1)
                               cambio <- 1
                               if (runif(1) < 0.5) {
                                 cambio <- -1
                               }
                               pos[cambiar] <- pos[cambiar] + cambio
                               if (eucl) {
                                 d <- sum(sqrt(pos**2))
                               } else { # Manhattan
                                 d <- sum(abs(pos))
                               }
                               if (d > mayor) {
                                 mayor <- d
                               }
                               if (all(pos==test)){
                                 counter <- counter + 1
```

```
                                    }
                                }
                                return(counter)
                            })
    datos <- rbind(datos, resultado)
  }
}
stopCluster(cluster)
Adata <- data.frame(datos)
```

Con ayuda de la paquetería *ggplot2* [1][3], se obtuvo la visualización de código anterior. En la figura 1 muestra que conforme avanzan las dimensiones los datos tienen más dispersión central, por lo que hay algunos datos atípicos que aparecen en forma de puntos en los gráficos que están fuera del rango intercuartílico. Por lo tanto hay más probabilidad de que los dos parámetros afectan al comportamiento de la nanopartícula.

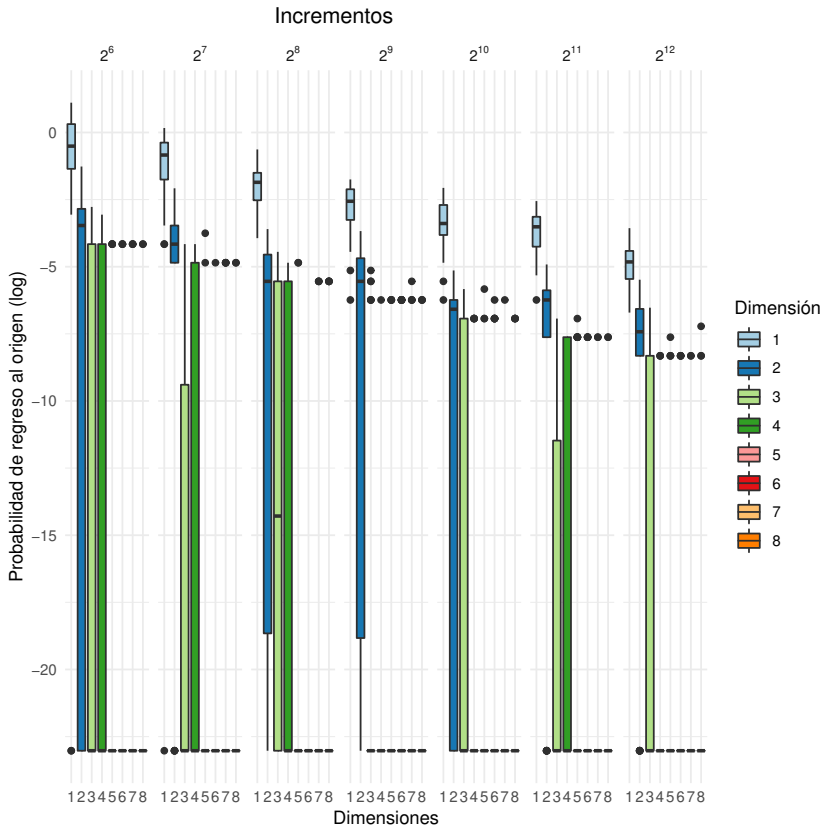

Figura 1: Distancia Manhattan de partícula en incrementos de 2^6 a 2^{12}

3.1. Reto 1

Para el primer reto se creo un código respecto a la literatura [6], con la intención de conocer el tiempo de ejecución respecto a la caminata que la nanopartícula realiza en 100 pasos.

```
rep <- 1
dur <- 100
dim <- 8
orig <- rep(0, dim)
tab <- numeric(length(rep*dim))
elapsed <- numeric(length(dur))
i = 1
j = 1
anterior <- 0
timeT <- system.time(
  for (dimension in dim:dim) {
    timeDim <- system.time(
      for (repeticion in 1:rep)  {
        cero <- 0
        pos <- rep(0, dimension)
        for (t in 1:dur) {
          timeD <- system.time(
           for (x in 1:1) {
            cambiar <- sample(1:dimension, 1)
             if (runif(1) < 0.5) {
                cambio <- 1
              } else {
                cambio <- -1
              }
              pos[cambiar] <- pos[cambiar] + cambio
              if (all (pos == orig)) {
                cero = cero + 1
              }
           })[3]
        time <- (timeD + anterior)
         elapsed[j] <- time
         anterior <- time
          j = j + 1
        }
        prob <- ((cero/dur)*100)
        tab[i] <- prob
        i = i + 1
      })[3]
  })
```

En la figura 2 se puede apreciar el comportamiento creciente del tiempo respecto a incremento en los pasos, esto nos indica que al tener una caminata más grande el tiempo tendrá una tendencia a aumentar.

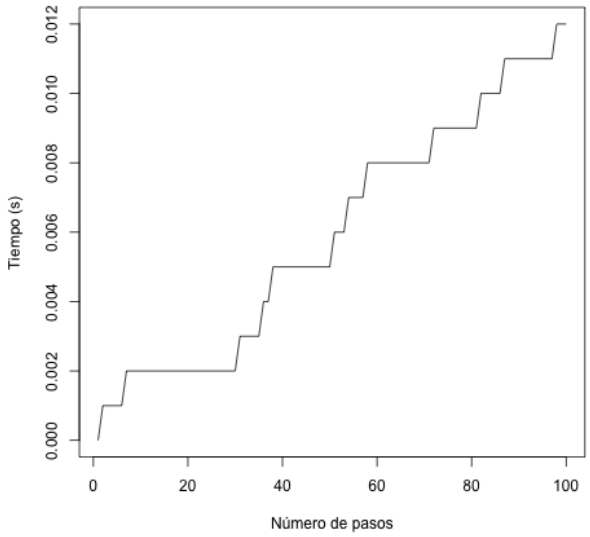

Figura 2: Porcentaje de soluciones

3.2. Reto 2

Por último se realizó el reto 2 con el fin de diferenciar el cambio de tiempo de ejecución al paralelizar el código.

```
library(parallel)

PSprocess <- Sys.time()
repetir <- 30
dur <-200
dimen <- 1:8

cluster <- makeCluster(detectCores() - 1)
clusterExport(cluster, "dur")
datos <-  data.frame()

for (dim in dimen) {
  clusterExport(cluster, "dim")
  resultado <- parSapply(cluster, 1:repetir,
                         function(x) {
                           pos <- rep(0, dim)
                           origen <- rep(0, dim)
                           contador <- 0
                           for (t in 1:dur) {
                             cambiar <- sample(1:dim, 1)
                             cambio <- 1
                             if (runif(1) < 0.5) {
                               cambio <- -1
                             }
                             pos[cambiar] <- pos[cambiar] + cambio
                             d <- dist(pos)
```

```
                                if (sum(pos == origen) == dim) {
                                  contador <-  contador + 1
                                }
                              }
                              return(contador)
                            }
  )
  datos <- rbind(datos, resultado)
}
stopCluster(cluster)

PEprocess <- Sys.time()
TimeP <- PEprocess - PSprocess
print(TimeP)

Sprocess <- Sys.time()
datos <-  data.frame()
for (dim in dimen) {
  resultado <- sapply(1:repetir,
                      function(x) {
                        pos <- rep(0, dim)
                        origen <- rep(0, dim)
                        contador <- 0
                        for (t in 1:dur) {
                          cambiar <- sample(1:dim, 1)
                          cambio <- 1
                          if (runif(1) < 0.5) {
                            cambio <- -1
                          }
                          pos[cambiar] <- pos[cambiar] + cambio
                          d <- dist(pos)
                          if (sum(pos == origen) == dim) {
                            contador <-  contador + 1
                          }
                        }
                        return(contador)
                      }
  )
  datos <- rbind(datos, resultado)
}

Eprocess <- Sys.time()
TimeSP <- Eprocess - Sprocess
print(TimeSP)

vstime <- data.frame(TimeSP, TimeP)
print(vstime)
ftc <- kruskal.test(vstime)
ftc$p.value
ftc$p.value < 0.05
```

En el cuadro 1 se encuentran los resultados del código anterior, donde se observa una diferencia de 0.0584 segundos menos al paralelizar. Asimismo se obtuvieron resultados de la prueba estadística *Shapiro-Wilk*

test del código anterior, en dónde se puede notar una diferencia en el valor $p < 0{,}05$. Por lo tanto se puede concluir que la diferencia es minima.

Cuadro 1: Comparación de tiempo de ejecución, normal versus paralelizada.

TimeSP (Seg)	TimeP (Seg)	ftc$p.value	ftc$p.value <0.05
2.020045	1.961556	0.3173105	FALSE

Referencias

[1] Jodie Burchell. Creating plots in r using ggplot2 - part 10: boxplots, 2016. URL `http://t-redactyl.io/blog/2016/04/creating-plots-in-r-using-ggplot2-part-10-boxplots.html`.

[2] Astrid González. Exercise1, 2018. URL `https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_1/`.

[3] Alboukadel Kassambara. ggplot2 box plot : Quick start guide - r software and data visualization, 2015. URL `http://www.sthda.com/english/wiki/ggplot2-box-plot-quick-start-guide-r-software-and-data-visualization`.

[4] Ricardo Rosas Macías. Práctica 1: Movimiento browniano, 2019. URL `https://github.com/RicardoRosMac/Simulation/tree/master/HWP1`.

[5] Elisa Schaeffer. Práctica 1: Movimiento browniano, 2019. URL `https://elisa.dyndns-web.com/teaching/comp/par/p1.html`.

[6] José Ángel Cavazos Contreras. P1, 2018. URL `https://github.com/Xhangelx13x/SimNano_Xhan/blob/master/P1/Reporte_P1`.