

**UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN**  
**FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA**  
**POSGRADO**

*Maestría en ciencias de la ingeniería con orientación en nanotecnología*

**PORTAFOLIO**

**SIMULACIÓN COMPUTACIONAL DE NANOMATERIALES**  
*Impartida por Dra. Elisa Shaeffer*

Dulce Esperanza Carrasco Castillo  
1445183

San Nicolás de los Garza, N.L. a 7 de junio de 2019

# Práctica 1: Movimiento Browniano

1445183

5 de junio de 2019

## 1. Retroalimentación

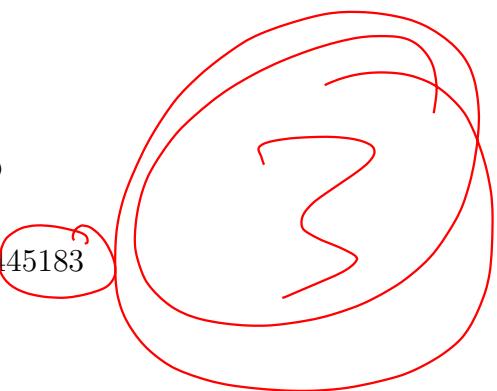
En la primera práctica batallé mucho para hacerla a pesar que era una práctica sencilla. No estaba familiarizada con nada de programación, menos con el software **Rproject**, al realizarla de verdad creí que la había echo bien, que solo era cambiar los valores, pero no era así, lo que tenía que hacer era hacerle alguna modificación al código para obtener las probabilidades que se requerían

A finales de curso y hacer las correcciones pertinentes me di cuenta que no había entendido al principio cosas muy fáciles, que ahora ya entiendo mejor por lo que pude hacer las correcciones.

# Movimiento Browniano

Dulce Esperanza Carrasco Castillo 1445183

29 de enero de 2019



## 1. Introducción

Se le llama Movimiento Browniano al movimiento aleatorio que realiza una partícula en cierto entorno, en esta práctica se describe este movimiento mediante probabilidad.

## 2. Objetivo

Introducir al uso de R por medio del Movimiento Browniano de una partícula y su probabilidad de regresar al origen.

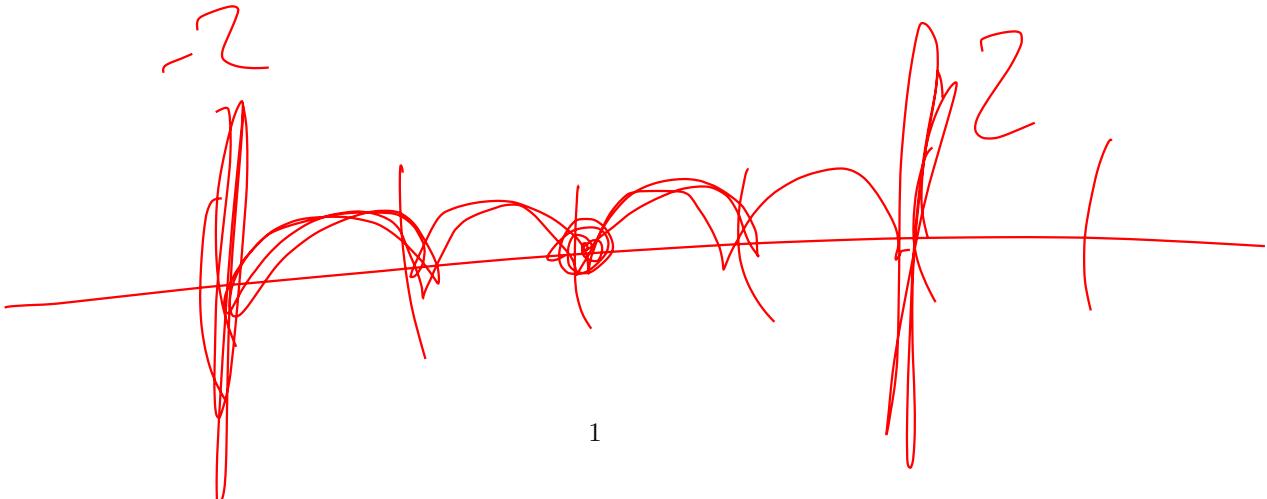
## 3. Descripción

En un script en R (p1.R) se introduce el código proporcionado por la práctica, se ajusta el código a las especificaciones que se piden en este caso se requieren de 1 a 8 dimensiones, con caminatas de potencias de 2 con exponentes de 6 a 12 y con 30 repeticiones para cada combinación, los resultados se interpretan en una gráfica donde la distancia es tipo Manhattan.

elisaweb

## 4. Resultados

Se puede observar como va aumentando el límite de pasos (distancia máxima) y la variación de probabilidad entre dimensiones, las cuales muestran un comportamiento aleatorio. (1)



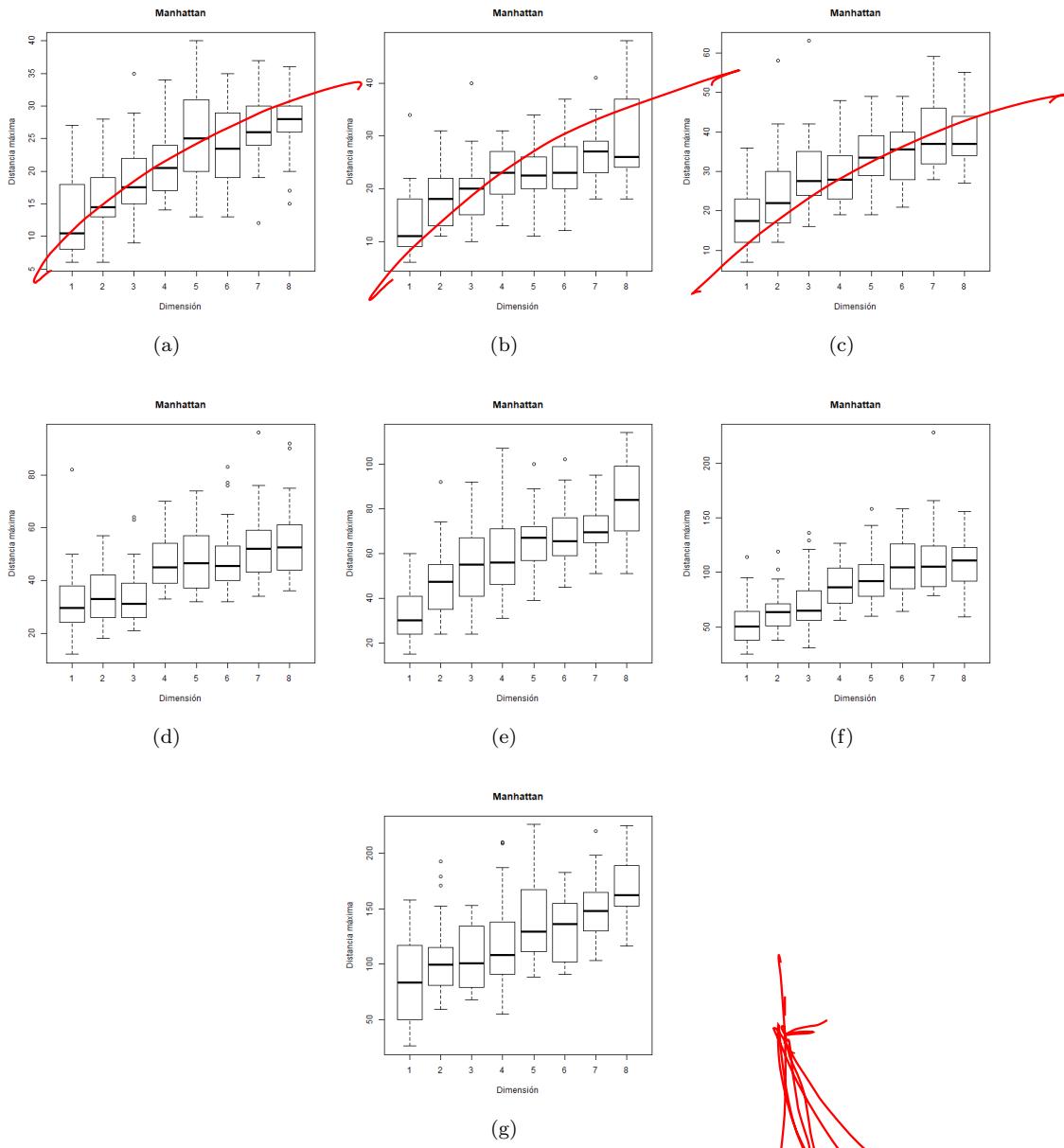


Figura 1: Gráficas Manhattan

# Movimiento Browniano

1445183

5 de junio de 2019

## 1. Introducción

Se le llama Movimiento Browniano al movimiento aleatorio que realiza una partícula en cierto entorno, en esta práctica se describe este movimiento mediante la probabilidad.

## 2. Objetivo

Examinar de manera sistemática los efectos de la dimensión en la probabilidad de regreso al origen de una partícula que se mueve aleatoriamente para dimensiones de 1 a 8, variando el número de pasos de la caminata y haciendo repeticiones del experimento.

## 3. Descripción

En el código proporcionado por la *práctica 1* [3], se ajusta el código para tener ocho dimensiones, con caminatas de potencias de 2 con exponentes de 6 a 12 y con 30 repeticiones para cada combinación, el algoritmo se modificó en la **línea 20** para indicar cuando la partícula regrese al origen.

```
1 P1 <- function(i){  
2   cluster <- makeCluster(detectCores()-1)  
3   datos <- data.frame()  
4   repeticiones <- 30  
5   for (dimension in 1:8) {  
6     duracion<-2^(i)  
7     clusterExport(cluster, "duracion") #Creamos un cluster donde le asignamos la duracion y dimension  
8     clusterExport(cluster, "dimension")  
9     resultado <- parSapply(cluster, 1:repeticiones,  
10       function(r) {  
11         pos <- rep(0, dimension)  
12         cont <- 0  
13         for (t in 1:duracion) {  
14           cambiar <- sample(1:dimension, 1)  
15           cambio <- 1  
16           if (runif(1) < 0.5) {  
17             cambio <- -1  
18           }  
19           pos[cambiar] <- pos[cambiar] + cambio #realizamos el cambio en la  
20           posicion anteriormente elegida  
21           if (all(pos==0)) { #Si llega al origen, sumara +1 al contador para poder  
22             sacar las probabilidades  
               cont <- cont + 1  
               #break; #rompemos la caminata
```

```

23
24         }
25     }
26     return(cont/duracion) #obtener la probabilidad que tuvo la particula para
27     regresar a su origen
28   })
29   datos <- rbind(datos, resultado) #la informacion de datos y resultados se guardan en filas dentro
30   de un dataframe
31 }

```

## 4. Resultados

Se puede observar en la figura [I](#) que la probabilidad tiene un decrecimiento conforme aumenta el número de dimensiones y el valor de la probabilidad es mayor conforme aumenta el número de pasos.

Si comparamos la figura [Ia](#) con la figura [Ig](#) en ambas se tiene la mayor probabilidad de que la partícula regrese al origen en una dimensión, en cambio la probabilidad es mayor cuando se realizan mayor cantidad de pasos.

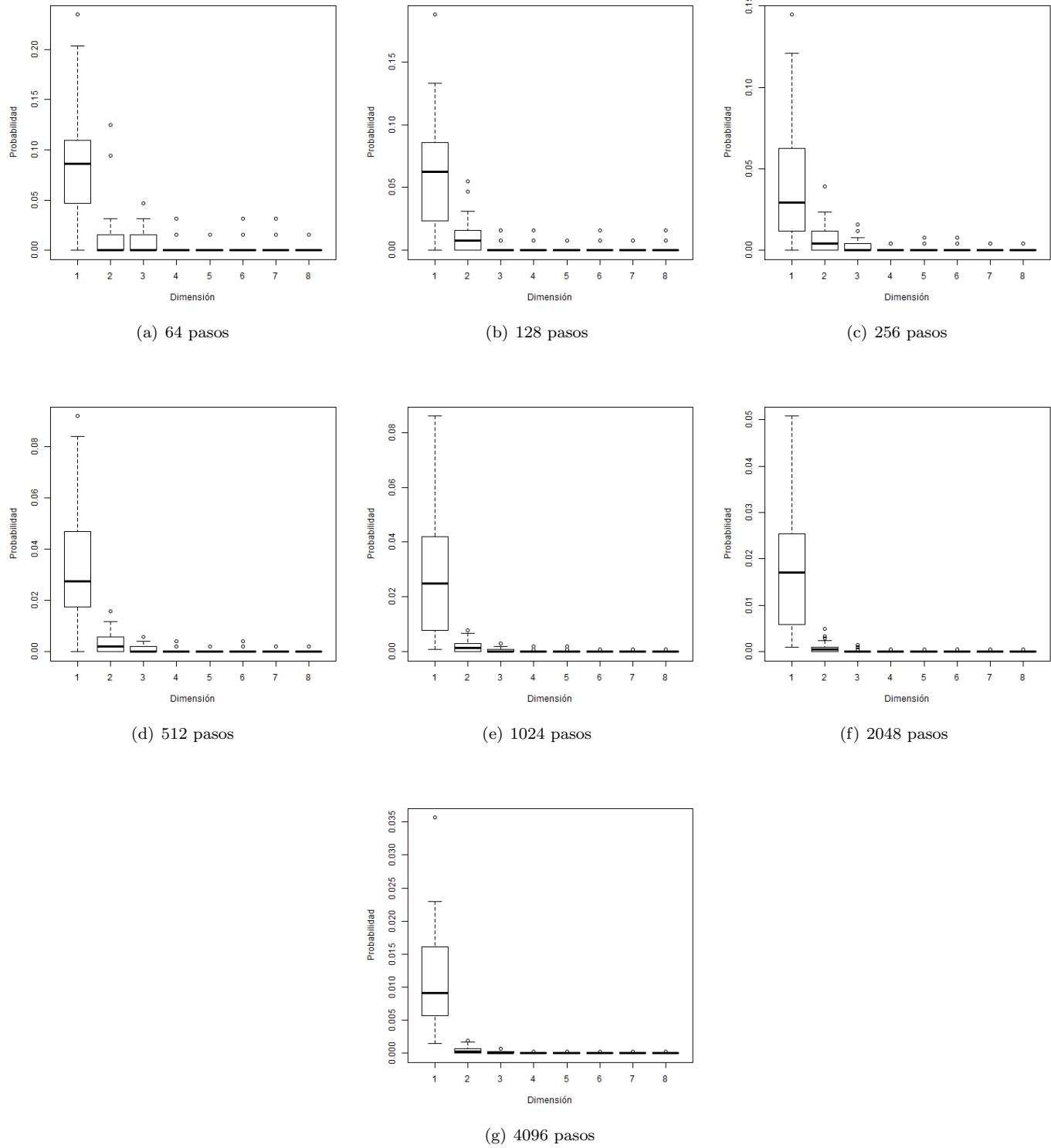


Figura 1: Probabilidad de regresar al origen.

## 5. Conclusión

Se puede concluir que la mejor combinación en este experimento con la mayor probabilidad de que la partícula regrese al origen tomando en cuenta la distancia manhattan es en una dimensión y con 4096 pasos (duración del experimento).

## 6. Reto 1

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 código proporcionado [3] se modifica de manera que se ejecute 100 veces, se hace uso de `system.time` para medir el tiempo de ejecución, los resultados se guardan en un `data.frame` [2]

```
1 library(parallel)
2 repetir <- 30
3 replicar<-100
4 t<-function(replicar){ #replicas para tiempo
5   a<-0
6   for(rep in 1:replicar){
7     tiempo=system.time(experimento(replica))
8     a<-a+tiempo[3]
9   }
10  }
11 }
12
13 datos <- data.frame()
14
15 experimento <- function(repetir) {
16   pos <- rep(0, dimension)
17   contador <- 0
18   for (t in 1:duracion) {
19     cambiar <- sample(1:dimension, 1)
20     cambio <- 1
21     if (runif(1) < 0.5) {
22       cambio <- -1
23     }
24     pos[cambiar] <- pos[cambiar] + cambio
25     if (all(pos==0)) {
26       contador <- contador+1
27     }
28   }
29   return(contador/duracion)
30 }
31
32 cluster <- makeCluster(detectCores())
33 clusterExport(cluster , "experimento")
34
35 for (dimension in 1:8) {
36   for(i in 6:12){
37     duracion<-2^(i)
38     clusterExport(cluster , "t")
39     clusterExport(cluster , "dimension")
40     clusterExport(cluster , "duracion")
41     resultado <- parSapply(cluster , 1:repetir , t)
42     datos <- rbind(datos , resultado)
43   }
44 }
45 stopCluster(cluster)
```

## 7. Resultados reto 1

En la figura 2 se compara el tiempo de ejecución para la dimensión uno y ocho con sus respectivos pasos (64 a 4096), se puede observar que para una dimensión el tiempo es menor en los primeros pasos que en la dimensión 8, pero ambas tienen casi el mismo tiempo de ejecución en los últimos pasos. También se puede observar que lo que afecta al tiempo de ejecución es el número de pasos y no las dimensiones.

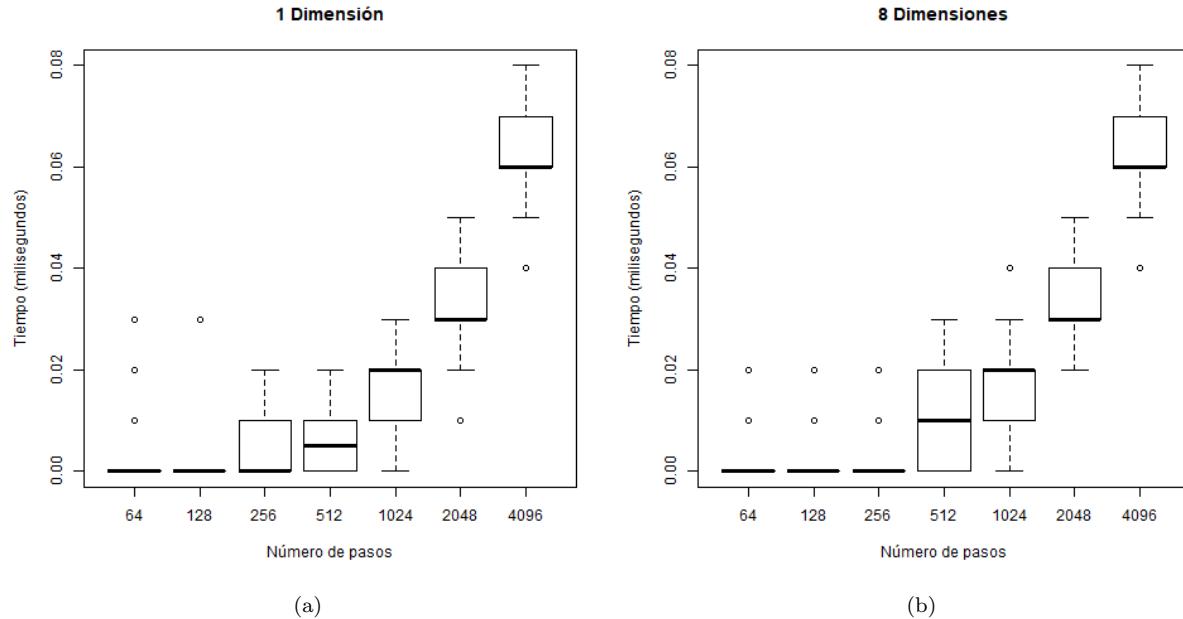


Figura 2: Tiempo de ejecución.

## 8. Reto 2

El reto dos es realizar una comparación entre la implementación paralela y otra que no aproveche el paralelismo en términos de tiempo de ejecución.

Se hizo uso de `parSapply` para medir el tiempo de forma paralela para una dimensión, en la figura 3 se puede observar que el tiempo de ejecución es menor de manera paralela que secuencial.[\[3\]](#)

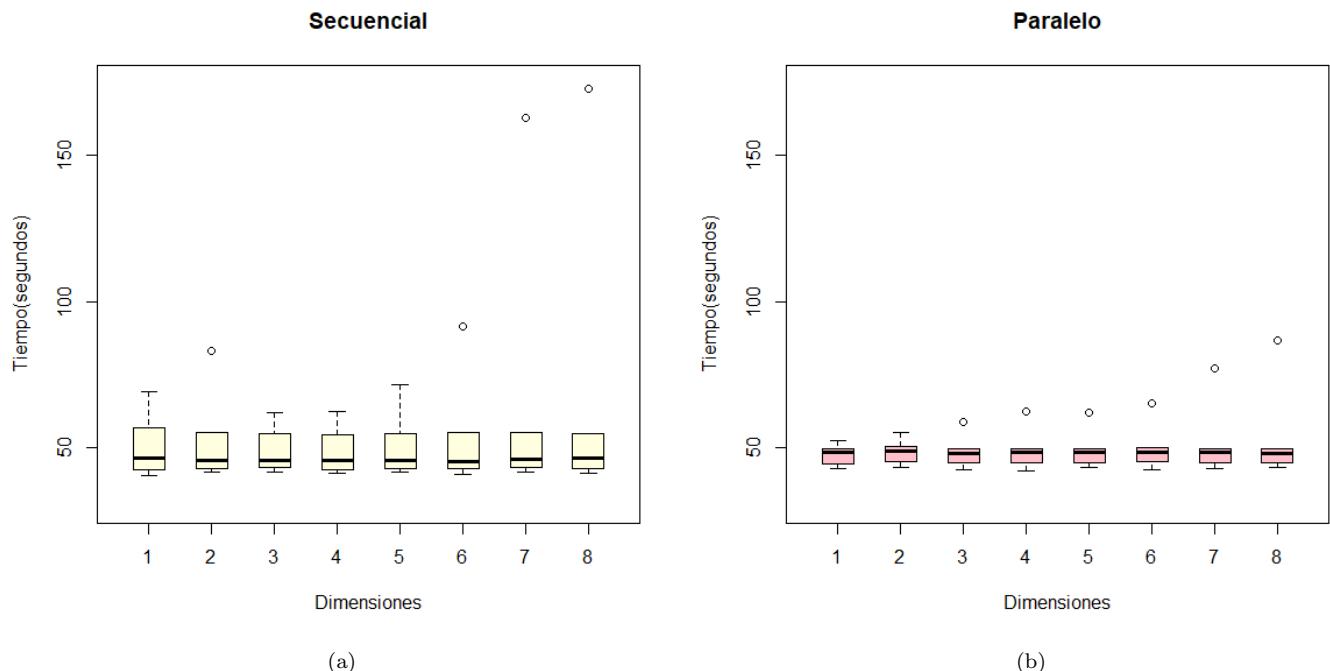


Figura 3: Comparación de ejecución secuencial y paralela.

## Referencias

- [1] Astrid González. Práctica 1, 2018. URL [https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise\\_1/1/](https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_1/1/).
- [2] Marco Guajardo. Práctica 1, 2019. URL <https://sourceforge.net/p/simulaciondesistemas/code/ci/master/tree/P1/>.
- [3] Elisa Schaeffer. Práctica 3, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p1.html>.

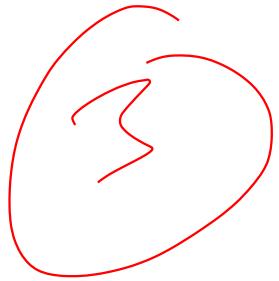
# Práctica 2: Autómata Celular

1445183

5 de junio de 2019

## 1. Retroalimentación

En esta práctica no sabía usar bien `boxplot` ni poner el código R ni la bibliografía, ya en el corregido se arreglaron todos esos errores además de mejorar la calidad de etiquetas y explicación.



## Práctica 2: autómata celular

Dulce Esperanza Carrasco Castillo 1445183

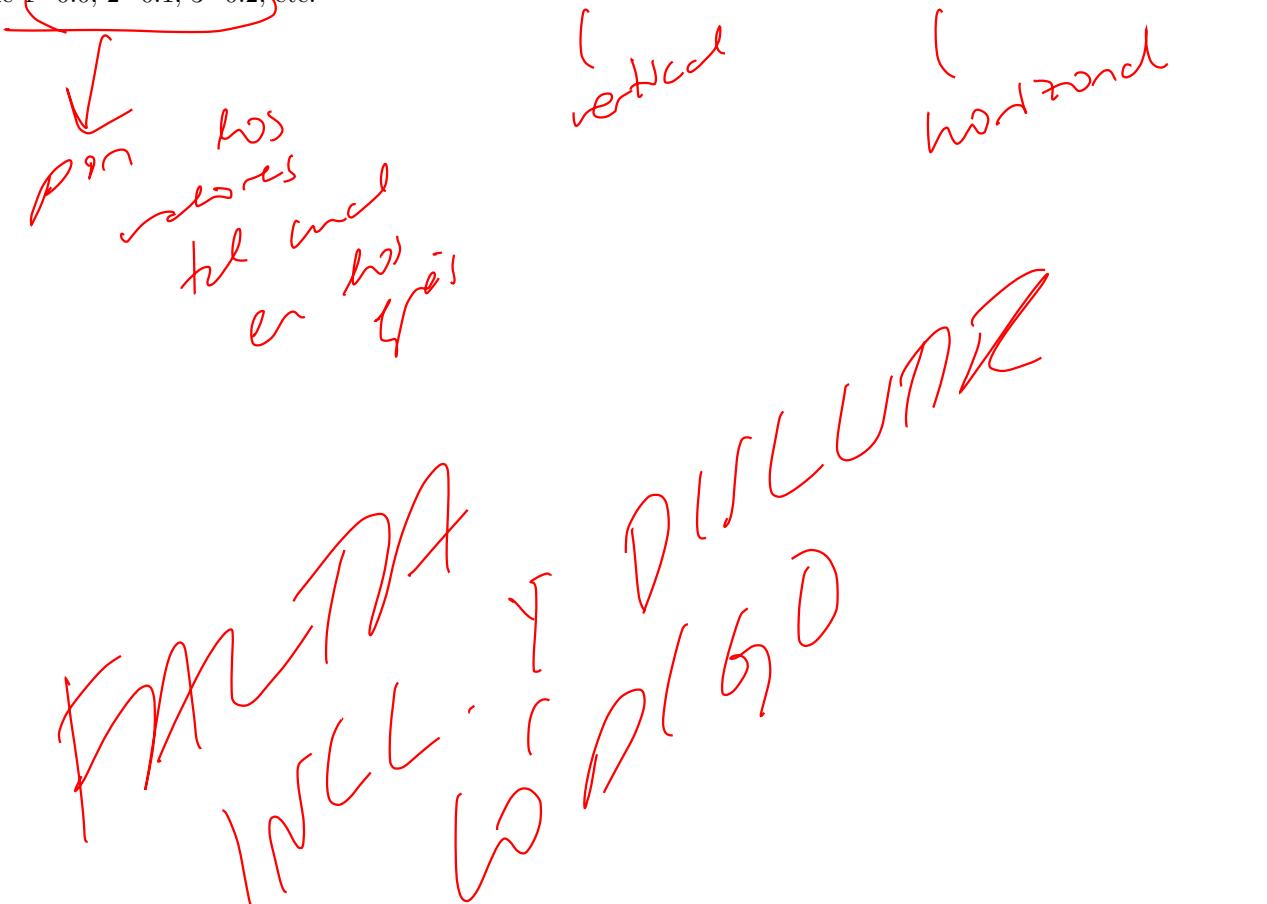
5 de febrero de 2019

### 1. Descripción

Usando R se hace una malla de 30 por 30 celdas, se varía la probabilidad inicial de 0 a 1 en cada corrida [?], después se ajustan los valores para indicar que una celda está viva (1) ó muerta (0) junto con los pasos requeridos, en este caso de 0.1; con lo anterior establecido se va variando el número de iteraciones, de manera que cada celda vive o muere dependiendo del tipo de celdas que tenga como vecinas. El número de iteraciones se determina cuando las células (celdas) mueren, esto se puede observar en R por el número de veces que aparece el escrito “Ya no queda nadie vivo”.

### 2. Resultados

En la figura 1 se muestran los resultados donde el eje ~~y~~ son las iteraciones y el eje ~~x~~ la probabilidad inicial de 0.0 a 1.0 donde  $1=0.0$ ,  $2=0.1$ ,  $3=0.2$ , etc.



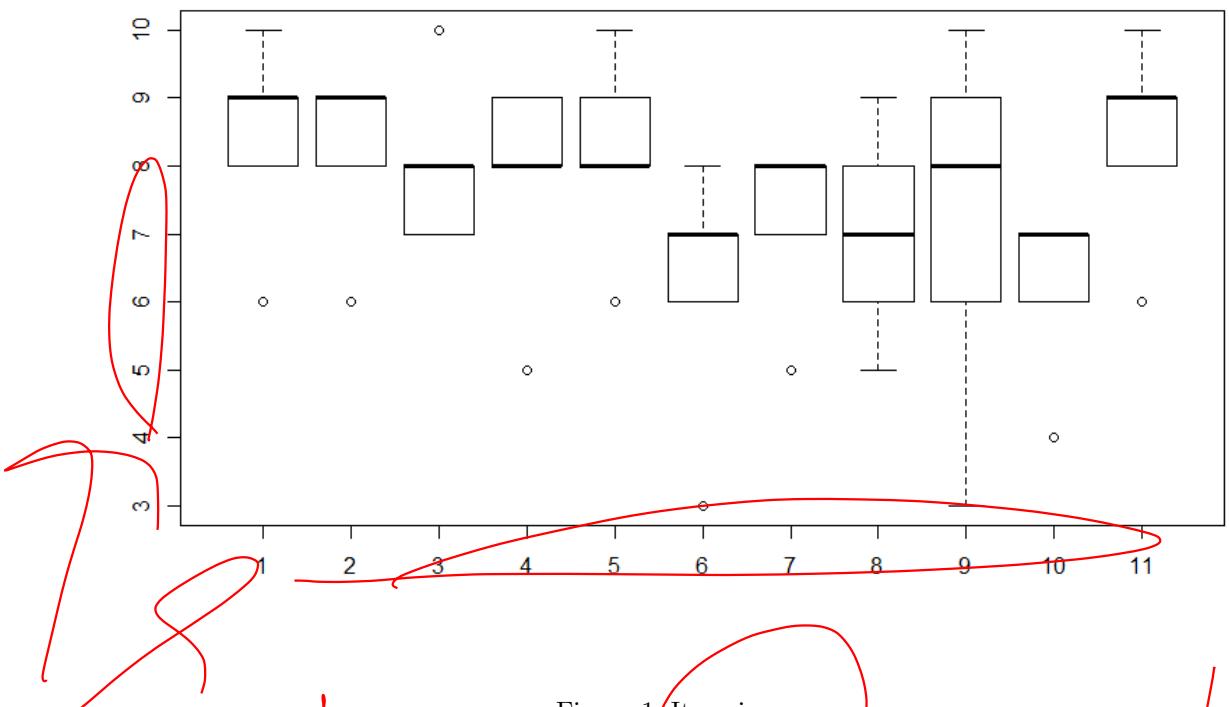


Figura 1: Iteraciones

Beta hypercube  
se descarta

FMD BIBLIOGRAFIA

# Práctica 2: autómata celular

1445183

4 de junio de 2019

## 1. Objetivo

Determinar el número de iteraciones que procede la simulación en un malla de 30 por 30 celdas hasta que se mueran todas las celdas, variando la probabilidad inicial de *celda viva*.

## 2. Descripción

Haciendo uso de Rstudio con el código proporcionado [2] se hace una malla de 30 por 30 celdas, con el uso de `for` se varía la probabilidad inicial de 0 a 1 en pasos de 0.1 para cada corrida con 30 repeticiones cada una.

```
1 dim <- 30 #matriz 30x30
2 num <- dim^2
3 data <- data.frame()
4 for(p in seq(0.1,0.9,0.1)) { #condicion para la probabilidad inicial
5   numitvector <- c()
6   for(rep in 1:30) { #repeticiones del experimento
7     actual <- matrix(1 * (runif(num) < p), nrow=dim, ncol=dim)
8     suppressMessages(library("sna"))
9     paso <- function(pos) {
10       fila <- floor((pos - 1) / dim) + 1
11       columna <- ((pos - 1) %%dim) + 1
12       vecindad <- actual[max(fila - 1, 1) : min(fila + 1, dim),
13                           max(columna - 1, 1) : min(columna + 1, dim)]
14       return(1 * ((sum(vecindad) - actual[fila, columna]) == 3))
15     }
16   }
17 }
```

Se establece en el código un rango de iteraciones, después el número de iteraciones que tuvo la celda antes de morir se puede observar en R por el número de veces que aparece el escrito “Ya no queda nadie vivo”.

```
1 for (iteracion in 1:50) { #iteraciones
2   clusterExport(cluster, "actual")
3   siguiente <- parSapply(cluster, 1:num, paso)
4   if (sum(siguiente) == 0) { # todos murieron
5     print("Ya no queda nadie vivo.")
6     numit <- iteracion
7     break;
8   }
```

### 3. Resultados

En la figura I se puede observar que para las probabilidades de 0.1 y 0.9 las iteraciones son pocas, en cambio para las probabilidades de 0.5 a 0.7 las iteraciones son más II.

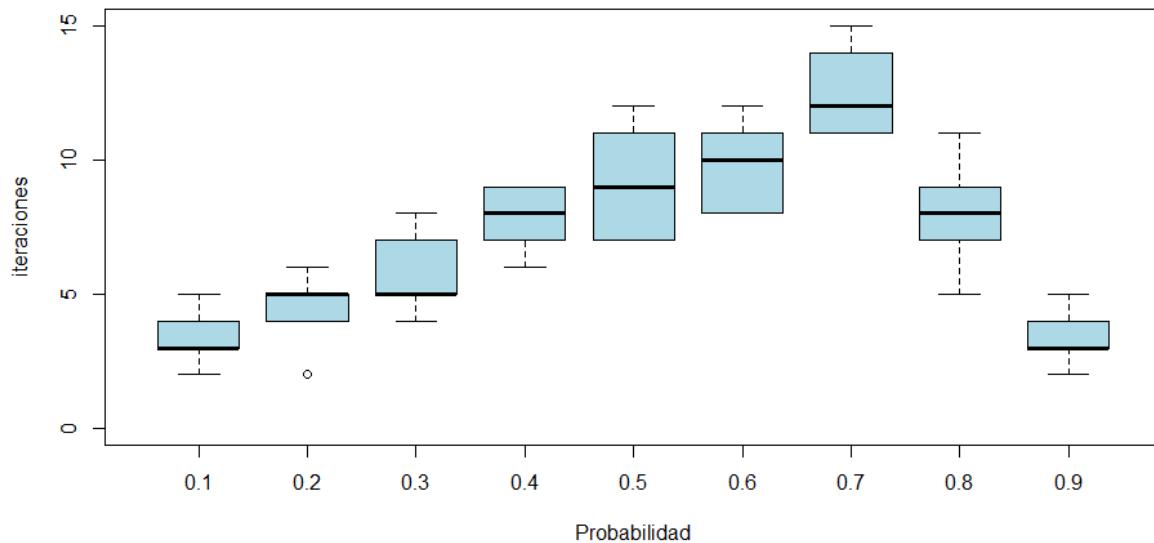


Figura 1: Iteraciones antes que mueran todas las celdas.

### 4. Conclusión

Para la probabilidad de 0.1 se generan pocas células (celdas) vivas por lo que las iteraciones necesarias para que mueran son pocas, esto es casi igual para las probabilidades menores a 0.4.

Para la probabilidad de 0.9 la mayor parte de las células que se generan están vivas por lo que se requieren menos iteraciones para desaparecer II.

## Referencias

- [1] Edson Cepeda. Práctica 2, 2019. URL <https://sourceforge.net/projects/systemssimulation/files/P2/>.
- [2] Elisa Schaeffer. Practica 2, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p1.html>.

# Práctica 3: Teoría de colas

1445183

5 de junio de 2019

## 1. Retroalimentación

En esta práctica se mejoraron los resultados, también la bibliografía.

4

## Práctica 3: Teoría de colas

Dulce Esperanza Carrasco Castillo 1445183

12 de febrero de 2019

### 1. Objetivo

Conocer que la rapidez de ejecución de tareas en paralelo depende de la cantidad de núcleos que realmente están trabajando y también del orden en el que se estén ejecutando las tareas [?].

### 2. Descripción

En esta práctica se midió el tiempo que se tarda realizar ciertas tareas, se utilizó la combinación de números primos y no primos en diferente proporción y se varió la cantidad de núcleos trabajando. Primero se verificó la cantidad de núcleos con los que cuenta el dispositivo en el que se trabajó usando el comando: `detectCores()`, se creó un vector para los números primos y otro para los números no primos (misma cantidad de números de cada uno) 1, después se hizo otro vector para combinar los números primos y no primos y que tomara los números aleatoriamente. Para variar la proporción se disminuía la cantidad de números a un vector(primos) y se le agregaban al otro(noprimos) para mantener la misma cantidad de números totales. Se corrieron para 3, 2 y 1 núcleo. Se repitió el proceso para números de 7 y 9 dígitos.

### 3. Resultados

Se observa en la figura 2 que al usar la mayor cantidad de núcleos los tiempos de ejecución no varían mucho pero si se lleva más tiempo cuando está en una proporción 50/50, de manera que al disminuir la cantidad de primos o noprimos disminuye el tiempo de ejecución. Con dos núcleos, el trabajo se reparte de manera más ordenada y con un núcleo se puede ver que los números primos hacen la tarea más pesada.

En la figura 3 se utilizan números de 7 dígitos, se puede observar que al usar la mayor cantidad de núcleos los tiempos de ejecución son menores y va aumentando conforme se usan menos núcleos para realizar la tarea. Al usar menor cantidad de núcleos los números primos hacen la tarea más pesada y el tiempo de ejecución es mayor.

En la figura 4 se utilizaron números con 9 dígitos, se puede ver que los tiempos son más grandes que los tiempos con 5 y 7 dígitos, y hay mucha diferencia entre los tiempos que se tarda cada combinación, lo que vuelve a confirmar que donde hay mayor cantidad de números primos, mayor es el tiempo que se tarda en realizar la tarea, por lo que los números primos hacen la tarea pesada.

```

1 primo <- function(n) {
2   if (n == 1 || n == 2) {
3     return(TRUE)
4   }
5   if (n %% 2 == 0) {
6     return(FALSE)
7   }
8   for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
9     if ((n %% i) == 0) {
10       return(FALSE)
11     }
12   }
13   return(TRUE)
14 }
15
16 noprimos<-c(valores)
17 mispri<-c(valores)
18
19 combinacion <- sort(sample(c(mispri, noprimos)))
20 original <- combinacion
21 invertido <- rev(combinacion)
22 replicas <- 10
23 suppressMessages(library(doParallel))
24 registerDoParallel(makeCluster(detectCores() - 3))
25 ot <- numeric()
26 it <- numeric()
27 at <- numeric()
28 for (r in 1:replicas) {
29   ot <- c(ot, system.time(foreach(n = original, .combine=c) %dopar% primo(n)[3])) # de menor a mayor
30   it <- c(it, system.time(foreach(n = invertido, .combine=c) %dopar% primo(n)[3])) # de mayor a menor
31   at <- c(at, system.time(foreach(n = sample(original)), .combine=c) %dopar% primo(n)[3])) # orden aleatorio
32 }
33 stopImplicitCluster()
34 summary(ot)
35 summary(it)
36 summary(at)

```

Figura 1: Código

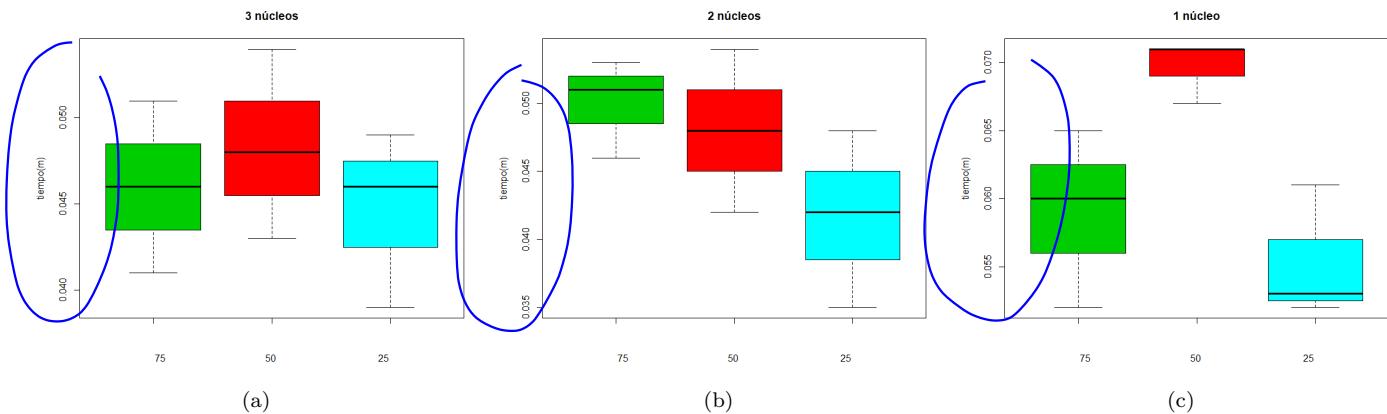


Figura 2: 5 dígitos

0.06

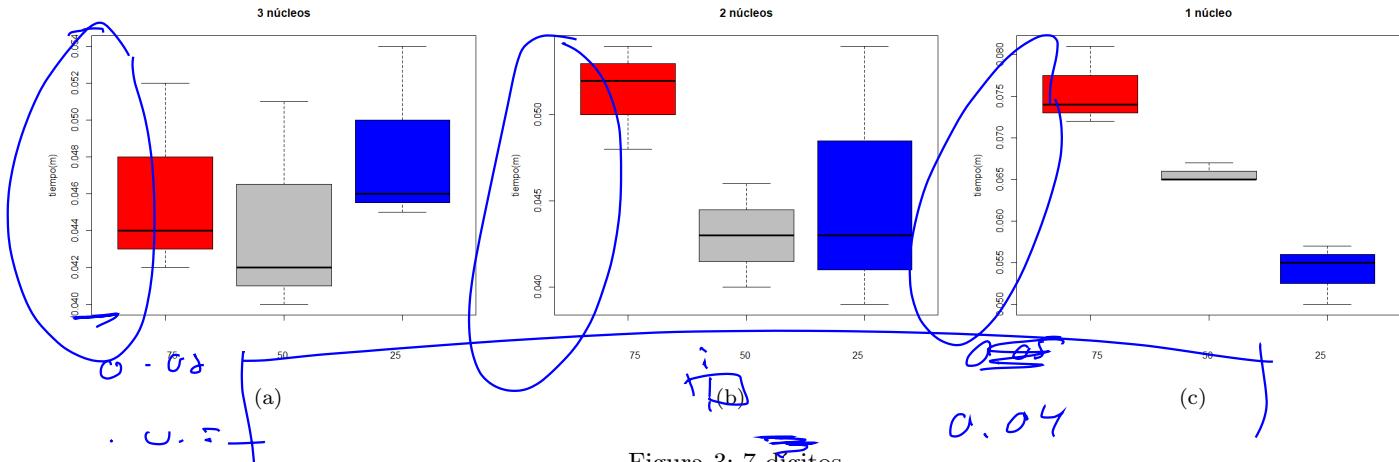
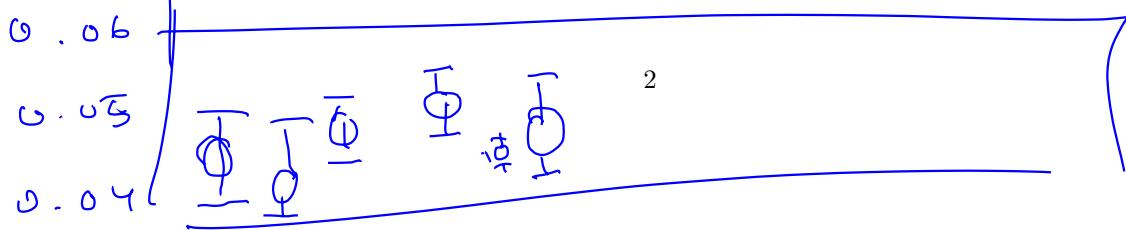


Figura 3: 7 dígitos



11  
 11  
 11

: Sistemas  
 de Penelación  
 Peridódica

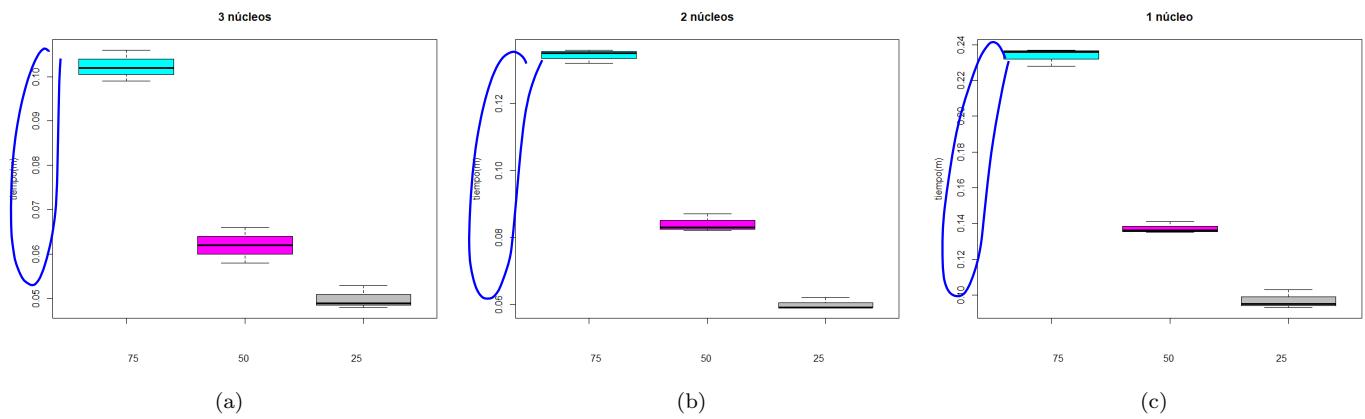


Figura 4: 9 dígitos

# Práctica 3: Teoría de colas

1445183

4 de junio de 2019

## 1. Objetivo

Conocer que la rapidez de ejecución de tareas en paralelo depende de la cantidad de núcleos que realmente están trabajando y también del orden en el que se estén ejecutando las tareas.

## 2. Descripción

Primero se verifica la cantidad de núcleos con los que cuenta el dispositivo usando el comando `detectCores()`, los números primos se toman de un archivo llamado `primos 20k - 30k.txt` [3] por lo que la cantidad de dígitos utilizados son cinco.

```
1 lista <- read.csv("primos 20k - 30k.txt", sep="", header=FALSE)
2
3 #se extraen los numeros primos del archivo en un vector
4 numeros <- c()
5 for(i in 1:129) {
6   for(j in 1:8) {
7     nums <- c(numeros, lista[i, j])
8   }
9 }
10 }
```

Se crean tres vectores, uno para los números primos, otro para los números no primos y otro para combinar los números primos y no primos de manera que se tomen aleatoriamente, también se varían las proporciones en 75/25, 50/50, 25/75.

```
1 #vectores de primos y no primos
2 primos <- c()
3 noprimos <- c()
4 desde <- numeros[1]
5 hasta <- numeros[length(nums)]
6 for(i in desde:hasta) {
7   if(primo(i) == TRUE) {
8     primos <- c(primos, i)
9   }
10  if(primo(i) == FALSE) {
11    noprimos <- c(noprimos, i)
12  }
13 }
14
15 noprimos <- sample(noprimos[1]:noprimos[length(noprimos)], length(primos))
16 }
```

```

17 # Se varian las proporciones las proporciones de primos y no primos
18 #75/25
19 mvectores1 <- c()
20 mvectorescomb1 <- c()
21 mvectorescomb1 <- c(mvectorescomb1, sample(primos, length(primos)*0.75))
22 mvectorescomb1<- c(mvectorescomb1, sample(noprimos, length(noprimos)*0.25))
23 mvectores1 <- sample(mvectorescomb1)
24
25 #50/50
26 mvectores2 <- c()
27 mvectorescomb2 <- c()
28 mvectorescomb2 <- c(mvectorescomb2, sample(primos, length(primos)*0.5))
29 mvectorescomb2 <- c(mvectorescomb2, sample(noprimos, length(noprimos)*0.5))
30 mvectores2 <- sample(mvectorescomb2)
31
32 #25/75
33 mvectores3 <- c()
34 mvectorescomb3 <- c()
35 mvectorescomb3 <- c(mvectorescomb3, sample(primos, length(primos)*0.25))
36 mvectorescomb3 <- c(mvectorescomb3, sample(noprimos, length(noprimos)*0.75))
37 mvectores3 <- sample(mvectorescomb3)

```

Con el uso de `for` se asignan los núcleos que se van a utilizar y las proporciones mencionadas anteriormente, para medir el tiempo se hace uso de `system.time` al final se hace la prueba de *Kruskal-Wallis* para saber si las proporciones causan algún efecto en el tiempo de ejecución [1].

```

1 nucleos <- detectCores(logical = FALSE)-1 #Numero de nucleos del computador
2 replicas <- 10 #Numero de veces a repetir el experimento
3
4 for(p in 1:3) { #Opciones de proporcion 1- 75%, 2- 50%, 3- 25%
5   if(p == 1) {
6     for(nuc in nucleos:1) { #Variacion el numero de nucleos
7       registerDoParallel(makeCluster(detectCores() - nuc))
8
9       for (r in 1:replicas) { #Repetir el experimento cuantas veces la variable replica indique
10
11         if(nuc == 1) { #comparando si se asigno 1 nucleo
12           Aot1 <- c(Aot1, system.time(foreach(n = original1 , .combine=c) %dopar% primo(n))[3]) # de
13             menor a mayor
14           Ait1 <- c(Ait1, system.time(foreach(n = invertido1 , .combine=c) %dopar% primo(n))[3]) # de
15             mayor a menor
16           Aat1 <- c(Aat1, system.time(foreach(n = sample(original1) , .combine=c) %dopar% primo(n))[3])
17             # orden aleatorio
18
19         }
20         if(nuc == 2) {
21           Aot2 <- c(Aot2, system.time(foreach(n = original1 , .combine=c) %dopar% primo(n))[3]) # de
22             menor a mayor
23           Ait2 <- c(Ait2, system.time(foreach(n = invertido1 , .combine=c) %dopar% primo(n))[3]) # de
24             mayor a menor
25           Aat2 <- c(Aat2, system.time(foreach(n = sample(original1) , .combine=c) %dopar% primo(n))[3])
26             # orden aleatorio
27
28       }
29       if(nuc == 3) {
30         Aot3 <- c(Aot3, system.time(foreach(n = original1 , .combine=c) %dopar% primo(n))[3]) # de
31             menor a mayor
32           Ait3 <- c(Ait3, system.time(foreach(n = invertido1 , .combine=c) %dopar% primo(n))[3]) # de
33             mayor a menor
34           Aat3 <- c(Aat3, system.time(foreach(n = sample(original1) , .combine=c) %dopar% primo(n))[3])
35             # orden aleatorio
36
37     }
38   }
39 }

```

```

28
29     }
30
31     }
32     stopImplicitCluster()
33   }
34 }
35
36
37 if(p == 2) { #comprobando si se asigno 2 nucleos
38   for(nuc in nucleos:1) { #Variacion el numero de nucleos
39     registerDoParallel(makeCluster(detectCores() - nuc))
40
41   for (r in 1:replicas) { #Repetir el experimento cuantas veces la variable replica indique
42
43     if(nuc == 1) {
44       Bot1 <- c(Bot1, system.time(foreach(n = original2, .combine=c) %dopar% primo(n))[3]) # de
45         menor a mayor
46       Bit1 <- c(Bit1, system.time(foreach(n = invertido2, .combine=c) %dopar% primo(n))[3]) # de
47         mayor a menor
48       Bat1 <- c(Bat1, system.time(foreach(n = sample(original2), .combine=c) %dopar% primo(n))[3])
49         # orden aleatorio
50
51     }
52     if(nuc == 2) {
53       Bot2 <- c(Bot2, system.time(foreach(n = original2, .combine=c) %dopar% primo(n))[3]) # de
54         menor a mayor
55       Bit2 <- c(Bit2, system.time(foreach(n = invertido2, .combine=c) %dopar% primo(n))[3]) # de
56         mayor a menor
57       Bat2 <- c(Bat2, system.time(foreach(n = sample(original2), .combine=c) %dopar% primo(n))[3])
58         # orden aleatorio
59
60     }
61
62   }
63   stopImplicitCluster()
64 }
65
66
67
68 if(p == 3) { #comprobando si se asigno 3 nucleos
69   for(nuc in nucleos:1) { #Variacion el numero de nucleos
70     registerDoParallel(makeCluster(detectCores() - nuc))
71
72   for (r in 1:replicas) { #Repetir el experimento cuantas veces la variable replica indique
73
74     if(nuc == 1) {
75       Cot1 <- c(Cot1, system.time(foreach(n = original3, .combine=c) %dopar% primo(n))[3]) # de
76         menor a mayor
77       Cit1 <- c(Cit1, system.time(foreach(n = invertido3, .combine=c) %dopar% primo(n))[3]) # de
78         mayor a menor
79       Cat1 <- c(Cat1, system.time(foreach(n = sample(original3), .combine=c) %dopar% primo(n))[3])
80         # orden aleatorio
81
82     }
83     if(nuc == 2) {
84       Cot2 <- c(Cot2, system.time(foreach(n = original3, .combine=c) %dopar% primo(n))[3]) # de
85         menor a mayor

```

```

82 Cit2 <- c(Cit2, system.time(foreach(n = invertido3, .combine=c) %dopar % primo(n))[3]) # de
83     mayor a menor
84 Cat2 <- c(Cat2, system.time(foreach(n = sample(original3), .combine=c) %dopar % primo(n))[3])
85     # orden aleatorio
86 }
87 if(nuc == 3) {
88     Cot3 <- c(Cot3, system.time(foreach(n = original3, .combine=c) %dopar % primo(n))[3]) # de
89         menor a mayor
90     Cit3 <- c(Cit3, system.time(foreach(n = invertido3, .combine=c) %dopar % primo(n))[3]) # de
91         mayor a menor
92     Cat3 <- c(Cat3, system.time(foreach(n = sample(original3), .combine=c) %dopar % primo(n))[3])
93         # orden aleatorio
94 }
95 stopImplicitCluster()
96 }
97 }
98 }

```

### 3. Resultados

Se puede observar en la figura 1 que el tiempo de ejecución es menor cuando se usa más cantidad de núcleos para hacer las tareas, también que al utilizar menor proporción de números primos el tiempo es menor.

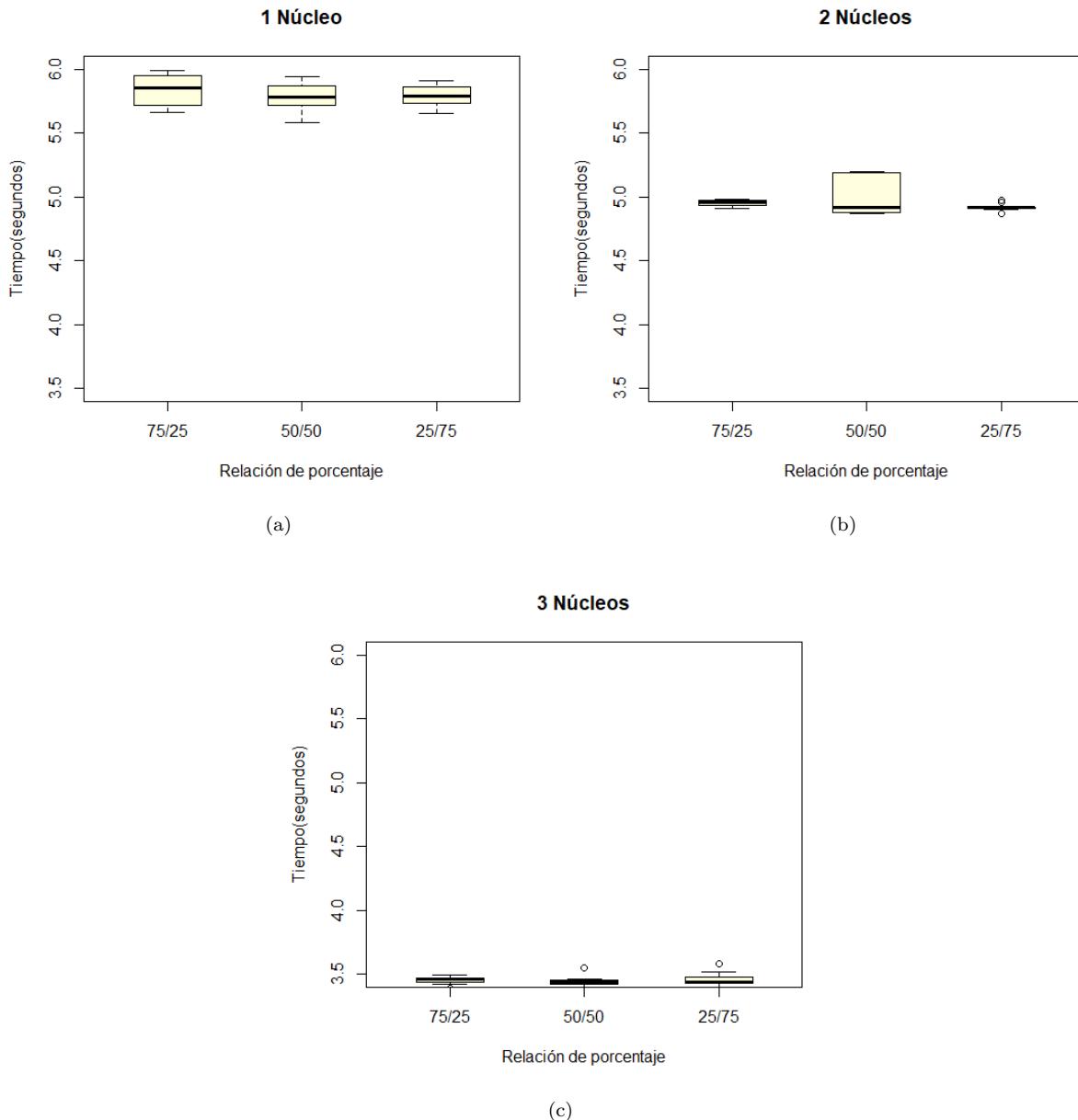


Figura 1: Tiempo en que tardan en ejecutarse las tareas variando los núcleos y las proporciones.

Como se puede observar el valor de  $p$  obtenido es mayor de 0.05 lo que indica que las proporciones de números primos no tienen un efecto significativo en el tiempo de ejecución<sup>2</sup>.

<sup>1</sup> Kruskal-Wallis rank sum test

<sup>2</sup> Kruskal-Wallis chi-squared = 19.227, df = 12, p-value = 0.08321

## 4. Conclusión

Lo que afecta considerablemente al tiempo de ejecución es el número de núcleos usados y no la proporción de números primos y no primos, ya que entre más núcleos estén trabajando las tareas se realizan más rápidamente, es mejor que trabajen tres núcleos a que sólo un núcleo esté trabajando.

## Referencias

- [1] Edson Cepeda. Práctica 3, 2019. URL <https://sourceforge.net/projects/systemssimulation/files/P3/>.
- [2] Yessica Reyna. Práctica 3, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/files/Practica3/>.
- [3] Elisa Schaeffer. Practica3, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.

# Práctica 4: Diagramas de Voronoi

1445183

7 de junio de 2019

## 1. Retroalimentación

En esta práctica aún se tuvo errores en redacción, biografía, así como en formato de figuras, se hacen esas correcciones además de agregar la sección modificada del código en R.

# Práctica 4: Diagramas de Voronoi

1445183

19 de febrero de 2019

## 1. Introducción

Se le llama diagrama de Voronoi a la representación de divisiones de un plano en regiones [3], éstas se forman a partir de un punto en un plano el cual al tomar los puntos más cercanos a él comienza a formar dichas regiones. De esta manera es más fácil diferenciar el espacio entre regiones.

## 2. Objetivo

Examinar el efecto de la relación de número de semillas ( $k$ ) y tamaño de zona ( $n$ ) con respecto al largo de una grieta, tomando en cuenta la cantidad de celdas que contiene una grieta y la mayor distancia Manhattan de la misma [4].

## 3. Descripción

De acuerdo con la práctica 4 [4] se tiene una matriz en la cual se riegan semillas aleatoriamente de manera que crean zonas de diferente tamaño, al producirse una grieta ésta puede tomar dirección hacia el centro o hacia los bordes de la matriz, esto depende de los límites de zona que determinó cada semilla aleatoriamente. Se varió el número de semillas ( $n$ ) tomando los valores 12, 16, y 20, para la dimensión de la matriz se tomaron los valores de 40, 60 y 80. Tomando como referencia el trabajo de Serna[5] y Gámez[2] se usó un vector para la distancia máxima, el cual se introdujo en el vector de resultados (`resultados = c(n, k, replica, largo, distanciamax)`) variando los valores de  $n$  y  $k$  también se modificó el código original de la práctica para obtener el número de celdas [1].

## 4. Resultados

Se muestra una imagen en representación de cada relación  $n, k$  para visualizar la grieta formada. En la matriz 40x40 se observa que el tamaño de grieta es mayor a mayor cantidad de semillas 1 esto se corrobora al observar las medias en las gráficas caja-bigote 2 donde es mayor con un  $k$  de 20.

Para la matriz 60x60 la grieta es grande 3 y profunda con un  $k$  de 20 pero aún está relativamente cerca de los límites de la matriz, siendo más larga la distancia que la matriz 40x40 a la misma  $k$ .

Para la matriz 80x80 se pueden observar grietas de mayor longitud 5 aun en un  $k$  de 12, se puede observar en las gráficas 6 que la distancia no varía mucho entre  $k=16$  y  $k=20$  aunque la cantidad de celdas ocupadas por la grieta sea grande.

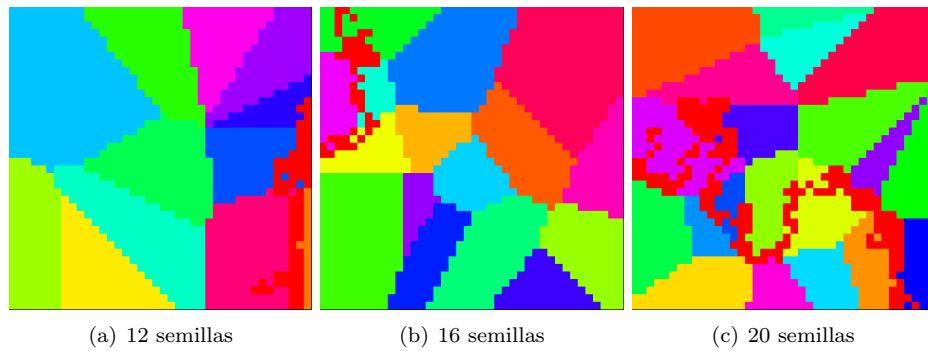


Figura 1: Grietas en matriz  $40 \times 40$  con diferente cantidad de semillas

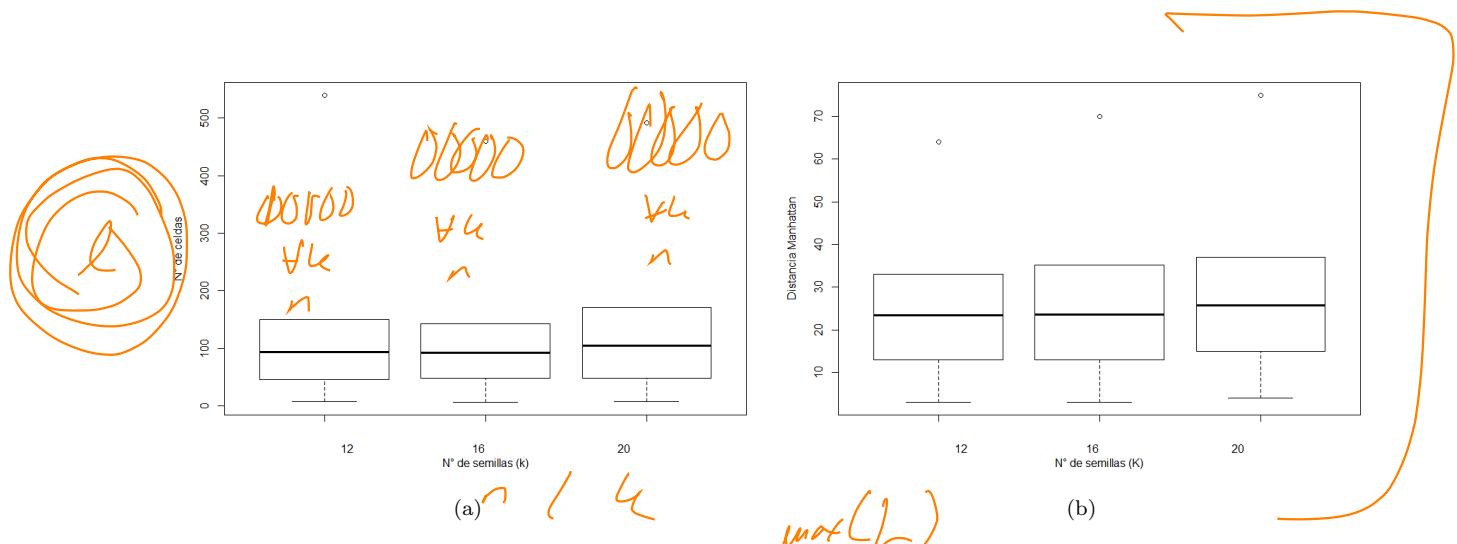


Figura 2: Gráficas caja-bigote para número de celdas y distancia Manhattan vs. número de semillas en matriz de  $40 \times 40$

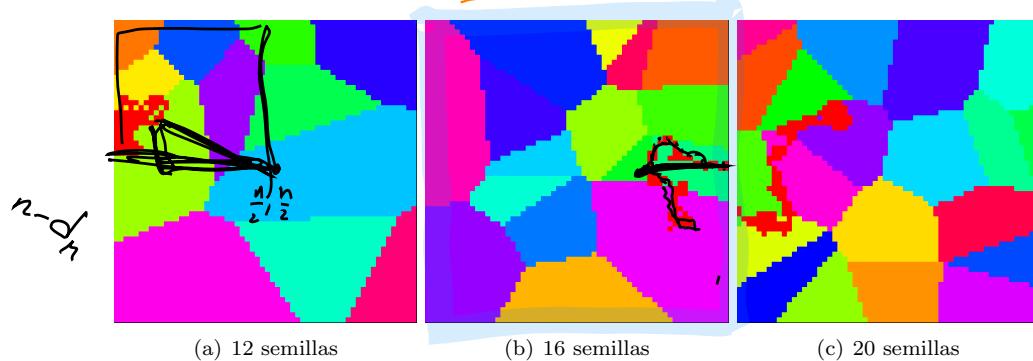


Figura 3: Grietas en matriz  $60 \times 60$  con diferente cantidad de semillas

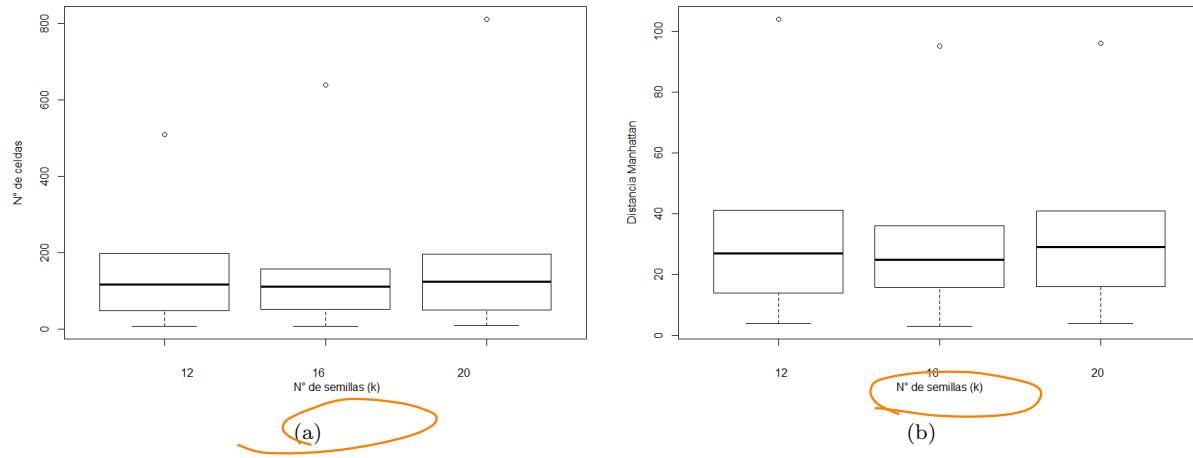


Figura 4: Gráficas caja-bigote para número de celdas y distancia Manhattan vs. número de semillas en matriz de 60x60

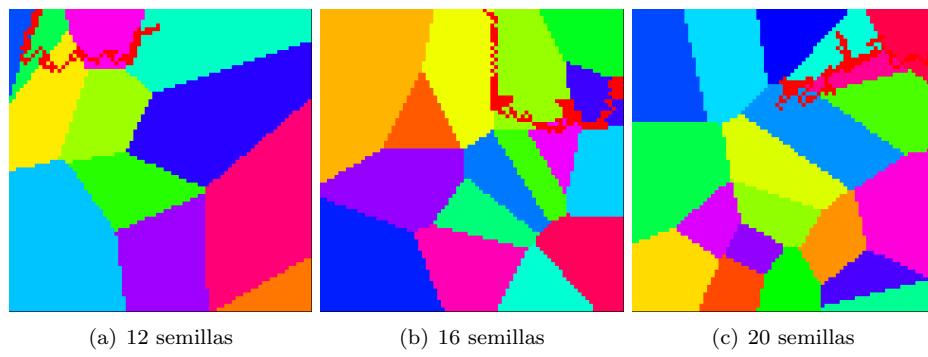


Figura 5: Grietas en matriz 80x80 con diferente cantidad de semillas

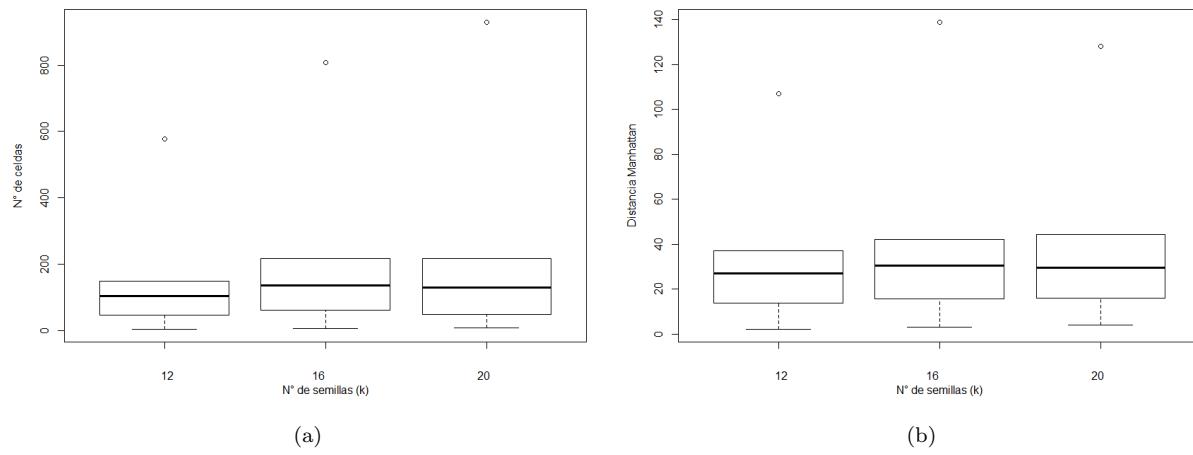


Figura 6: Gráficas caja-bigote para número de celdas y distancia Manhattan vs. número de semillas en matriz de 80x80

## 5. Conclusiones

De acuerdo con los resultados a una relación de menor n y mayor k en este caso 40 y 20, la grieta se propaga de manera más fácil hacia el centro de la matriz a causa de haber más fronteras por donde pasar y un espacio angosto en el cual crecer. Así mismo, a una relación de mayor n y menor k, en este caso 80 y 12, el tamaño de grieta y su propagación hacia el centro no son significativos, debido a que el espacio es demasiado amplio y hay menor cantidad de fronteras por las cuales la grieta pueda crecer.

## Referencias

- [1] Dulce Carrasco. Repositorio simulación, 2019. URL <https://github.com/DulceCarrasco/SimulacionNANO>.
- [2] A. Gámez. p4, 2018. URL <https://sourceforge.net/projects/simulacionr/files/P4/>.
- [3] Clara Grima. Diagramas de Voronoi, 2017. URL [https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101\\_noticia.html](https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101_noticia.html).
- [4] Elisa Schaeffer. Practica 4, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p4.html>.
- [5] JA Serna Mendoza. Práctica 4, 2018. URL <https://sourceforge.net/projects/simulacionenr/files/P4/>.



# Práctica 4: Diagramas de Voronoi

1445183

4 de junio de 2019

## 1. Introducción

Se le llama diagrama de Voronoi a la representación de divisiones de un plano en regiones **[1]**, éstas se forman a partir de un punto en un plano el cual al tomar los puntos más cercanos a él comienza a formar dichas regiones. De esta manera es más fácil diferenciar el espacio entre regiones.

## 2. Objetivo

Examinar el efecto de la relación de número de semillas  $k$  y tamaño de zona  $n$  con respecto al largo de una grieta, tomando en cuenta la cantidad de celdas que contiene una grieta y la mayor distancia Manhattan de la misma **[2]**.

## 3. Descripción

De acuerdo con la práctica **[2]** se tiene una matriz en la cual se riegan semillas aleatoriamente de manera que crean zonas de diferente tamaño como se muestra en la figura **[1]**, al producirse una grieta ésta puede tomar dirección hacia el centro o hacia los bordes de la matriz, esto depende de los límites de zona que determinó cada semilla aleatoriamente.

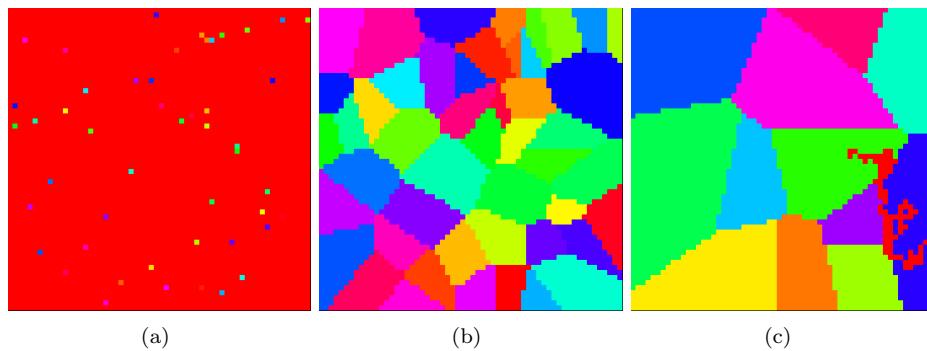


Figura 1: a) siembra de semillas, b) crecimiento de zonas, c) crecimiento de grieta.

Tomando como referencia el trabajo de Serna**[3]** y Guajardo**[?]** se varía el número de semillas  $k$  tomando los valores 12, 24, 36 y 48, para la dimensión de la matriz  $n$  se toman los valores de 15, 30, 45 y 60.

```
[1] n <- c(15,30,45,60)
[2] k <- c(12,24,36,48)
```

```

3| datos <- data.frame()
4| Largo <- c()
5| Manhattan <- TRUE
6| replicas <- 20
7|
8| for (mat in n) {
9|   for (sem in k) {
10|     zona <- matrix(rep(0, n * n), nrow = n, ncol = n)
11|     x <- rep(0, sem)
12|     y <- rep(0, sem)
13|     for (semilla in 1:k) {
14|       while (TRUE) {
15|         fila <- sample(1:n, 1)
16|         columna <- sample(1:n, 1)
17|         if (zona[fila, columna] == 0) {
18|           zona[fila, columna] = semilla
19|           x[semilla] <- columna
20|           y[semilla] <- fila
21|           break
22|         }
23|       }
24|     }
25|   }
26| }
27|
28| sum <- c(n, k, summary(largos))
29| datos <- rbind(datos, sum)
30| Largo <- c(Largo, largos)
31| colnames(datos) <- c("n", "k", "Min", "Q1", "Median", "Mean", "Q3", "Max")
32}
33}

```

## 4. Resultados

En la figura 2 se puede observar que para la matriz de 30 la distancia manhattan y el largo de grieta es mayor de manera que la distancia de la grieta hacia el borde disminuye, en cambio para la matriz de 60 la distancia manhattan y la grieta es menor por lo que la distancia de la grieta hacia el borde es mayor.

Al realizar una prueba estadística [?] de los datos se obtiene que las diferentes combinaciones de  $n$  y  $k$  son significativas, de manera que se puede decir que la mejor combinación para la distancia máxima es de 36  $k$  y 60  $n$

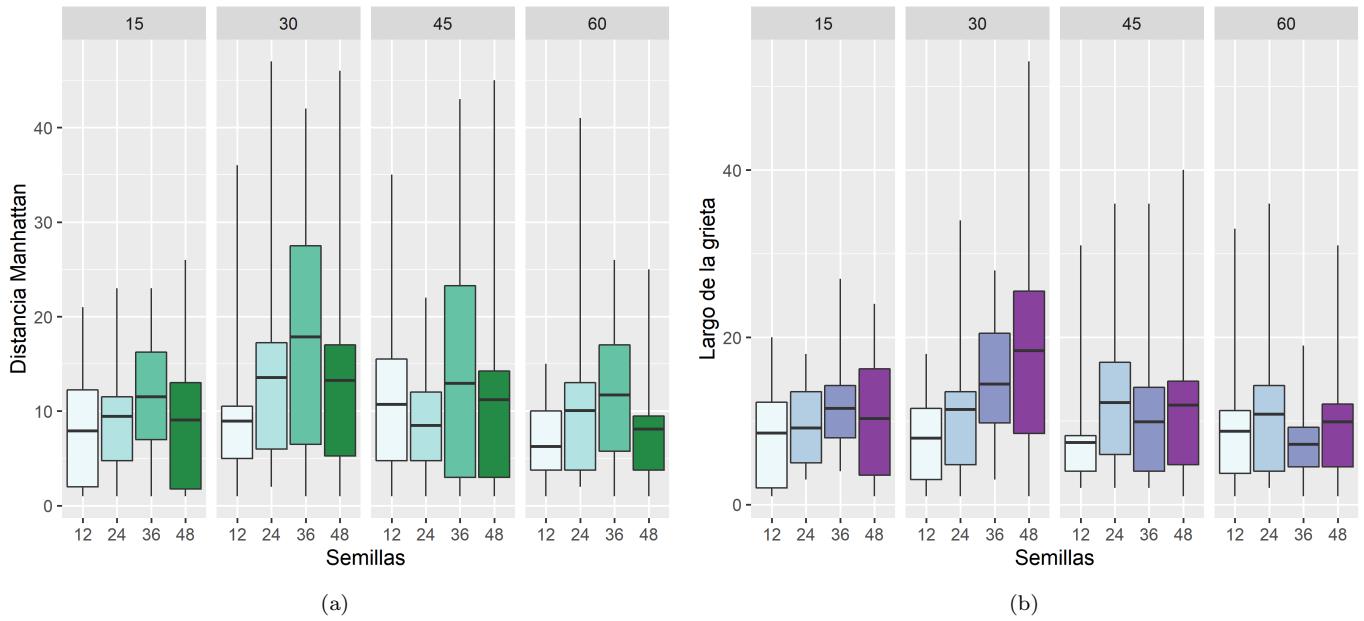


Figura 2: Distancia manhattan y largo de grieta para diferentes combinaciones de semillas  $k$  y matriz  $n$ .

```

1 Response: prueba
2 Df Sum Sq Mean Sq F value Pr(>F)
3 group      15  949.0   63.269  0.4443  0.9586 #grupo 15, 36 semillas y matriz de 60
4 Residuals  64 9113.7 142.402
5 alpha= 1

```

Como se muestra en la figura 3 se observa el crecimiento máximo de la grieta en tres repeticiones del experimento para la combinación de 36  $k$  y 60  $n$ .

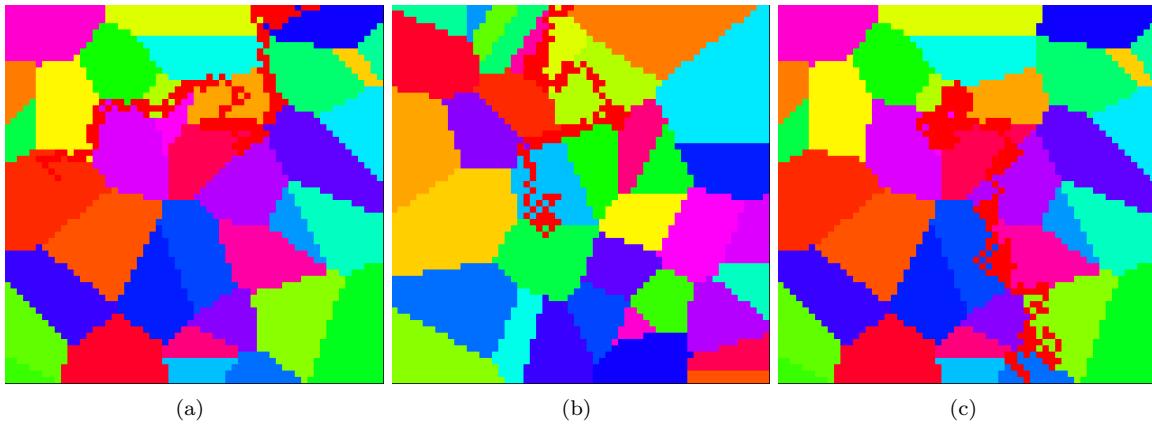


Figura 3: Distancia máxima de grieta para la relación de 39  $k$  y 60  $n$ .

## 5. Conclusiones

Al tener mayor cantidad de semillas se generan más zonas, por lo tanto existen más límites de zonas, la grieta se propaga hacia el centro de manera más sencilla en una matriz pequeña que en una grande.

Para este caso puntual la grieta se propagó de mejor manera en una relación de semilla y matriz con valores grandes, la grieta se posicionó en el centro de la matriz, teniendo una distancia máxima entre los ejes [25,25].

## Referencias

- [1] Clara Grima. Diagramas de voronoi, 2017. URL [https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101\\_noticia.html](https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101_noticia.html).
- [2] Elisa Schaeffer. Practica 4, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p4.html>.
- [3] JA Serna Mendoza. Práctica 4, 2018. URL <https://sourceforge.net/projects/simulacionenr/files/P4/>.

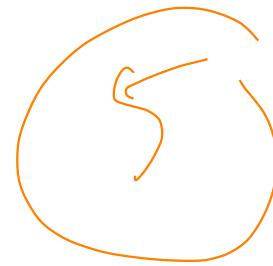
# Práctica 5: Método Monte-Carlo

1445183

7 de junio de 2019

## 1. Retroalimentación

En esta práctica solo era cuestión de hacer una mejora a los resultados, además se agregaron pruebas estadísticas.



# Práctica 5: Método Monte-Carlo

1445183

25 de febrero de 2019

## 1. Introducción

El método Monte-Carlo [2] permite calcular estadísticamente algún valor que no se conoce y que es difícil calcular analíticamente.

## 2. Objetivo

Calcular el valor de la integral (1) proporcionada por la práctica usando el método Monte-Carlo para la función (2), teniendo como referencia el valor de la integral obtenido mediante Wolfram Alpha [1].

$$\int_3^7 f(x) d(x) \quad (1)$$

$$f(x) = \frac{1}{\exp(x) + \exp(-x)} \quad (2)$$

## 3. Descripción

Para la elaboración de la práctica se utilizó el código base proporcionado [3], al cual se le modificó el tamaño de muestra (pedazo) con 5 diferentes valores con 50 repeticiones cada uno, haciendo uso de `for` como se muestra en el código R siguiente:

```
1 for (pedazo in c(100,1000,10000,100000,1000000)) {  
2   print(paste("pedazo", pedazo))  
3   for (repeticiones in 1:50) {  
4     montecarlo <- foreach(i = 1:cuantos, .combine=c) %dopar% parte()  
5     integral <- sum(montecarlo) / (cuantos * pedazo)  
6     resultados <- (pi / 2) * integral  
7     diferencia <- (valor - resultados)  
8     vectores <- rbind(vectores, c(repeticiones, pedazo, valor, resultados, diferencia))  
9   }  
10 }
```

De esta manera se obtienen los valores de la integral para cada tamaño de muestra así como el error de cada uno, expresados en gráficas caja-bigote, para comparación de los resultados se toma en cuenta el valor de la integral obtenido mediante Wolfram Alpha [1].

## 4. Resultados

Como se muestra en la figura 1 la aproximación al valor real 0.048834 de la integral (línea roja) es mayor conforme aumenta el tamaño de muestra lo que también afecta en el error, ya que este disminuye al aumentar el tamaño de muestra.

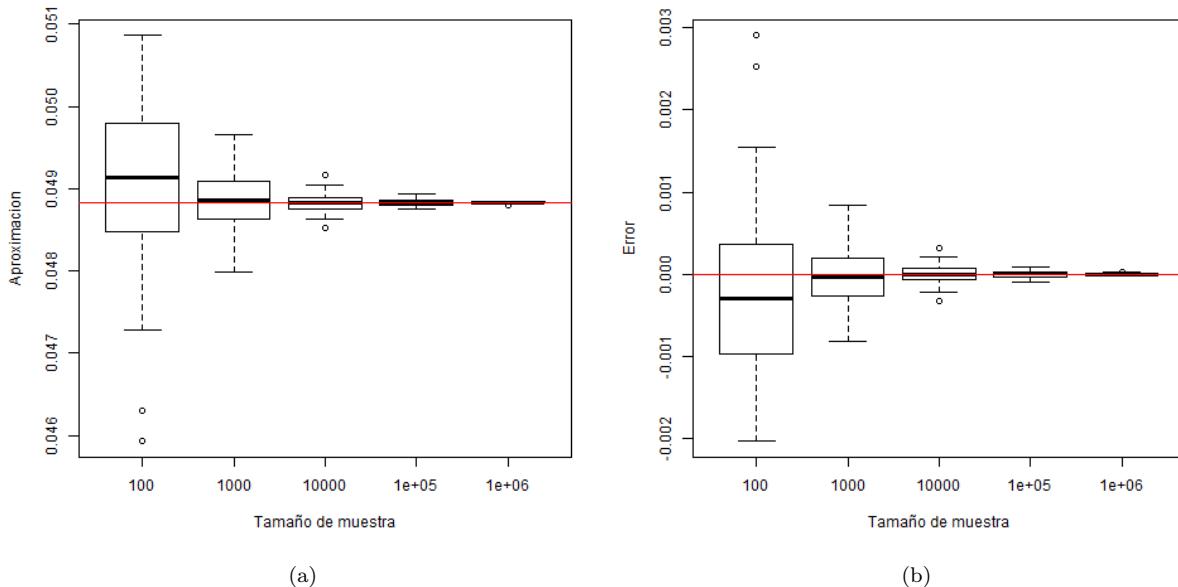
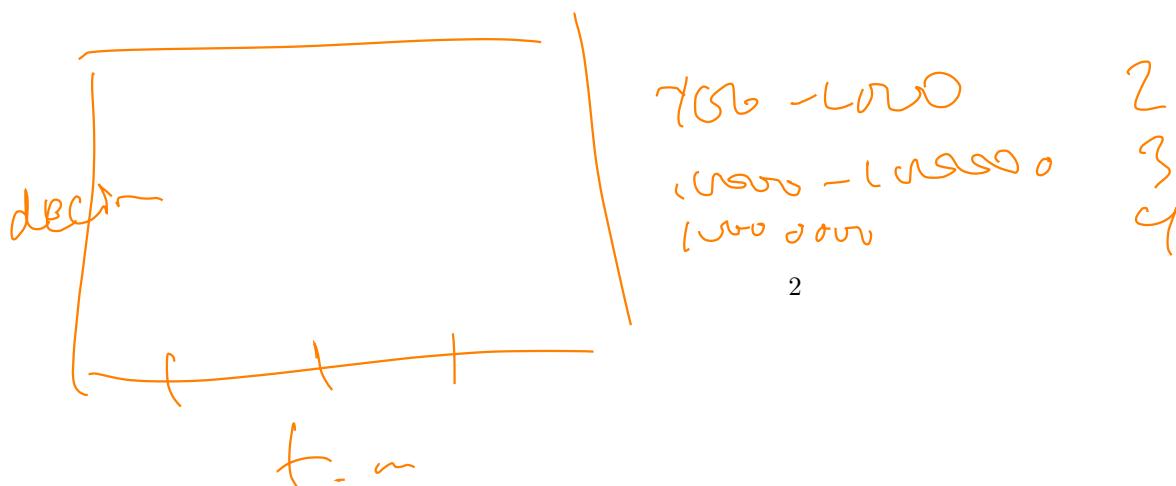


Figura 1: Aproximación al valor real y error vs tamaño de muestra

En el cuadro 1 se puede ver que el tamaño de muestra con mayor valor es el que tiene mayor precisión en más lugares decimales y viceversa con el valor de referencia 0.048834

Cuadro 1: Comparación de decimales por tamaño de muestra

Tamaño de muestra	Valor aproximado	Error
100	0.04611858	0.000003597
1000	0.04931986	-0.0004858631
10000	0.04864882	0.0001851811
100000	0.04879861	0.00003538998
1000000	0.04883115	0.000002852509



## 5. Conclusiones

Mediante el método Monte-Carlo se obtuvieron los valores aproximados al de referencia, para los valores asignados en este reporte el tamaño de muestra con valor de 1,000,000 (valor mayor utilizado) es el valor con más decimales de precisión con el de referencia y también es el valor con menor error. De manera que entre mayor sea el tamaño de muestra, es decir, los puntos dentro de la integral, la precisión de obtener el resultado correcto aumenta.

## Referencias

- [1] Wolfram Alpha, 2019. URL <https://www.wolframalpha.com>.
- [2] Ryan Moulton. Método Monte-Carlo, 2019. URL [https://es.overleaf.com/learn/latex/Integrals,\\_sums\\_and\\_limits](https://es.overleaf.com/learn/latex/Integrals,_sums_and_limits).
- [3] Elisa Schaeffer. Práctica 5: Método Monte-Carlo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p5.html>.

# Práctica 5: Método Monte-Carlo

1445183

7 de junio de 2019

## 1. Introducción

El método Monte-Carlo [3] permite calcular estadísticamente algún valor que no se conoce y que es difícil calcular analíticamente.

## 2. Objetivo

Calcular el valor de la ecuación [1] proporcionada por la práctica usando el método Monte-Carlo para la función [2], teniendo como referencia el valor de la integral obtenido mediante Wolfram Alpha.

$$\int_3^7 f(x)dx \quad (1)$$

Donde,

$$f(x) = \frac{1}{\exp^{(x)} + \exp^{(-x)}} \quad (2)$$

## 3. Descripción

El código base proporcionado [4] se modifica [2] para variar el tamaño de muestra (pedazo) con cuatro diferentes valores (100, 1000, 10000 y 1000000) y con 20 repeticiones cada uno, haciendo uso de `for` como se muestra en el código R siguiente:

```
1 desde <- 3
2 hasta <- 7
3 pedazo <- 50000
4 suppressMessages(library(doParallel))
5 registerDoParallel(makeCluster(detectCores() - 3))
6 parte <- function() {
7   valores <- generador(pedazo)
8   return(sum(valores >= desde & valores <= hasta))
9 }
10
11 resultadoEsp <- (pi/2) * 0.031089
12 resultadoEsp_str <- strsplit(paste(resultadoEsp), split = "") #resultado esperado
13 datos <- c()
14 datos_str <- c()
15
16 replicas <- 20 #Replicas del experimento
17
```

```

18 for(i in 1:replicas) {
19   for(j in c(100,1000,10000,100000)){ #Variacion de las muestras
20     montecarlo <- foreach(i = 1:j, .combine=c) %dopar% parte()
21     stopImplicitCluster()
22     integral <- sum(montecarlo) / (j * pedazo)
23     resultado <- (pi / 2) * integral
24     datos <- c(datos , resultado)
25   }
26 }
27 .
28 .
29 .
30 .

```

Se agrega un vector que esté contando los decimales acertados para cada tamaño de muestra.

```

1 for(i in 1:length(data.matrix(datos))) {
2   resultados <- c(resultados , data.matrix(datos)[i])
3   resultados_str <- c(resultados_str , strsplit(paste(resultados[i]), split=""))
4 }
5
6 for(i in 1:length(resultados)) {
7   contador <- 0
8   for(j in 1:7) {
9     b <- (unlist(resultados_str[i])[j]) == (unlist(resultadoEsp_str)[j])
10    if(b) {
11      contador <- contador + 1
12    } else {
13      break;
14    }
15  }
16 }
17
18 res_contadores <- c(res_contadores , contador -2)
19 }
20 }
21 }
22

```

De esta manera se obtienen los valores de la integral para cada tamaño de muestra así como el error de cada uno, expresados en gráficas caja-bigote, para comparación de los resultados se toma en cuenta el valor de la integral obtenido mediante Wolfram Alpha [1].

## 4. Resultados

Como se muestra en la figura [1] la aproximación al valor real [2] de la integral (0.048834) es mayor conforme aumenta el tamaño de muestra esto se puede observar en la anchura del diagrama de violín para el valor de 100000 y su media.

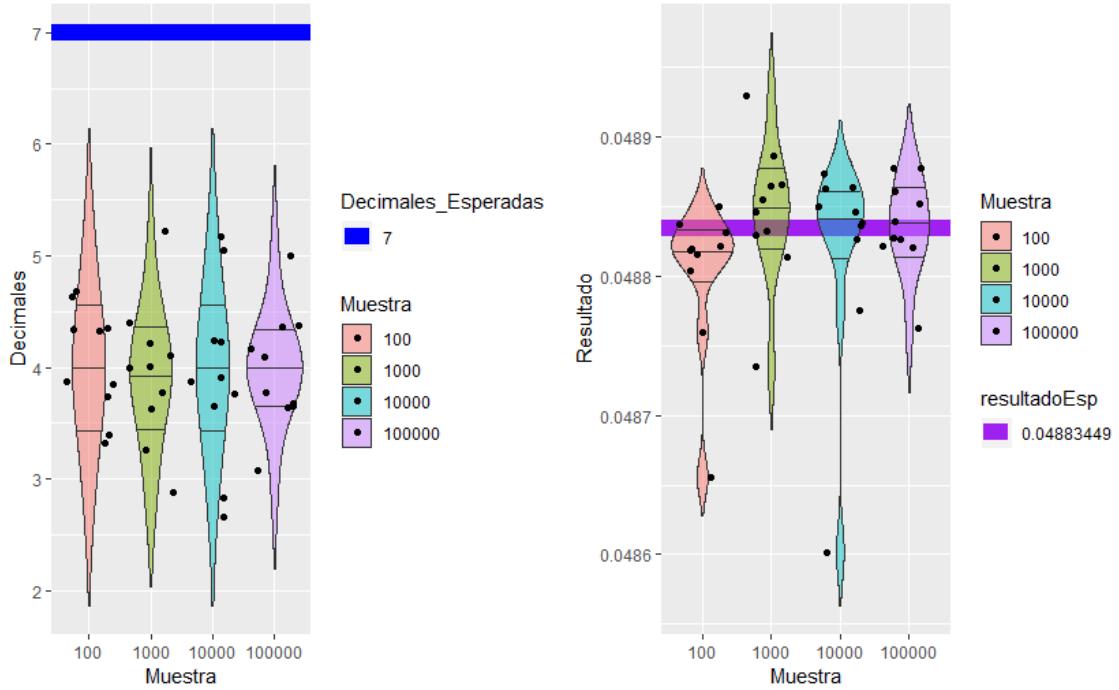


Figura 1: Aproximación al valor de referencia y decimales acertadas.

## 5. Conclusiones

Entre mayor sea el tamaño de muestra, es decir, los puntos dentro de la integral, la precisión de obtener el resultado correcto aumenta.

## 6. Reto 1

El primer reto [4] consiste en implementar la estimación del valor Pi, de Kurt con paralelismo y determinar la relación matemática entre el npumero de muestras obtenidas y la precisión obtenida en términos de cantidad de lugares decimales correctos.

## 7. Descripción

Al código usado anteriormente se modifica [2] para tener las condiciones de obtener el valor de Pi, usando como tamaño de muestra los valores de 100, 1000 y 3000 con 10 réplicas cada uno.

```
1 suppressMessages(library(doParallel))
2 registerDoParallel(makeCluster(detectCores() - 1))
3
4 calculo <- function(generar) {
5   r <- 1 #radio
6   contcirc <- 0 #contador dentro del circulo
7   contcuad <- 0 #contador fuera del circulo
8   num <- generar #muestra
9   for(i in 1:num){
10     x <- runif(1) #valor x dentro del cuadrado
11     y <- runif(1) #valor y dentro del cuadrado
12     d <- (x*x) + (y*y) # distancia
13     if (d < r*r) {
14       contcirc <- contcirc + 1
15     }
16   }
17 }
18 pi <- as.double(4 * (contcirc/num)) #relacion para pi
19 return(pi)
20 }
21
22
23 replicas <- 10 #Numero de replicas del experimento
24 datos <- data.frame()
25 variacion <- c(100,5000,10000) #Variacion de las muestras
26 for(var in variacion) {
27   calcularPi <- foreach(i=1:replicas , .combine = c) %dopar% calculo(var)
28   datos <- rbind(datos, calcularPi)
29 }
```

## 8. Resultados Reto 1

Se puede observar en la figura 2 que la precisión al valor de referencia 3.141593 aumenta conforme incrementa el tamaño de muestra y el número de decimales acertadas al valor de referencia son más para el tamaño de muestra mayor.

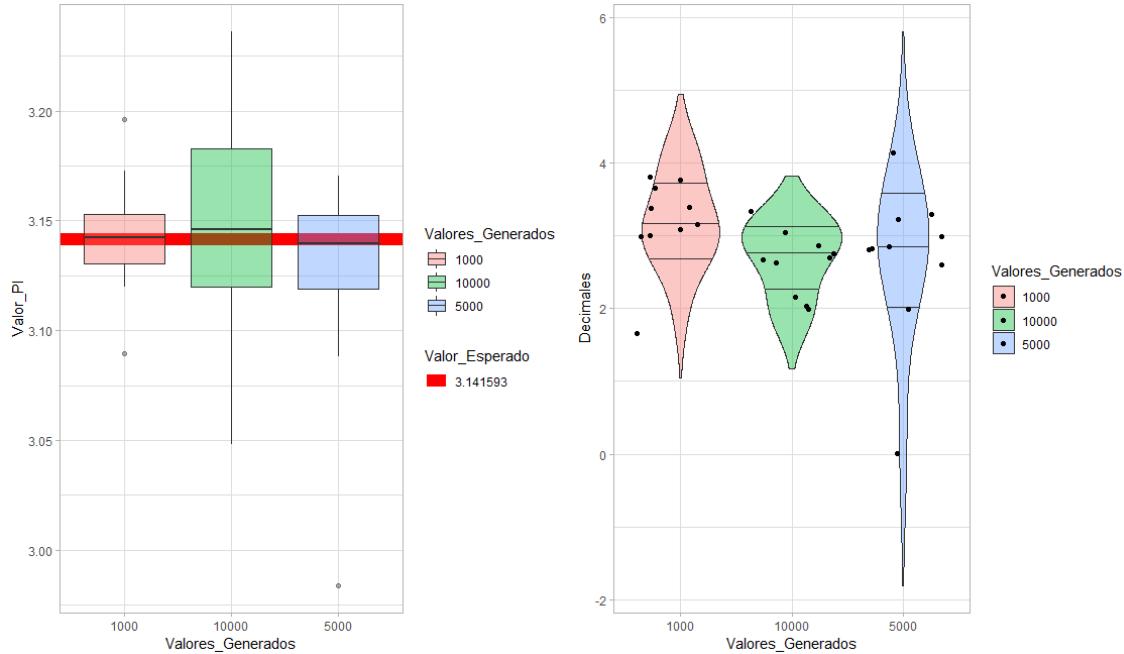


Figura 2: Aproximación al valor de referencia y decimales acertadas.

## 9. Conclusión

Para el método Monte-Carlo es necesario tener un tamaño de muestra grande (puntos) para poder tener mayor precisión al resultado que se quiere llegar. Como se pudo constatar en esta práctica, que el valor fue más preciso cuando se tenía mayor tamaño de muestra.

## Referencias

- [1] Wolfram Alpha. Wolfram Alpha, 2019. URL <https://www.wolframalpha.com>.
- [2] Edson Cepeda. Práctica 5, 2019. URL <https://sourceforge.net/projects/systemssimulation/files/P5/>.
- [3] Ryan Moulton. Método Monte-Carlo, 2019. URL [https://es.overleaf.com/learn/latex/Integrals,\\_sums\\_and\\_limits](https://es.overleaf.com/learn/latex/Integrals,_sums_and_limits).
- [4] Elisa Schaeffer. Práctica 5: Método Monte-Carlo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p5.html>.

# Práctica 6: Sistema multiagente

1445183

7 de junio de 2019

## 1. Retroalimentación

Se agregó el código en R, y se mejoró el formato de la gráfica.

# Práctica 6: Sistema multiagente

S+

1445183

5 de marzo de 2019

## 1. Objetivo

Encontrar el porcentaje máximo de agentes infectados previamente vacunados dentro del sistema multiagente propuesto en esta práctica [1].

## 2. Descripción

El sistema multiagente está compuesto por 50 agentes, los cuales se pueden encontrar en uno de los siguientes estados: *susceptible*, *recuperado* o *infectado*, de manera aleatoria. Los agentes se “vacunan” con una probabilidad  $P_v$  de 0 a 1 con 30 réplicas para cada probabilidad, lo cual afectará a la cantidad de agentes en estado *infectado* finales.

```
1 Pv <- seq(0,1,0.1) #probabilidad
2 for(pv in Pv){ #vacuna
3   for(rep in 1:30){
4     agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(), estado = character()
5       , amigo = NULL)
6     for (i in 1:n) {
7       if(runif(1) < pv){
8         e <- "R"
9       } else if(runif(1) < pi){
10         e <- "I"
11       } else{
12         e <- "S"
13       }
14     }
15   }
16 }
```

### 3. Resultados

En la figura 1 se puede observar el porcentaje máximo de infectados para cada valor de probabilidad mostrando un decrecimiento conforme la probabilidad aumenta. El cuadro 1 muestra más a detalle el valor máximo de porcentaje de infectados para cada probabilidad.

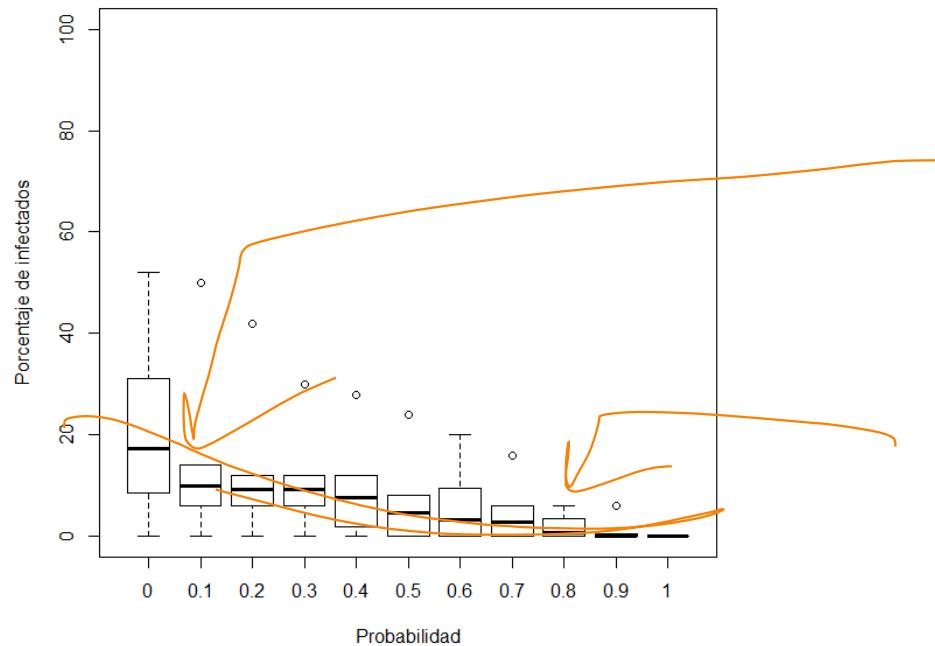


Figura 1: Porcentaje de infectados vs probabilidad

Figura 1: Valores de porcentajes máximos

Probabilidad	Porcentaje máximo de infectados
0	52
0.1	50
0.2	42
0.3	30
0.4	28
0.5	24
0.6	20
0.7	16
0.8	6
0.9	6
0.10	0

## 4. Conclusiones

La cantidad de agentes *infectados* disminuye conforme aumenta la dosis de la vacuna, es decir, al aumentar la probabilidad  $P_v$  aumenta la cantidad de agentes en estado *recuperado* por lo que disminuye la cantidad de agentes *infectados*.

## Referencias

- [1] Práctica 6: Sistema multiagente, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.

# Práctica 6: Sistema multiagente

1445183

5 de junio de 2019

## 1. Objetivo

Encontrar el porcentaje máximo de agentes infectados previamente vacunados dentro del sistema multiagente propuesto en esta práctica [2].

## 2. Descripción

El sistema multiagente está compuesto por 50 agentes, los cuales se pueden encontrar en uno de los siguientes estados: *susceptible*, *recuperado* o *infectado*, de manera aleatoria. Los agentes se “vacunan” con una probabilidad  $P_v$  de 0 a 1 con 20 réplicas para cada probabilidad, lo cual afectará a la cantidad de agentes en estado *infectado* finales.

```
1 l <- 1.5
2 n <- 50
3 pi <- 0.05
4 pr <- 0.02
5 v <- 1 / 30
6 PV <- seq(0, 1, 0.1)
7 r <- 0.1 #umbral
8 t <- 1
9 replicas <- 20
10 Rvacunas <- data.frame()
11 dibujarSistema <- FALSE
12 ggplot <- TRUE
13 pasos <- 30
14
15 for(pv in PV){
16   for(rep in 1:replicas){
17     agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(), estado = character())
18     for(i in 1:n) { #estados iniciales de los agentes
19       if(runif(1) < pv){ #vacunados
20         e <- "R"
21       } else if(runif(1) < pi){
22         e <- "I"
23       } else{
24         e <- "S"
25       }
```

Se generan vectores [1] para saber el estado de cada agente después de la epidemia y se van guardando en un `data.frame` donde después se tomarán los valores máximos infectados según la probabilidad de ser vacunados inicial.

```

1      aS <- agentes[agentes$estado == "S",]
2      aI <- agentes[agentes$estado == "I",]
3      aR <- agentes[agentes$estado == "R",]
4      tl <- paste(tiempo, "", sep="")
5      while (nchar(tl) < digitos) {
6          tl <- paste("0", tl, sep="")
7      }
8
9
10
11     }
12     recu <- dim(agentes[agentes$estado == "R",])[1]
13     if(recu == n){
14         break
15     }
16     t <- t + 1
17 }
18 maximo_infectados <- max(epidemia)
19 porcentaje <- 100 * maximo_infectados / n
20 Rvacunas <- rbind(Rvacunas, c(pv, rep, maximo_infectados, porcentaje))
21 print(pv)
22 }
23 }
```

### 3. Resultados

En la figura 1 se puede observar el porcentaje máximo de infectados para cada valor de probabilidad mostrando un decrecimiento conforme la probabilidad aumenta.

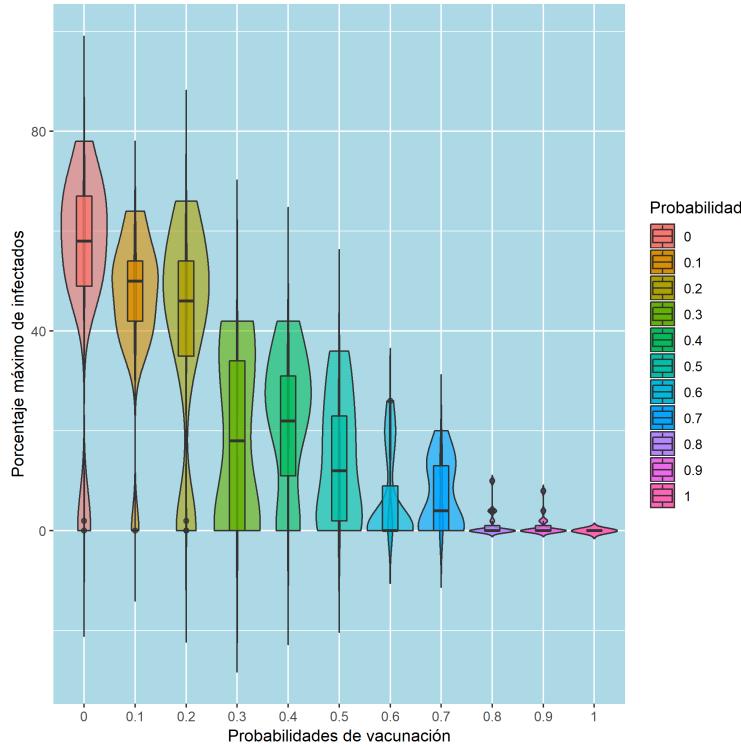


Figura 1: Porcentaje de infectados vs la probabilidad de estado vacunado.

El valor  $p$  obtenido en la prueba estadística es menor al valor de significancia 0.05, esto quiere decir que la probabilidad afecta significativamente al porcentaje de *infectados*.

```
1 Kruskal-Wallis rank sum test
2
3
4 data: Rvacunas$Porcentaje by Rvacunas$Probabilidad
5 Kruskal-Wallis chi-squared = 94.513, df = 10, p-value = 6.792e-16
```

### 4. Conclusiones

La cantidad de agentes *infectados* disminuye conforme aumenta la dosis de la vacuna, es decir, al aumentar la probabilidad  $P_v$  aumenta la cantidad de agentes en estado *recuperado* por lo que disminuye la cantidad de agentes *infectados*.

### Referencias

- [1] Marco Guajardo. Práctica 1, 2019. URL <https://sourceforge.net/p/simulaciondesistemas/code/ci/master/tree/P6/>

- [2] Elisa Schaeffer. Práctica 6: Sistema multiagente, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.

# Práctica 7: Búsqueda local

1445183

7 de junio de 2019

## 1. Retroalimentación

No hubo mejoras.

# Práctica 7: Búsqueda local

1445183



19 de marzo de 2019

## 1. Objetivo

Maximizar la función bidimensional de  $g(x,y)$  encontrando el valor máximo de  $g$  por medio del método de “búsqueda local”.

## 2. Descripción

Para encontrar el valor máximo de la función  $g(x,y)$ , se toma como base el código escrito en la práctica 7 [2], se pide como restricciones que los valores de los ejes  $x$  y  $y$  deben estar entre  $-3$  y  $3$ ; cuando  $x,y$  tienen una “posición actual” tienen ocho vecinos de los cuales tomarán la posición del vecino que tenga el valor mayor, de esta manera el valor de la función  $g$  será el mayor conforme avance en pasos hasta el punto que llegue a tener el valor máximo.

```
1 g <- function(x, y) {  
2   return (((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y^2 - 20 * y)/100)  
3 }  
4  
5 low <- -3  
6 high <- 3  
7 step <- 0.10  
8 puntos <- 15  
9 tmax <- 10^pot
```

Para identificar a los vecinos de  $g(x,y)$  se le añaden al código variables para ubicar la posición y de esta manera se identifique el vecino con mayor valor:

```
1 posiciones <- data.frame(x = double(), y = double(), bestgxy = double())  
2 for (n in 1:puntos) {  
3   currx <- runif(1, low, high)  
4   curry <- runif(1, low, high)  
5   posiciones <- rbind(posiciones, data.frame(x = currx, y = curry, bestgxy = g(currx, curry)))  
6 }
```

*curr <- runif(2, low, high)*

```

1  for (pot in 1:tmax) {
2    resulx <- double()
3    resuly <- double()
4
5    for (n in 1:puntos) {
6      delta <- runif(1, 0, step)
7      deltax <- c(-delta, 0, delta)
8      delta <- runif(1, 0, step)
9      deltay <- c(-delta, 0, delta)
10
11   #codigo de la practica 4
12   vecinosx <- numeric()
13   vecinosy <- numeric()
14   for (dx in deltax) {
15     for (dy in deltay) {
16       if (dx != 0 | dy != 0) { # descartar la posicion misma
17         vecinosx <- c(vecinosx, dx)
18         vecinosy <- c(vecinosy, dy)
19     }
20   }
21
22   tablax <- rbind(resulx, vecinosx + posiciones$x[n])
23   tablay <- rbind(resuly, vecinosy + posiciones$y[n])
24 }
25
26 vecinos <- g(resulx, resuly)
27 maxvalue <- max.col(vecinos)
28
29 for (i in 1:puntos) {
30   posiciones$x[i] <- resulx[i, maxvalue[i]]
31   posiciones$y[i] <- resuly[i, maxvalue[i]]
32 }
33

```

Para visualizar mejor la búsqueda se realizan 15 réplicas puntos, es decir, 15 diferentes valores de  $x, y$  desplazándose hasta llegar a la posición máxima.

Código gal.

### 3. Resultados

Se puede observar en la figura 1 cómo los puntos (valor de  $g$ ) tienden a dirigirse hacia la posición máxima al aumentar la cantidad de pasos [1].

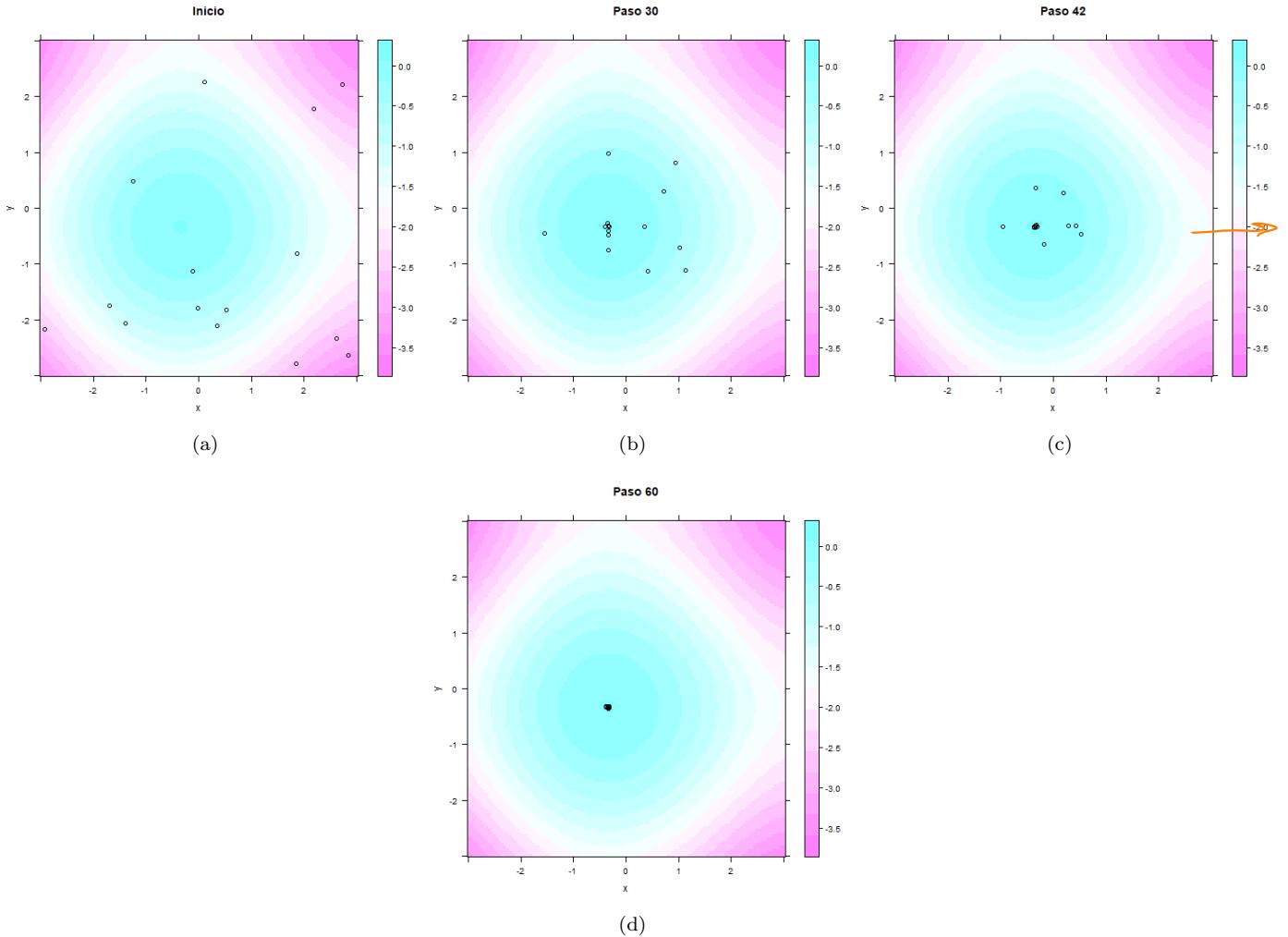


Figura 1: Réplicas simultáneas de la búsqueda

## 4. Conclusiones

Una función puede ser optimizada por medio del método de búsqueda local, la cual puede ser un poco parecida al método Monte-Carlo, ya que se observó que a mayor cantidad de pasos la precisión aumenta llegando a un solo valor, aunque en este caso es el valor **máximo**, es decir, no se puede obtener uno mayor a ese. El valor máximo se encontró dentro de los primeros 60 pasos aunque se esperaba obtener una cantidad de pasos mucho mayor. Si hay algún error probablemente sea en los vectores designados para encontrar a los vecinos.

## Referencias

- [1] Dulce Carrasco. Práctica 7: Búsqueda local, 2019. URL <https://media.giphy.com/media/3Hz3I7TsfkC32k0mEc/giphy.gif>.
- [2] Elisa Schaeffer. Práctica 7: Búsqueda local, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p7.html>.

# Práctica 8: Modelo de urnas

1445183

7 de junio de 2019

## 1. Retroalimentación

No hubo mejoras.

# Práctica 8: Modelo de urnas



1445183

26 de marzo de 2019

## 1. Objetivo

Paralelizar el código proporcionado [1] y medir el tiempo que se ahorra con la paralelización, observar si el ahorro es estadísticamente significativo para diferentes combinaciones de  $k$  y  $n$ , donde  $k$  es tamaño de cúmulo y  $n$  número de partículas.

## 2. Descripción

Para medir el tiempo se hace uso de `Sys.time()`, después se hacen vectores para los diferentes valores de  $k$  y  $n$  con 30 réplicas, recopilando los datos obtenidos en un `data.frame` llamado `resultado`:

```
1 resultado<-data.frame()
2 k <- 10000
3 n <- 1000000
4 #cumulo <- c(1000, 10000)
5 #particula <- c(1000000, 10000000, 100000000)
6 #for (k in cumulo) {
7 #  for (n in particula) {
8   for (replicas in 1:30) {
9     inicial<- Sys.time()
10    originales <- rnorm(k)
11    cumulos <- originales - min(originales) + 1
12    cumulos <- round(n * cumulos / sum(cumulos))
13    assert(min(cumulos) > 0)
14    diferencia <- n - sum(cumulos)
15    if (diferencia > 0) {
16      for (i in 1:diferencia) {
17        p <- sample(1:k, 1)
18        cumulos[p] <- cumulos[p] + 1
19      }
20    } else if (diferencia < 0) {
21      for (i in 1:-diferencia) {
22        p <- sample(1:k, 1)
23        if (cumulos[p] > 1) {
24          cumulos[p] <- cumulos[p] - 1
25        }
26      }
27    }
```

Para la prueba estadística se hace uso de `qqplot` para normalizar los valores.

### 3. Resultados

Se puede ver en la figura 1 la diferencia de tiempo usando los valores dados por la práctica de  $k$  y  $n$  siendo el código paralelizado el que presenta menor tiempo de compilación.

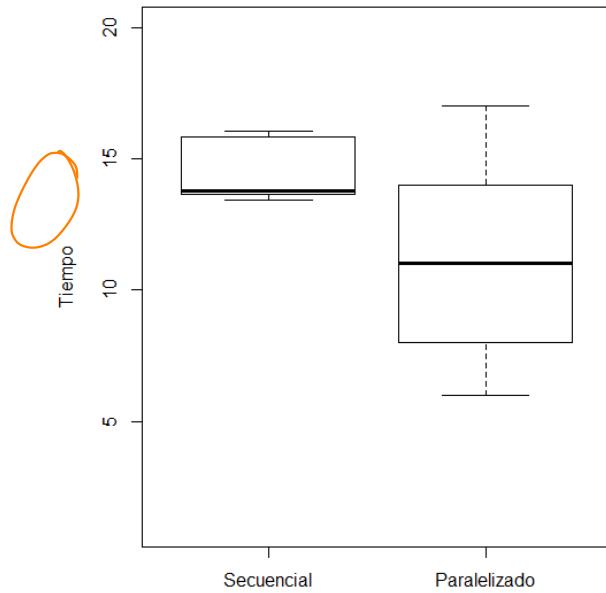


Figura 1: comparación de tiempos

En la figura 2 se puede observar que el tiempo es menor cuando se tiene una relación de  $k$  y  $n$  con los valores más bajos, las combinaciones de  $k$  y  $n$  se proporcionan en el cuadro 1.

Combinación	$k$	$n$
1	1000	1000000
2	1000	10000000
3	1000	100000000
4	10000	1000000
5	10000	10000000
6	10000	100000000

En la figura 3 se observa la normalización del experimento paralelizado, cúmulos ( $k$ ) de 1000 y 10000 con las 3 combinaciones de  $n$  correspondientes. Donde la relación de  $k$  y  $n$  con valores mayores dan una mejor normalización.

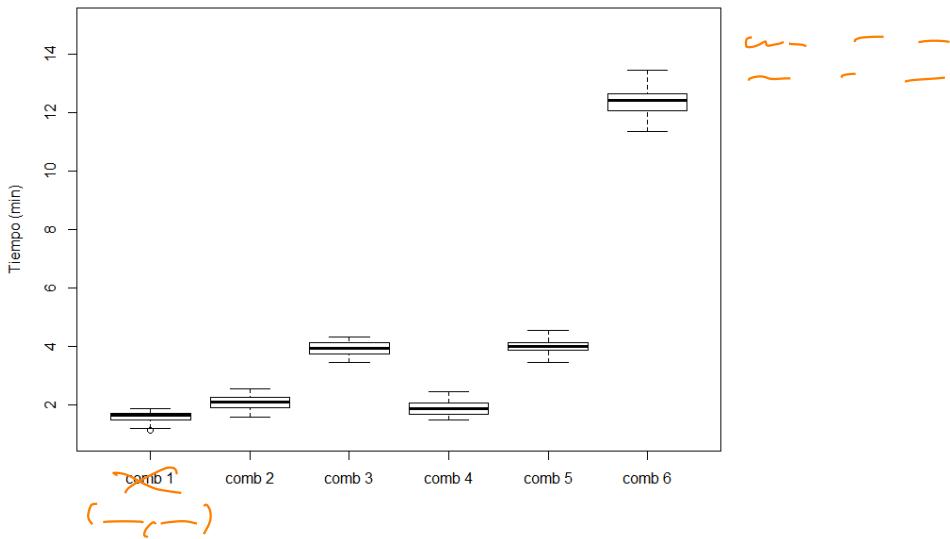


Figura 2: tiempo vs combinaciones de  $k$  y  $n$

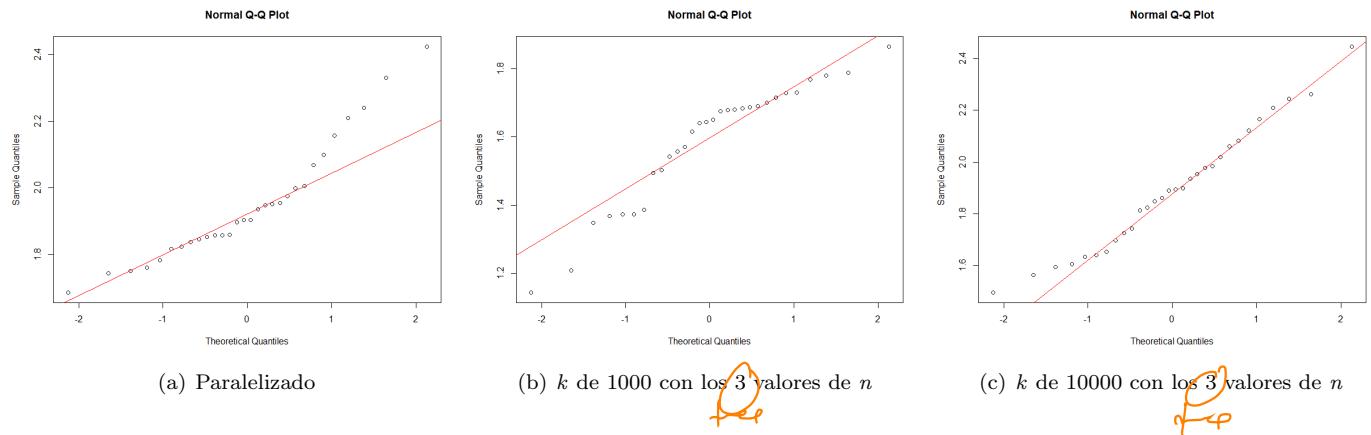


Figura 3: Normalización de resultados

## 4. Conclusiones

Al tener una relación con valores de  $k$  y  $n$  pequeños el tiempo es menor pero los valores no son muy significativos, no se normalizan, y al tener una relación de  $k$  y  $n$  con valores grandes mejora la normalización pero el tiempo es mayor. Si se paraleliza el código aún más, se podrían obtener mejores tiempos al igual que un resultado más significativo.

c1

Reto 1

sección

## 5. Descripción

Para el *reto 1* se necesita saber en cuál paso los cúmulos son suficientemente grandes, usando los valores para  $k$  y  $n$  proporcionados por el código de la práctica [1] se obtienen los valores máximos de los cúmulos usando `max(cumulos)` y por medio de los histogramas se observan los pasos, se hacen 30 réplicas y se obtienen los datos estadísticos con `qqplot`.

## 6. Resultados

En la figura 4 se observan los valores máximos de cúmulos normalizados y en la figura 5 se puede observar que en el paso 3 hay mayor cantidad de cúmulos grandes que ya pueden filtrarse.

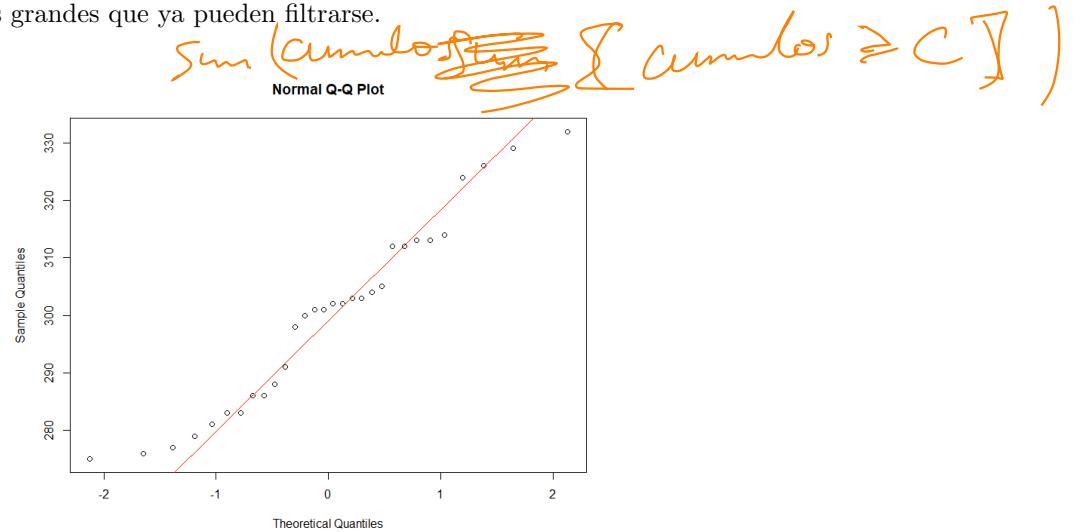


Figura 4: normalización de valores máximos de cúmulos

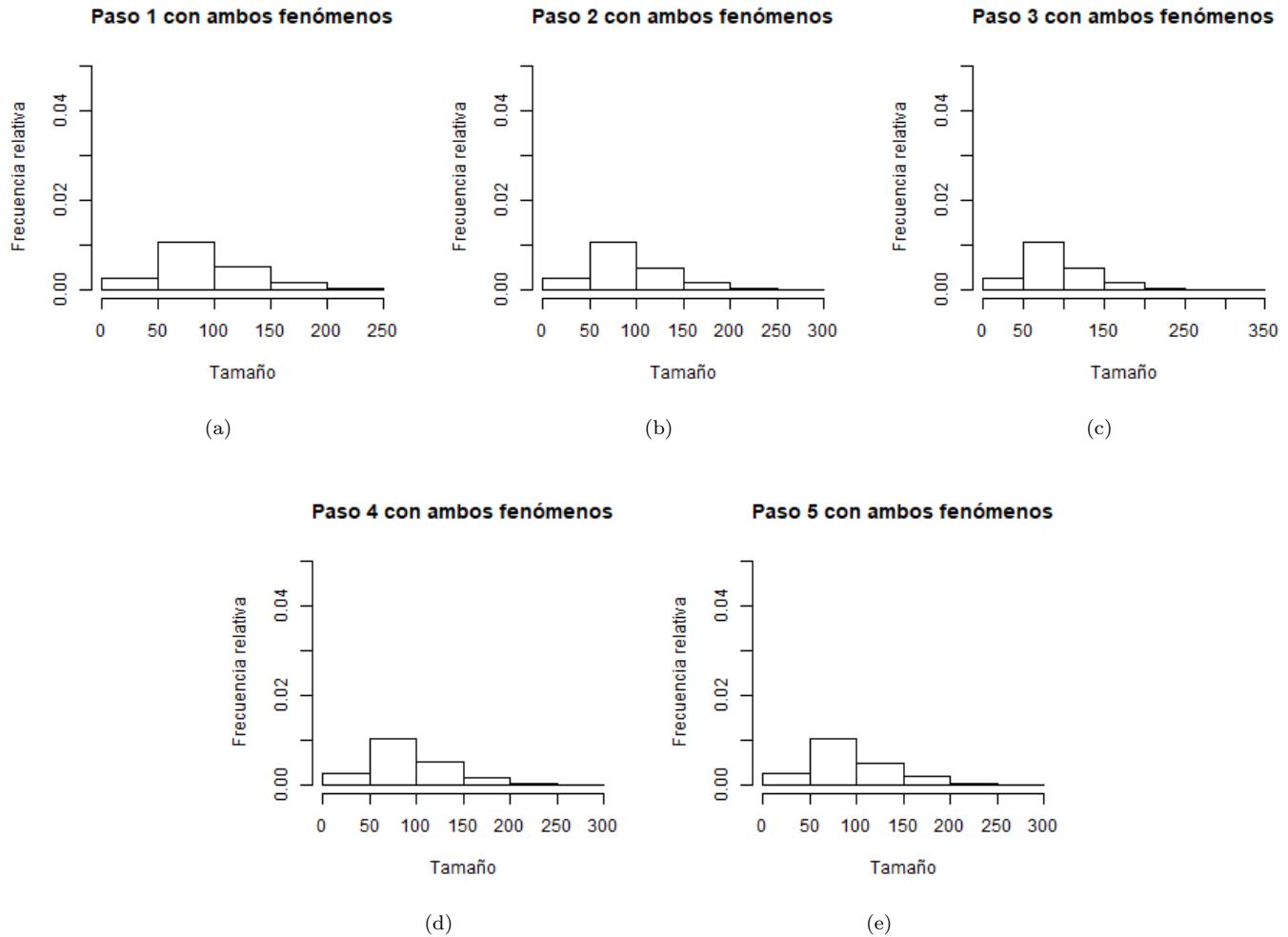


Figura 5: Histograma de pasos  
(frecuencia vs tamaño de cúmulos)

Reto 2

## 7. Descripción

Para determinar la importancia del valor de  $c$  (valor crítico), en el código modificado para el *Reto 1* se cambia su valor original dándole un valor de 40 y se grafica en una curva sigmoidal.

## 8. Resultados

En la figura 6 se observa que al darle a  $c$  un valor de 40, la probabilidad de que se formen cúmulos que ya no se rompan. En la figura 7 se confirma en crecimiento de cúmulos a mayor cantidad de pasos.

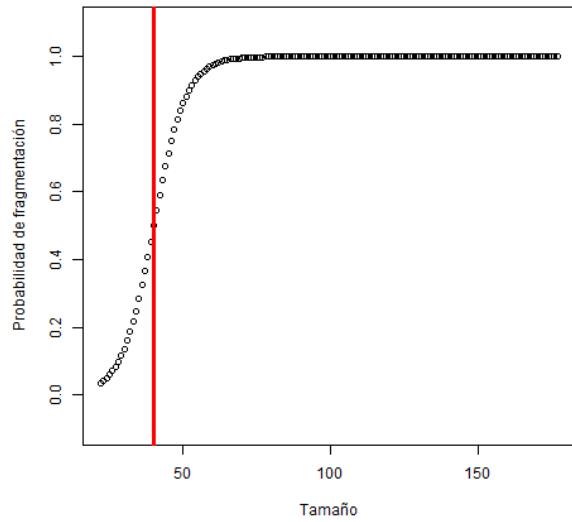


Figura 6: Curva sigmoidal

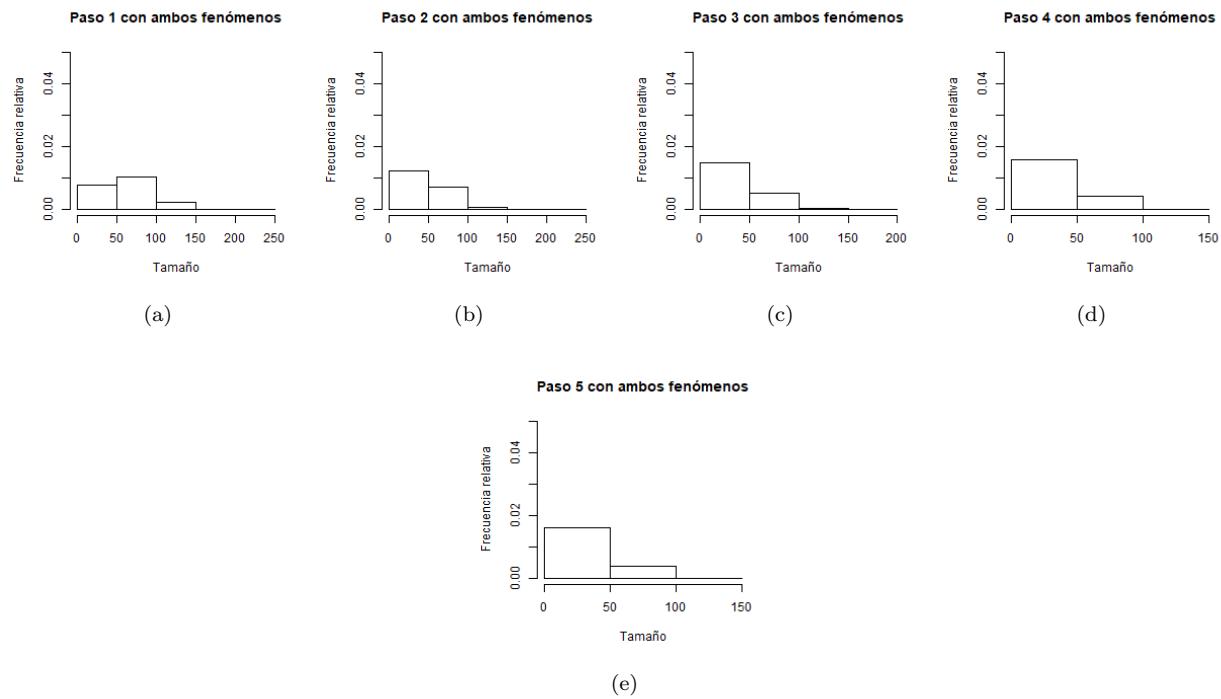


Figura 7: Histograma de pasos  
(frecuencia vs tamaño de cúmulos)

## 9. Conclusiones

El valor de  $c$  afecta en la probabilidad de que los cúmulos se rompan o no, esto afecta en cuál paso se tendrán cúmulos de tamaño grande para poder filtrarlos y por lo tanto los tiempos varían también, por ejemplo, en este caso de  $c=40$ , hay mayor probabilidad de que los cúmulos no se rompan, por lo que disminuirá el tiempo ya que no se tendrán que unir cúmulos rotos y se podrán filtrar más rápido.

## Referencias

- [1] Elisa Schaeffer. Práctica 8: modelo de urnas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p8.html>.

# Práctica 9: Interacciones entre partículas

1445183

7 de junio de 2019

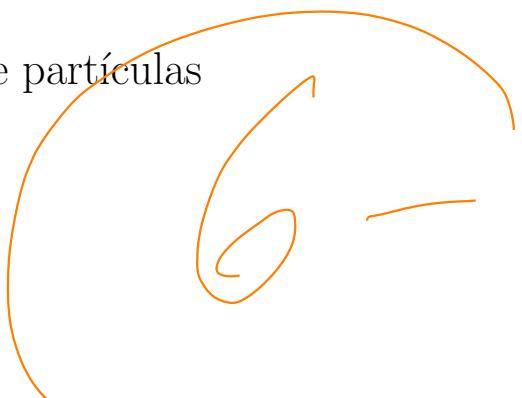
## 1. Retroalimentación

En esta práctica faltaba discutir sobre los efectos entre *carga*, *masa* y *velocidad* además de agregar un gráfico donde se relacionaran.

# Práctica 9: interacciones entre partículas

1445183

2 de abril de 2019



## 1. Objetivo

Agregar masa a cada partícula generada en el código proporcionado por la *práctica 9* lo cual genera atracciones además de las fuerzas causadas por las cargas y estudiar la distribución de las velocidades de las partículas, verificando gráficamente la relación entre la velocidad, la carga y la masa de las partículas.

## 2. Descripción

La masa se agrega siguiendo el ejemplo del código dado para agregar carga a las partículas [2] y se añade al `data.frame`, en la figura 1 se puede observar las partículas generadas con masa y carga:

```
1 p <- data.frame(x = rnorm(n), y=rnorm(n), c=rnorm(n), m=rnorm(n))
2 mmax <- max(p$m)
3 mmin <- min(p$m)
4 p$m <- ((p$m - mmin)*(p$m - mmin) / (mmax - mmin))+1 #masa entre 1 y 5
```

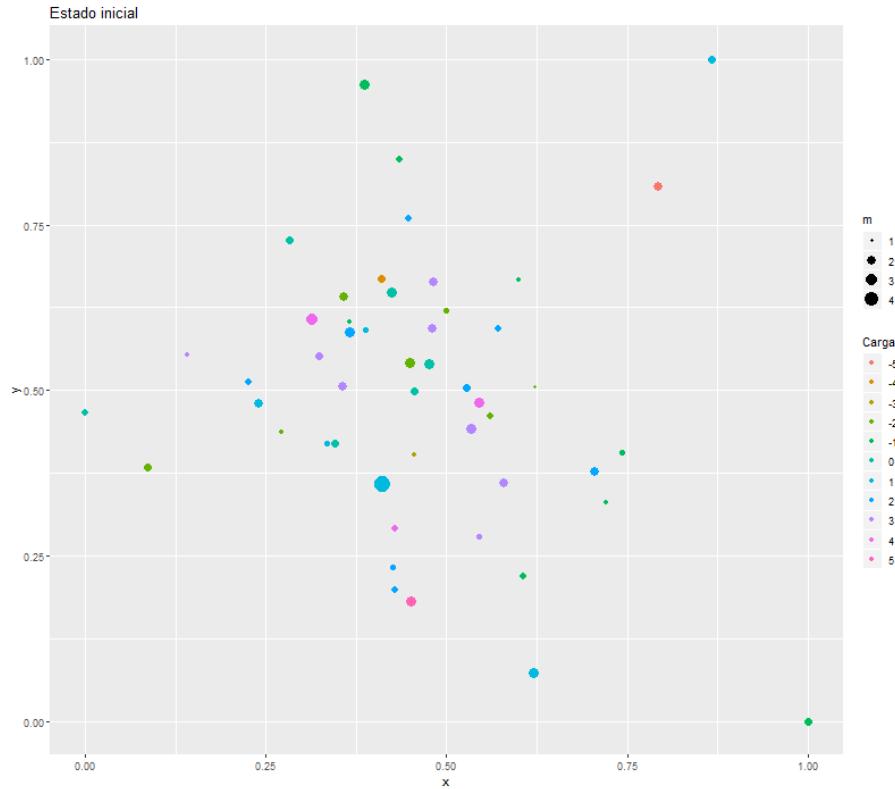


Figura 1: Partículas con carga y masa

Después la masa se añade a la función de **fuerza** para que interactúe junto con la carga al movimiento de las partículas:

```

1 fuerza <- function(i) {
2   xi <- p[i,]$x
3   yi <- p[i,]$y
4   ci <- p[i,]$c
5   mi <- p[i,]$m #masa
6   fx <- 0
7   fy <- 0
8   for (j in 1:n) {
9     cj <- p[j,]$c
10    dir <- (-1)^(1 + 1 * (ci * cj < 0))
11    dx <- xi - p[j,]$x
12    dy <- yi - p[j,]$y
13    factor <- dir * abs(ci - cj) / (sqrt(dx^2 + dy^2) + eps)
14    fx <- fx - dx * factor
15    fy <- fy - dy * factor
16  }
17  return(c(fx, fy)/(mi+1))

```

La velocidad se añade al **data.frame** con un vector que se relaciona con la masa, tomando en cuenta los valores obtenidos para  $x$  y  $y$  obtenidos en el **for** de las iteraciones.

```

1 p$v <- foreach(i = 1:n, .combine=c) %dopar% (xdifmax[i] + ydifmax[i])
2 p$v <- p$m * p$v

```

### 3. Resultados

Para visualizar los resultados se consideran los pasos 1, 25, 50, 75 y 100. En la figura 2 se observa el movimiento de las partículas bajo la influencia de la carga y la masa.

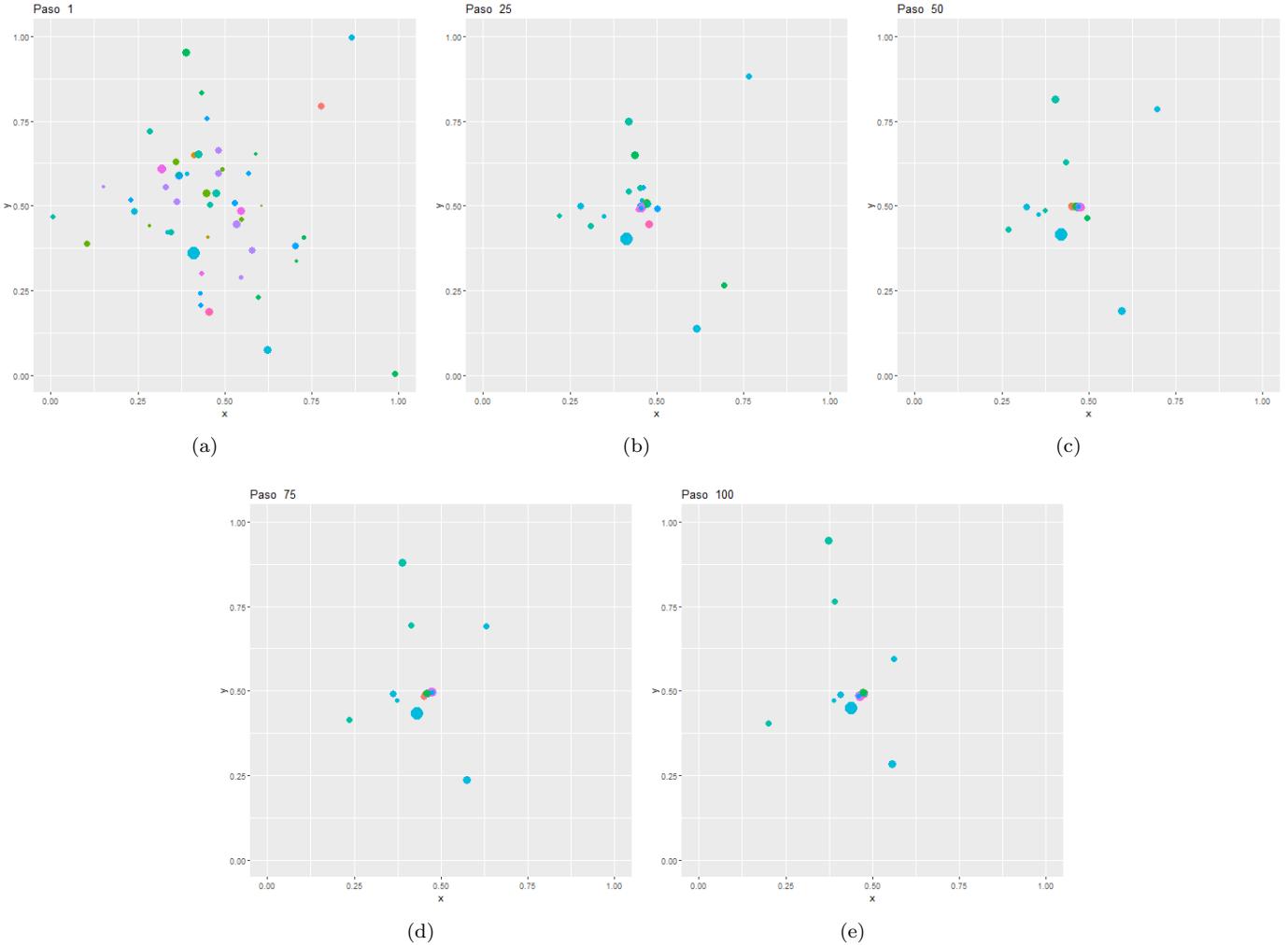


Figura 2: Interacción de partículas

En cuanto a la relación de la velocidad, carga y masa se puede observar en la figura 3 que conforme aumentan los pasos, la distribución de velocidad-carga tiende hacia un mismo punto, en cambio en relación con la masa, no se ve una distribución uniforme [1].

En las siguientes figuras se observa las distribuciones de velocidad en relación con la masa (figura 4) y en relación con la carga (figura 5).

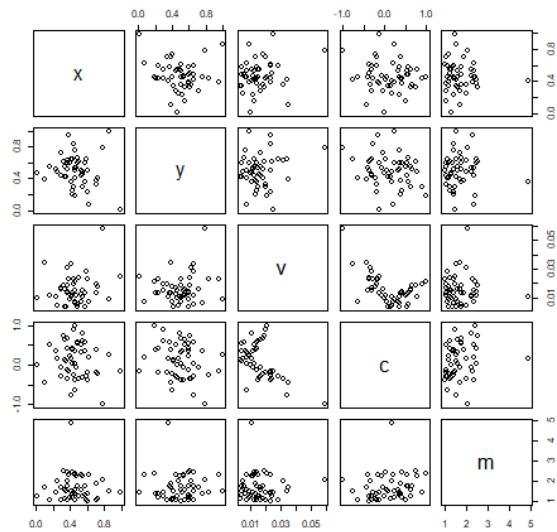


## 4. Conclusión

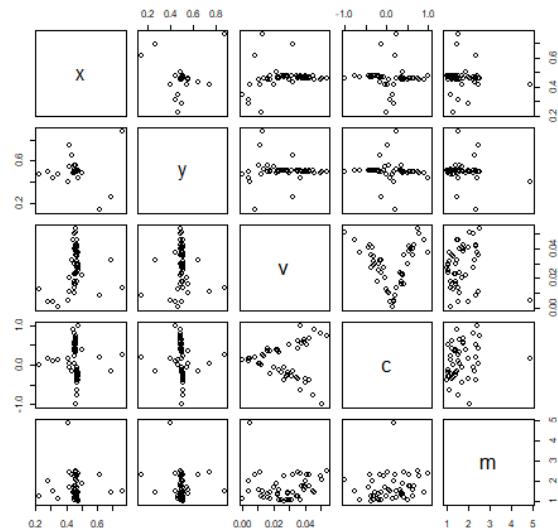
La velocidad en relación con la carga tiene el mismo comportamiento para un valor absoluto de carga, la velocidad es menor cuando las cargas están cerca de cero, es decir, la fuerzas entre partículas es menor. La velocidad en relación con la masa tiene más influencia con masas pequeñas que con las grandes. En el *paso 100* algunas partículas ya no interactúan con las demás partículas y se alejan por las cargas concentradas en un punto, lo que afecta a la relación velocidad-carga y velocidad-masa.

## Referencias

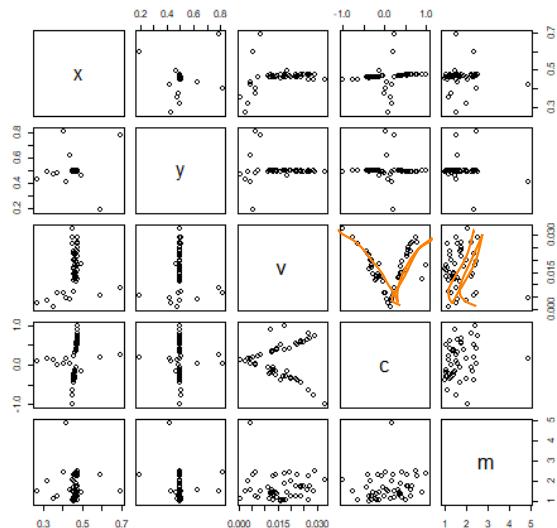
- [1] Robert Kabacoff. Quick R by DataCamp, 2017. URL <https://www.statmethods.net/graphs/scatterplot.html>.
- [2] Elisa Schaeffer. Práctica 9: interacciones entre partículas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.

**Paso 001**

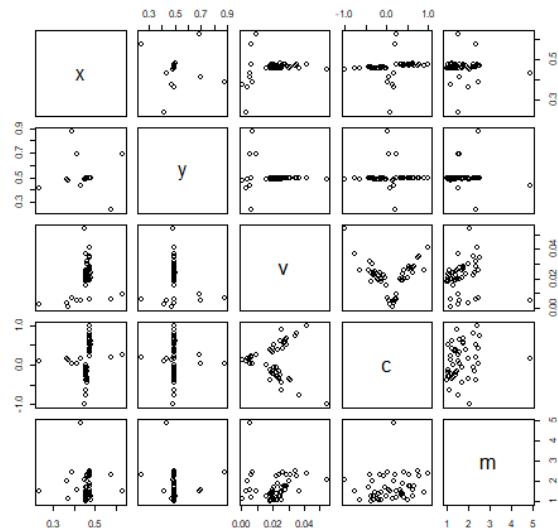
(a)

**Paso 025**

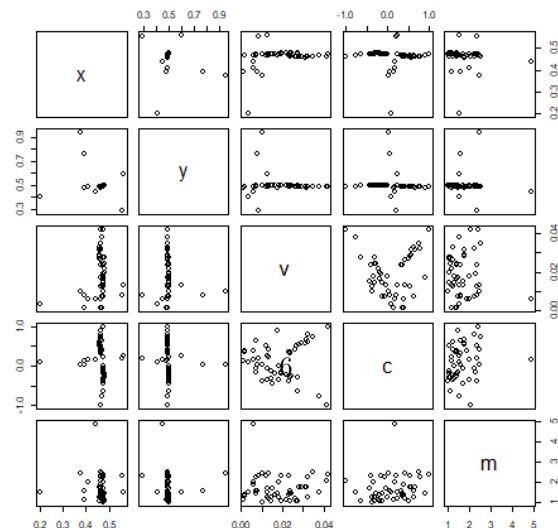
(b)

**Paso 050**

(c)

**Paso 075**

(d)

**Paso 100**

(e)

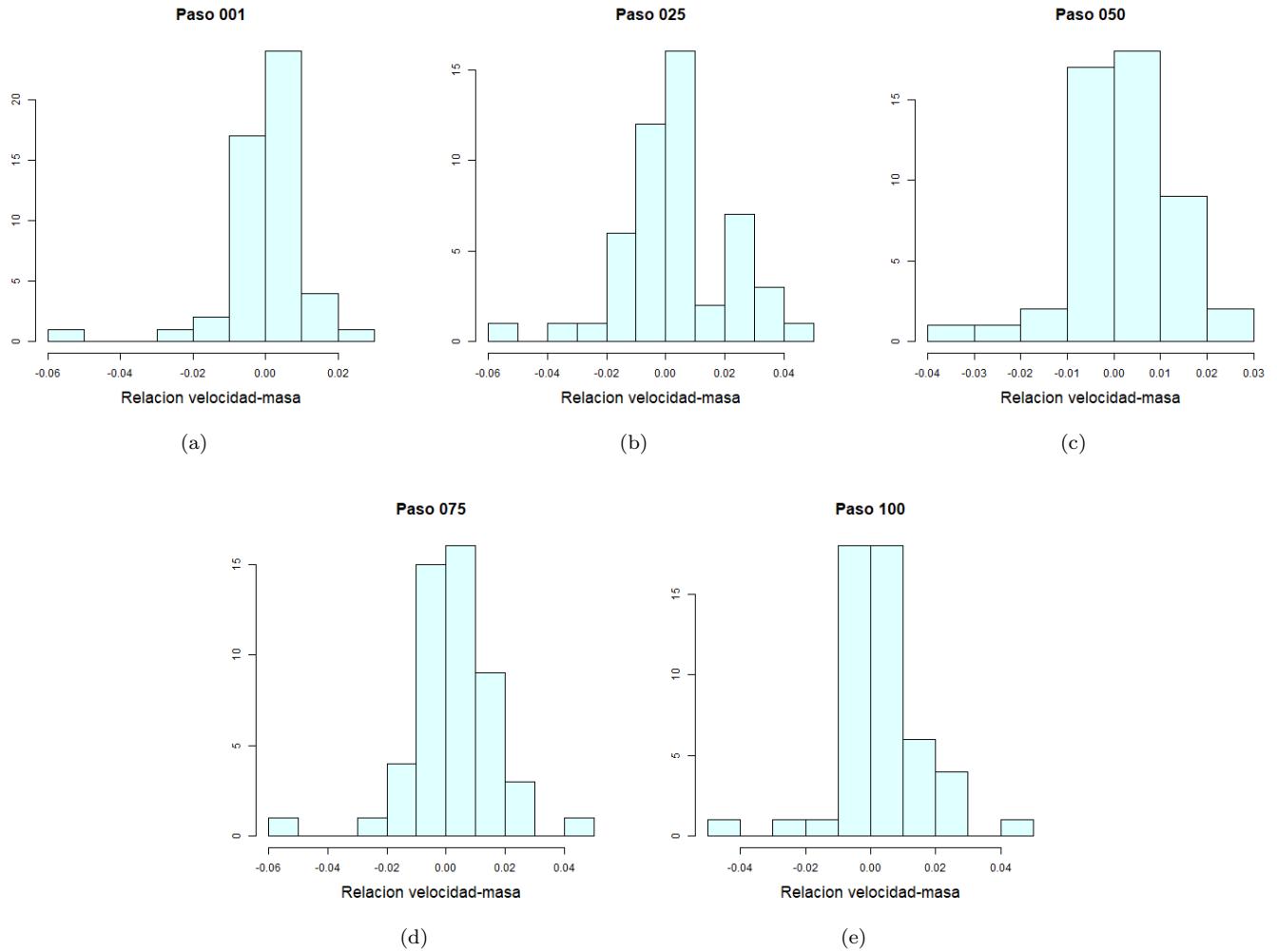
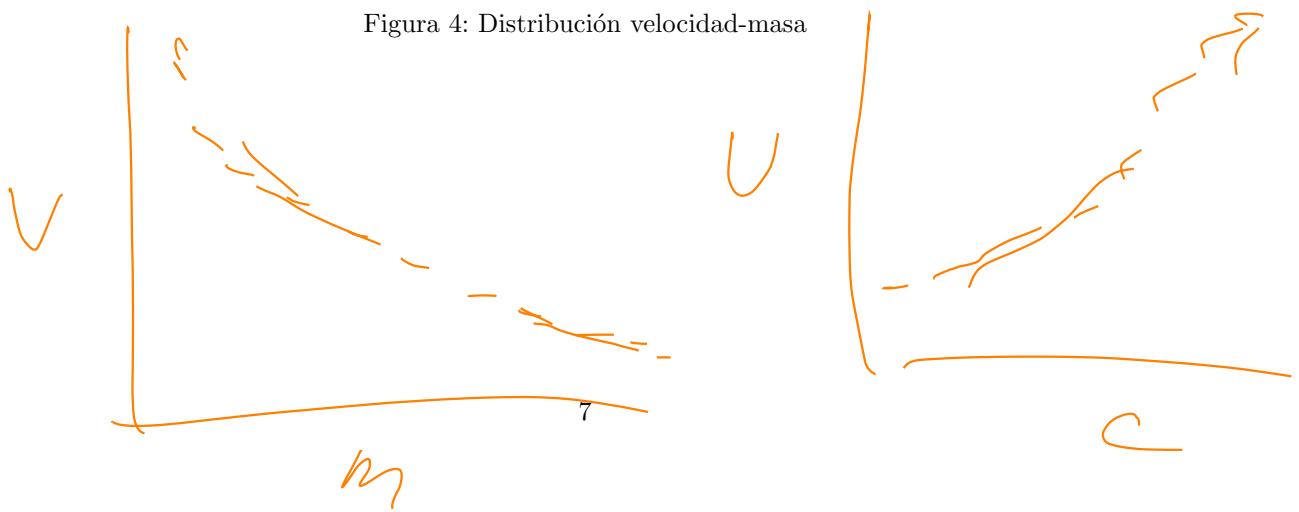


Figura 4: Distribución velocidad-masa



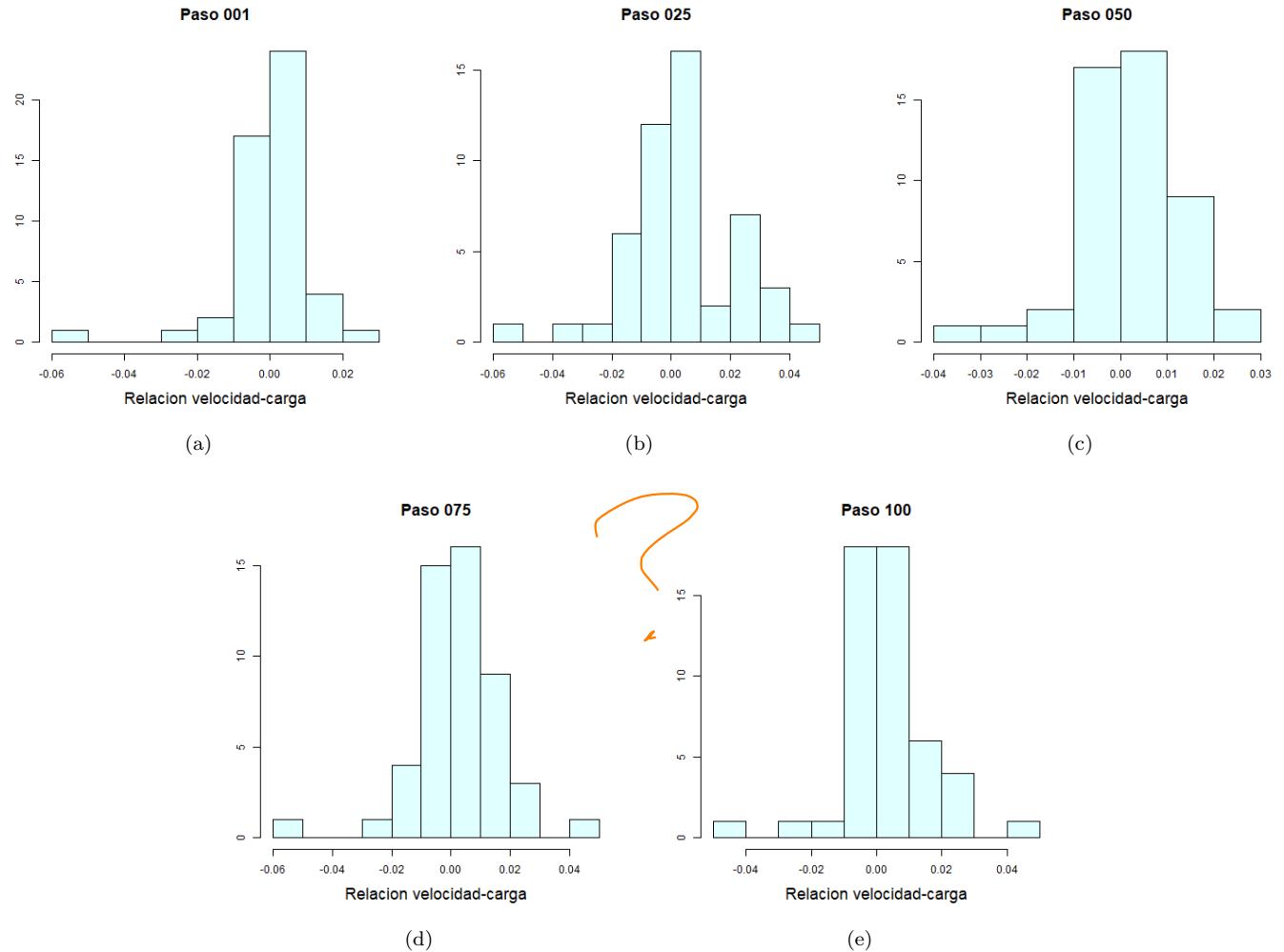


Figura 5: Distribución velocidad-carga

# Práctica 9: interacciones entre partículas

1445183

5 de junio de 2019

## 1. Objetivo

Estudiar la distribución de las velocidades de las partículas, verificando gráficamente la relación entre la *velocidad, carga y masa* de las partículas.

## 2. Descripción

Agregar *masa* a cada partícula generada en el código proporcionado [2], se agrega siguiendo el ejemplo del código dado para agregar *carga* a las partículas y se añade al `data.frame`, en la figura [1] se puede observar las partículas generadas con *masa* y *carga*:

```
1 p <- data.frame(x = rnorm(n), y=rnorm(n), c=rnorm(n), m=rnorm(n))
2 mmax <- max(p$m)
3 mmin <- min(p$m)
4 p$m <- ((p$m - mmin)*(p$m - mmin) / (mmax - mmin))+1 #masa entre 1 y 5
```

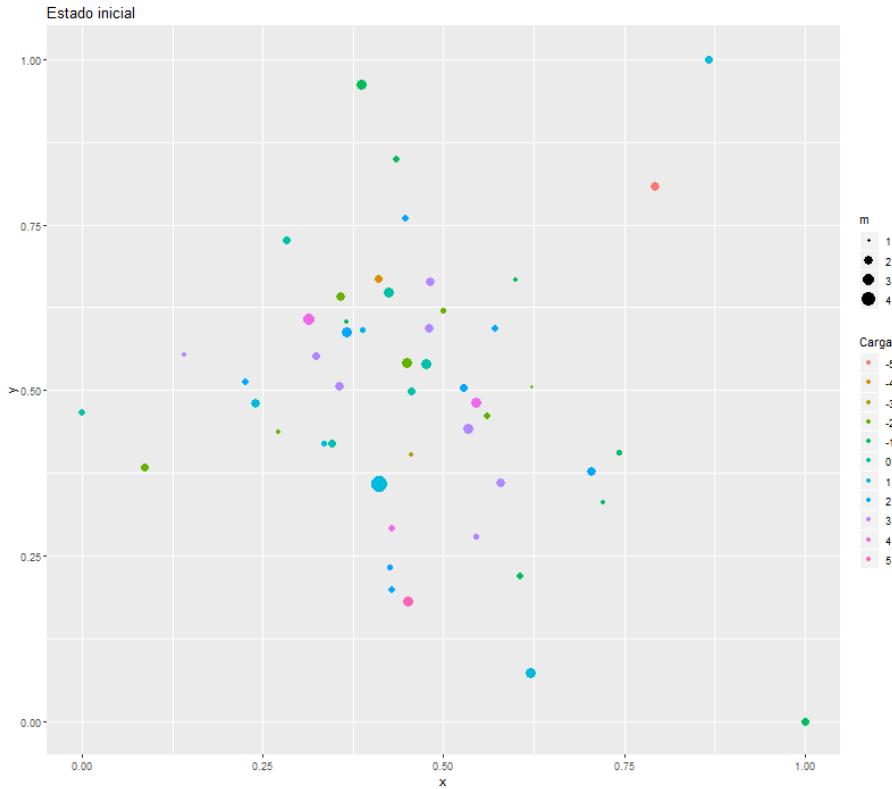


Figura 1: Partículas con carga y masa.

Después la *masa* se añade a la función de fuerza `fuerza` para que interactúe junto con la *carga* al movimiento de las partículas:

```

1 fuerza <- function(i) {
2   xi <- p[i,]$x
3   yi <- p[i,]$y
4   ci <- p[i,]$c
5   mi <- p[i,]$m #masa
6   fx <- 0
7   fy <- 0
8   for (j in 1:n) {
9     cj <- p[j,]$c
10    dir <- (-1)^(1 + 1 * (ci * cj < 0))
11    dx <- xi - p[j,]$x
12    dy <- yi - p[j,]$y
13    factor <- dir * abs(ci - cj) / (sqrt(dx^2 + dy^2) + eps)
14    fx <- fx - dx * factor
15    fy <- fy - dy * factor
16  }
17  return(c(fx, fy)/(mi+1))

```

La *velocidad* se añade al `data.frame` con un vector que se relaciona con la masa, tomando en cuenta los valores obtenidos para *x* y *y* obtenidos en el `for` de las iteraciones.

```

1 p$v <- foreach(i = 1:n, .combine=c) %dopar% (xdifmax[i] + ydifmax[i])
2 p$v <- p$m * p$v

```

### 3. Resultados

Para visualizar los resultados se consideran los pasos 1, 25, 50, 75 y 100. En la figura 2 se observa el movimiento de las partículas bajo la influencia de la *carga* y la *masa*.

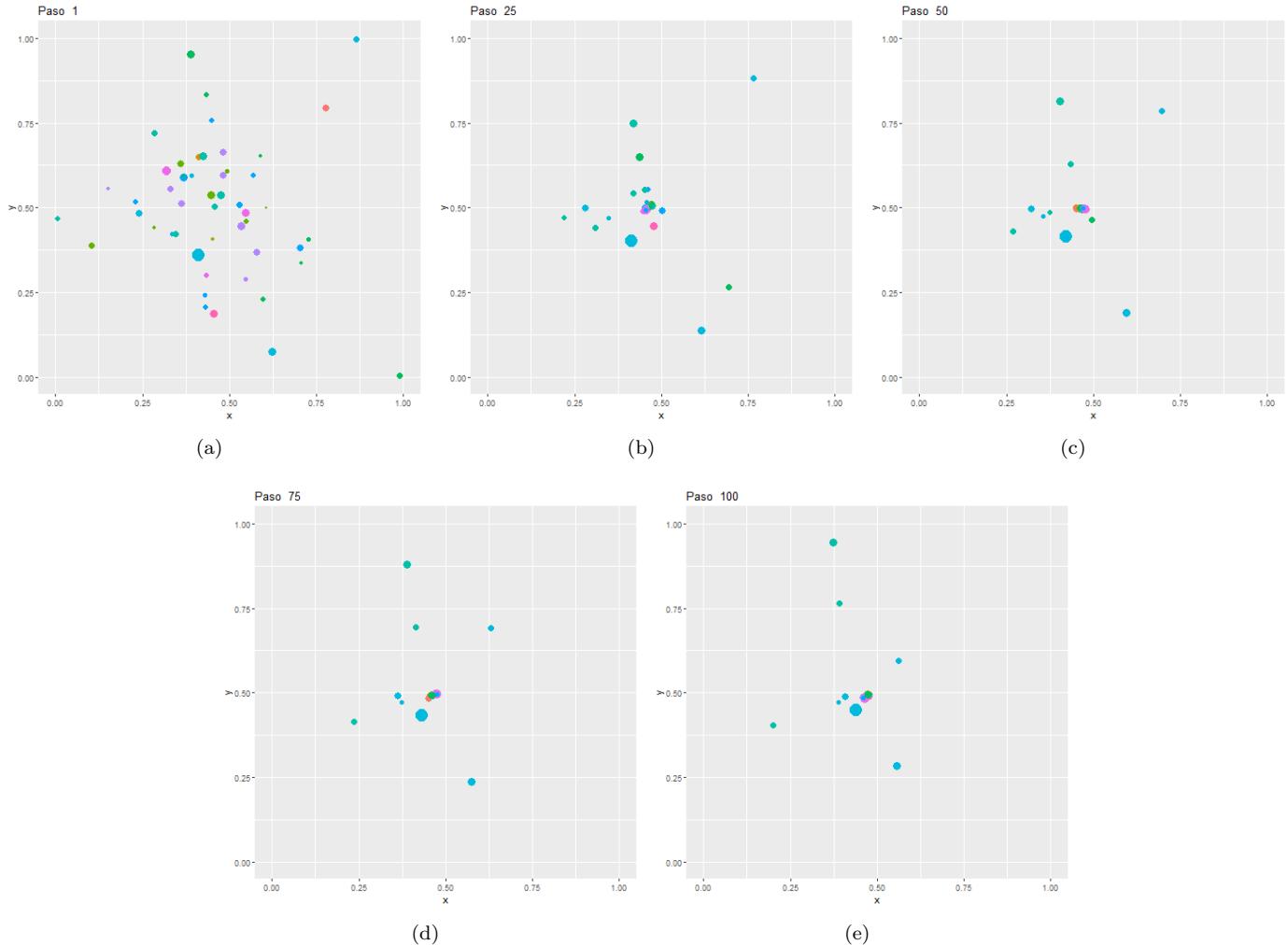


Figura 2: Interacción de partículas.

En cuanto a la relación *velocidad*, *carga* y *masa* se puede observar en la figura 3 que la *velocidad* de movimiento de las partículas es mayor cuando solo se toma la *carga* como fuerza que actúa en las partículas y que la *velocidad* de movimiento de las partículas disminuye al tomar solo en cuenta la *masa*, cuando las dos fuerzas *carga-masa* actúan al mismo tiempo sobre las partículas, la *velocidad* de movimiento de ambas se “combina” por lo que el resultado es una *velocidad media*.

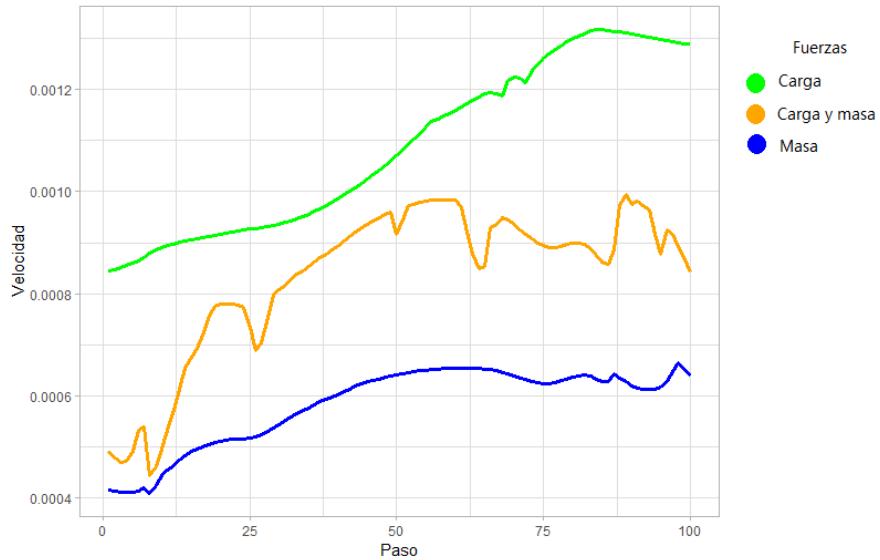


Figura 3: Velocidad de movimiento de las partículas según las fuerzas que actúan en ellas.

## 4. Conclusión

La fuerza más influyente en la *velocidad* de movimiento de las partículas es la *carga*, la *carga* genera mayor atracción entre las partículas que la *masa* y *masa-carga* juntas.

## Referencias

- [1] Edson Cepeda. Práctica 9, 2019. URL <https://sourceforge.net/projects/systemssimulation/files/P9/>.
- [2] Elisa Schaeffer. Práctica 10: Algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.

# Práctica 10: Algoritmo genético

1445183

7 de junio de 2019

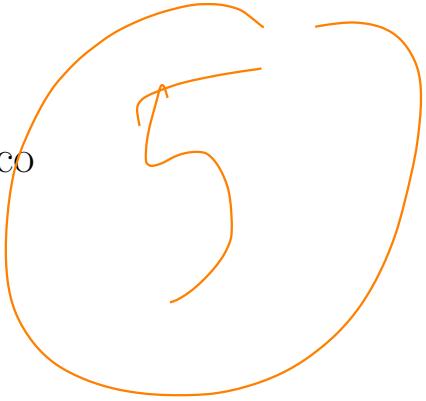
## 1. Retroalimentación

No hubo mejoras.

# Práctica 10: algoritmo genético

1445183

9 de abril de 2019



## 1. Objetivo

Estudiar los efectos del tiempo de ejecución del código paralelizado proporcionado por la práctica [2] variando el número de objetos en las tres instancias siguientes:

- 1.- El peso y el valor de cada objeto se generan independientemente con una distribución normal.
- 2.- El peso de cada objeto se generan independientemente con una distribución normal y su valor es correlacionado con el peso, con un ruido normalmente distribuido de baja magnitud.
- 3.- El peso de cada objeto se generan independientemente con una distribución normal y su valor es inversamente correlacionado con el peso, con un ruido normalmente distribuido de baja magnitud.

## 2. Descripción

El código se paraleliza desde el principio, usando tres núcleos de los cuatro posibles.

```
1 library(testit)
2 suppressMessages(library(doParallel))
3 cls <- makeCluster(detectCores() - 1)
4 registerDoParallel(cls)
```

Se paralelizan las funciones de mutación, reproducción, factibilidad y objetivo basado en el código de *Reyna Fernández*

```
[1] 
1 mutacion2<- function() {
2   if (runif(1) < pm) {
3     return(mutacion(p[i,], n))
4   }
5 }
6
7 reproduccion2<- function() {
8   padres <- sample(1:tam, 2, replace=FALSE)
9   hijos_t <- reproduccion(p[ padres[1],], p[ padres[2],], n)
10  return(hijos_t)
11 }
12
13 objetivo2<- function() {
14   obj_t <- double()
15   obj_t <- c(obj_t, objetivo(p[i,], valores))
16   return(obj_t)
17 }
```

```

18
19 factible2 <- function() {
20   fact_f <- integer()
21   fact_f <- c(fact_f, factible(p[i,], pesos, capacidad))
22   return(fact_f)
23 }
24 }
```

D después se dan las indicaciones que se piden en el *objetivo* en relación *peso* y *valor* y se varía el número de objetos por instancia con valores de 30, 50 y 80, se toma el tiempo de ejecución usando `system.time`

```

1 n <- seq(30,50,80)
2 for(ins in 1:3) {
3
4   if(ins == 1) {
5     valores <- generador.valores(pesos, 10, 500)
6   } else if(ins == 2) {
7     valores <- generador.valores.correlacionados(pesos,10,500)
8   } else if(ins == 3) {
9     valores <- generador.valores.correlacionados(rev(pesos),10,500)
10 }
```

### 3. Resultados

En la figura 1 se puede observar el tiempo que se tarda en dar 50 pasos, los colores corresponden a las instancias indicadas anteriormente.

El tiempo es aumenta cuando se tiene correlación y la cantidad de objetos aumenta, cuando la correlación es inversa se tarda menos tiempo conforme se tiene más cantidad de objetos y cuando los valores son independientes, es decir, no están correlacionados no se tiene una secuencia fija en el tiempo al cambiar la cantidad de objetos.

Línea negra - sin correlación

Línea verde - correlación

Línea morada - correlación inversa

auto - apal

13 - 26

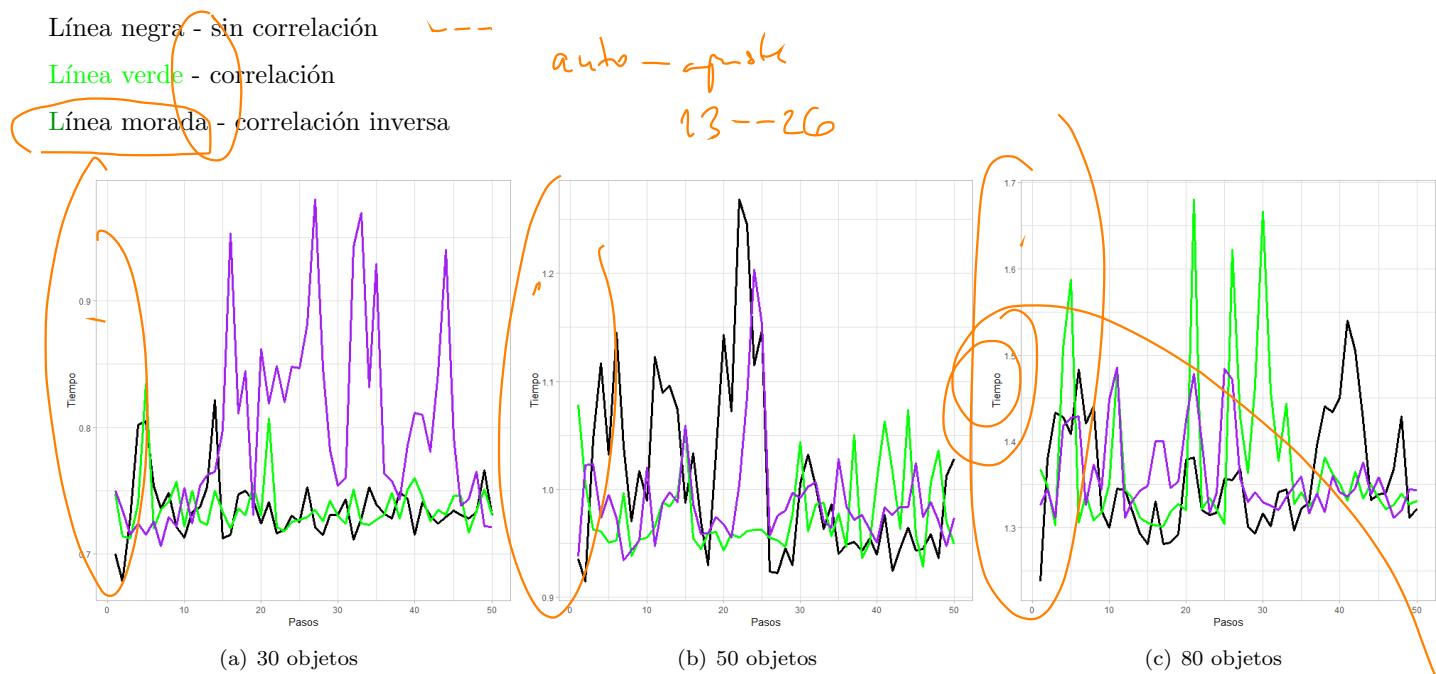
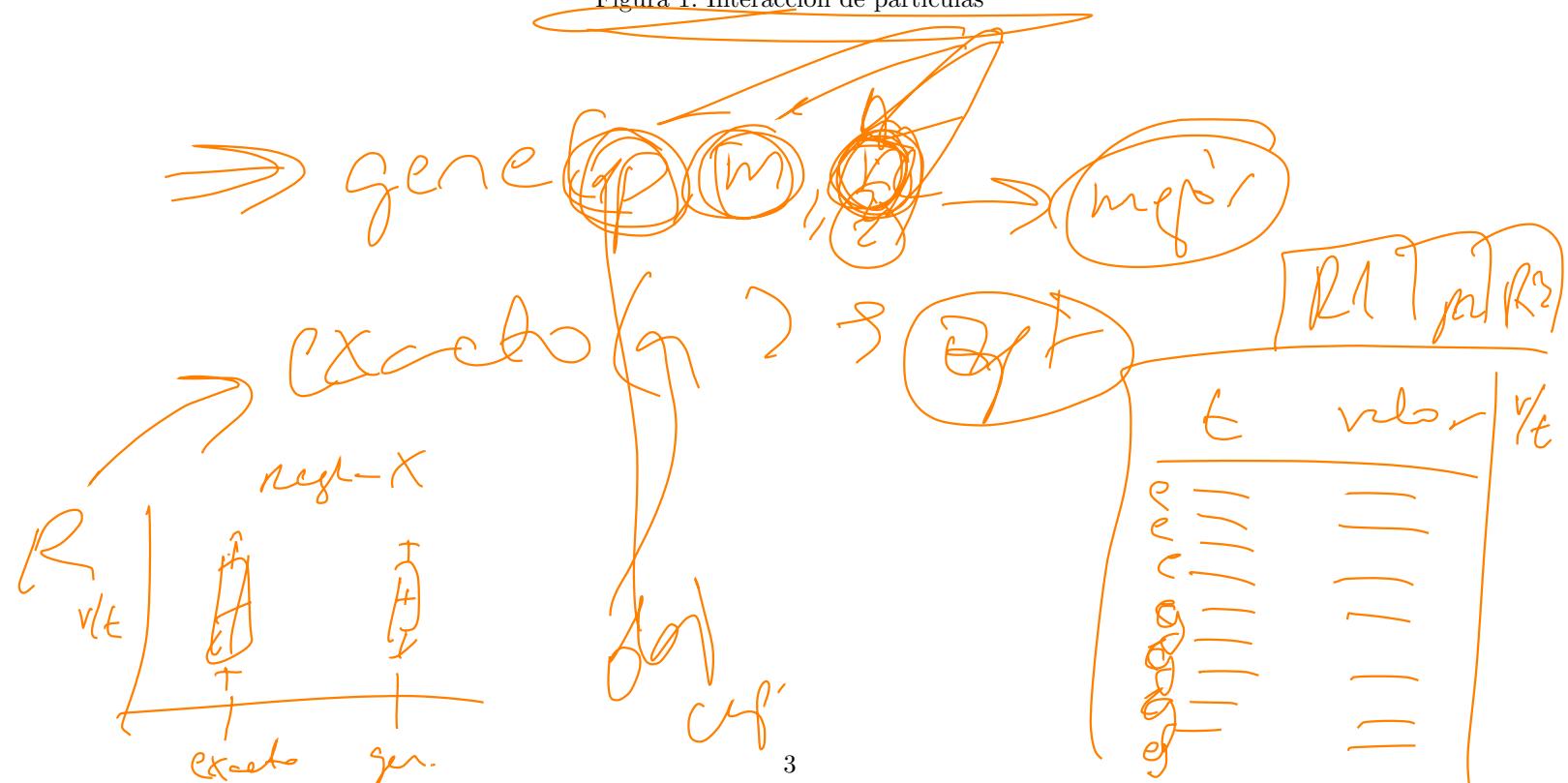


Figura 1: Interacción de partículas



## **4. Conclusión**

Es más fácil (rápido) decidir por objetos que valen mucho y pesan poco como pasa en la correlación inversa, donde el tiempo es menor.

Que los objetos pesen más pero tengan poco valor, hace que tarde en decidir si llevarlos o no, como pasa en la correlación.

## Referencias

- [1] Yessica Reyna. Práctica 10: Algoritmo genético, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/>.
- [2] Elisa Schaeffer. Práctica 10: Algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

# Práctica 11: Frentes de Pareto

1445183

7 de junio de 2019

## 1. Retroalimentación

Se agregó el reto uno.

# Frentes de Pareto

1445183

30 de abril de 2019



## 1. Objetivo

Paralelizar el código y graficar el porcentaje de soluciones de Pareto como función del número de funciones objetivo, verificando que las diferencias observadas sean estadísticamente significativas.

## 2. Descripción

Se empieza paralelizando el código proporcionado por la práctica [3] desde el inicio.

```
1 library(doParallel)
2 cluster <- makeCluster(detectCores() - 1)
3 pick.one <- function(x) {
4   if (length(x) == 1) {
5     return(x)
6   } else {
7     return(sample(x, 1))
8   }
9 }
10 [...]
```

Después el código es modificado para que sean 100 soluciones aleatorias ( $n$ ), con funciones objetivo ( $k$ ) de 2 a 8 con 25 réplicas cada uno haciendo uso de **for**.

```
1 vc <- 4
2 md <- 3
3 tc <- 5
4 n=100
5 datos<-data.frame(funciones= integer(), replicas=integer(),
6                      soluciones= integer(), porcentaje=integer())
7
8 for(k in 2:8){
9   print(k)
10  for(replica in 1:25){
11    print(replica)
12    obj <- list()
13    for(i in 1:k) {
14      obj[[i]] <- poli(vc, md, tc)
15    }
16    minim <- (runif(k) > 0.5)
17    sign <- (1 + -2 * minim)
18    sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
19    clusterExport(cluster, c("n", "k", "sol", "tc", "obj", "eval", "dim", "valor"))
```



Para las pruebas estadísticas se hizo uso de `Shapiro.test` y `Dunn.test` para este último se tuvo que instalar el paquete en R.

### 3. Resultados

Se puede observar en la figura 1 los porcentajes obtenidos para cada función objetivo ( $k$ ), donde a mayor cantidad de  $k$  los porcentajes tienden a ser mayores.

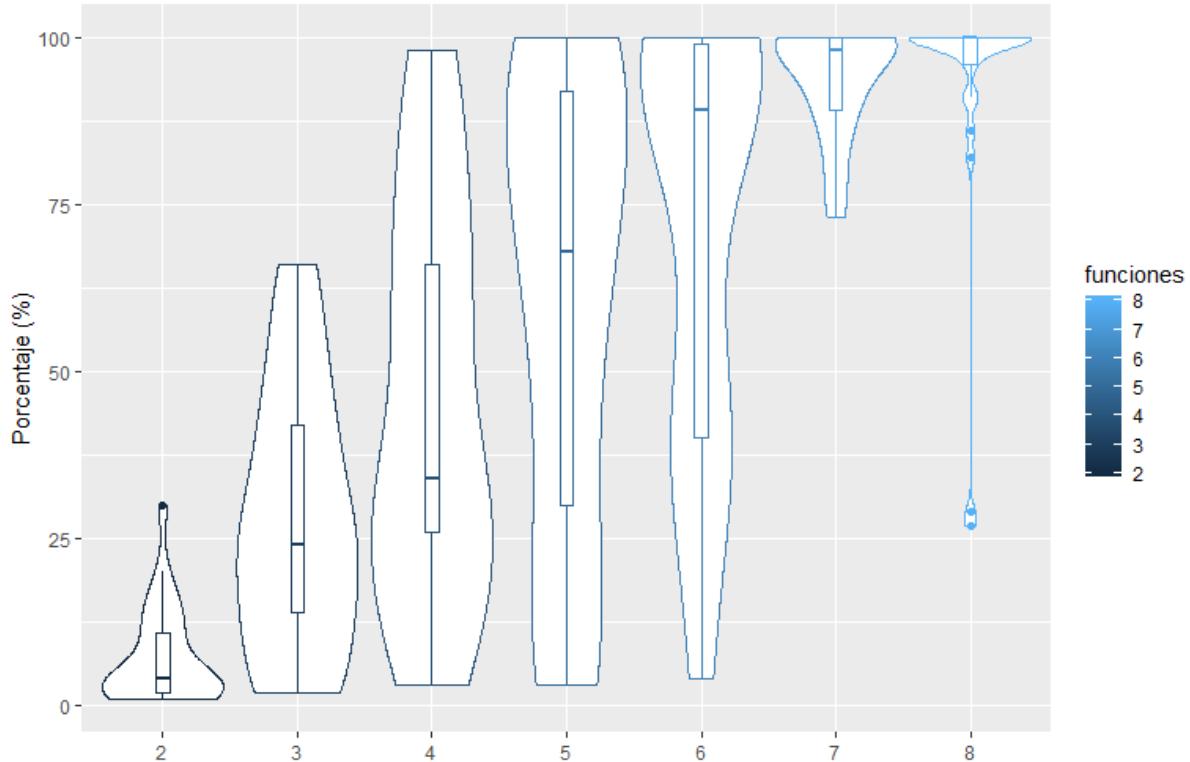


Figura 1: porcentaje de las funciones objetivo ( $k$ )

La prueba de `shapiro.test` nos da el valor  $p$  que en este caso es menor de 0.05 por lo que los datos no cumplen la normalidad, tomando de referencia el trabajo de *Saus* [2] se hace la prueba `dunn.test` [1].

```
1 shapiro.test(datos$porcentaje)
2
3 Shapiro-Wilk normality test
4
5 data: datos$porcentaje
6 W = 0.84612, p-value = 2.668e-12
```

```

1 Kruskal-Wallis rank sum test
2
3 data: x and group
4 Kruskal-Wallis chi-squared = 106.5637, df = 6, p-value = 0
5
6
7             Comparison of x by group
8             (No adjustment)
9 Col Mean -|
10 Row Mean |   2     3     4     5     6     7
11
12   3 | -2.190205
13           0.0143 *
14
15   4 | -3.499568 -1.309362
16           0.0002 *   0.0952
17
18   5 | -4.787924 -2.597718 -1.288356
19           0.0000 *   0.0047 *   0.0988
20
21   6 | -5.957247 -3.767042 -2.457679 -1.169323
22           0.0000 *   0.0001 *   0.0070 *   0.1211
23
24   7 | -7.877179 -5.686973 -4.377611 -3.089254 -1.919931
25           0.0000 *   0.0000 *   0.0000 *   0.0010 *   0.0274
26
27   8 | -8.134850 -5.944644 -4.635282 -3.346925 -2.177602 -0.257671
28           0.0000 *   0.0000 *   0.0000 *   0.0004 *   0.0147 *   0.3983
29
30 alpha = 0.05
31 Reject Ho if p <= alpha/2

```

## 4. Conclusión

A partir de siete objetivos se empieza a observar que los porcentajes de las soluciones no dominadas son equivalentes, es decir, que no existen diferencias significativas. Al existir mayor cantidad de objetivos la comparación entre ellos aumenta por lo que si al compararse un objetivo con otro no mejora hay más posibilidades de que con otro sí, de manera que hay mayor cantidad de soluciones dominantes, esto es la frente de pareto.

## Referencias

- [1] Alexis Dinno. Package ‘dunn.test’, 2017. URL <https://cran.r-project.org/web/packages/dunn.test/dunn.test.pdf>.
- [2] Liliana Saus. Práctica 11: Frentes de Pareto, 2018. URL <https://github.com/pejli/simulacion/blob/master/p11/p11.pdf>.
- [3] Elisa Schaeffer. Práctica 11: Frentes de Pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.

# Práctica 11: Frentes de Pareto

1445183

7 de junio de 2019

## 1. Objetivo

Paralelizar el código y graficar el porcentaje de soluciones de Pareto como función del número de funciones objetivo, verificando que las diferencias observadas sean estadísticamente significativas.

## 2. Descripción

Se empieza paralelizando el código proporcionado por la práctica [3] desde el inicio.

```
1 library(doParallel)
2 cluster <- makeCluster(detectCores() - 1)
3 pick.one <- function(x) {
4   if (length(x) == 1) {
5     return(x)
6   } else {
7     return(sample(x, 1))
8   }
9 }
10 [...]
```

Después el código es modificado para que sean 100 soluciones aleatorias ( $n$ ), con funciones objetivo ( $k$ ) de 2 a 8 con 25 réplicas cada uno haciendo uso de **for**.

```
1 vc <- 4
2 md <- 3
3 tc <- 5
4 n=100
5 datos<-data.frame(funciones= integer(), replicas=integer(),
6                      soluciones= integer(), porcentaje=integer())
7
8 for(k in 2:8){
9   print(k)
10  for(replica in 1:25){
11    print(replica)
12    obj <- list()
13    for(i in 1:k) {
14      obj[[i]] <- poli(vc, md, tc)
15    }
16    minim <- (runif(k) > 0.5)
17    sign <- (1 + -2 * minim)
18    sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
19    clusterExport(cluster, c("n", "k", "sol", "tc", "obj", "eval", "dim", "valor"))
```

```

20
21 val <- parSapply(cluster , 1:n, valor)
22   val <- t(val)
23   mejor1 <- which.max(sign[1] * val[,1])
24   mejor2 <- which.max(sign[2] * val[,2])
25   no.dom <- logical()
26   dominadores <- integer()
27
28   for (i in 1:n) {
29     d <- logical()
30     for (j in 1:n) {
31       d <- c(d, domin/by(sign * val[i,], sign * val[j,], k))
32     }
33     cuantos <- sum(d)
34     dominadores <- c(dominadores, cuantos)
35     no.dom <- c(no.dom, cuantos == 0) # nadie le domina
36   }
37   frente <- subset(val, no.dom) # solamente las no dominadas
38   porcentaje<-((dim(frente)[1])/n)*100
39   datos<-rbind(datos, data.frame(funciones=k, replicas=replica, soluciones=n, porcentaje))
40 }
41 }
```

Para las pruebas estadísticas se hizo uso de `Shapiro.test` y `Dunn.test` para este último se tuvo que instalar el paquete en R.

### 3. Resultados

Se puede observar en la figura 1 los porcentajes obtenidos para cada función objetivo ( $k$ ), donde a mayor cantidad de  $k$  los porcentajes tienden a ser mayores.

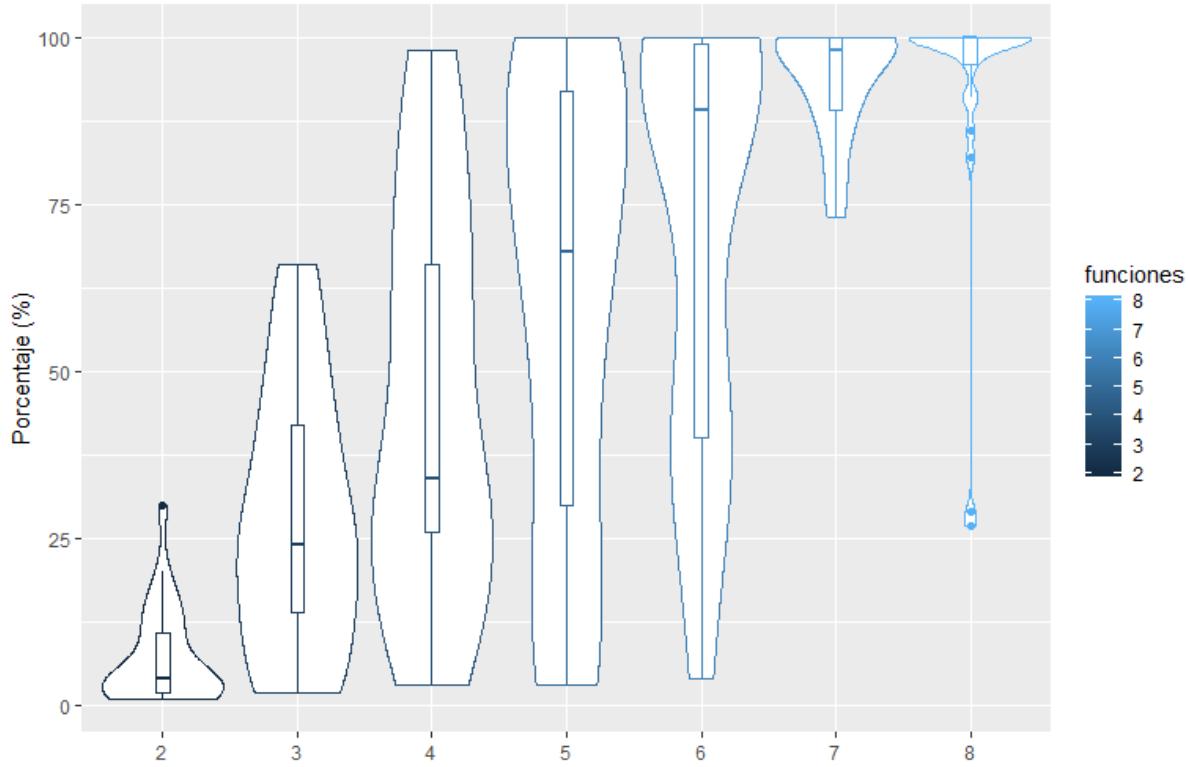


Figura 1: Porcentaje de las funciones objetivo ( $k$ ).

La prueba de `shapiro.test` nos da el valor  $p$  que en este caso es menor de 0.05 por lo que los datos no cumplen la normalidad, tomando de referencia el trabajo de *Saus* [2] se hace la prueba `dunn.test` [1].

```
1 shapiro.test(datos$porcentaje)
2   Shapiro-Wilk normality test
3 data:  datos$porcentaje
4 W = 0.84612, p-value = 2.668e-12
```

```

1 Kruskal-Wallis rank sum test
2
3 data: x and group
4 Kruskal-Wallis chi-squared = 106.5637, df = 6, p-value = 0
5
6
7 Comparison of x by group
8 (No adjustment)
9 Col Mean -|
10 Row Mean | 2 3 4 5 6 7
11
12 3 | -2.190205
13 | 0.0143 *
14
15 4 | -3.499568 -1.309362
16 | 0.0002 * 0.0952
17
18 5 | -4.787924 -2.597718 -1.288356
19 | 0.0000 * 0.0047 * 0.0988
20
21 6 | -5.957247 -3.767042 -2.457679 -1.169323
22 | 0.0000 * 0.0001 * 0.0070 * 0.1211
23
24 7 | -7.877179 -5.686973 -4.377611 -3.089254 -1.919931
25 | 0.0000 * 0.0000 * 0.0000 * 0.0010 * 0.0274
26
27 8 | -8.134850 -5.944644 -4.635282 -3.346925 -2.177602 -0.257671
28 | 0.0000 * 0.0000 * 0.0000 * 0.0004 * 0.0147 * 0.3983
29
30 alpha = 0.05
31 Reject Ho if p <= alpha/2

```

## 4. Conclusión

A partir de siete objetivos se empieza a observar que los porcentajes de las soluciones no dominadas son equivalentes, es decir, que no existen diferencias significativas. Al existir mayor cantidad de objetivos la comparación de un objetivo con otro no mejora hay más posibilidades de que con otro si, de manera que hay mayor cantidad de soluciones dominantes, esto es la frente de Pareto.

## 5. Reto 1

El primer reto consiste en seleccionar un subconjunto del frente de Pareto de tal forma que la selección esté diversificada, es decir, que no estén agrupados juntos en una sola zona del frente las soluciones seleccionadas. Graficar los resultados de la selección, indicando con un color cuáles se incluyen en el subconjunto diverso.

El código se modificó de manera que se pudiera seleccionar los subconjuntos diversificados a través del frente de Pareto, obteniendo un máximo y un mínimo del frente que se seleccionará.

```

1 cluster <- makeCluster(detectCores()- 1)
2 clusterExport(cluster , c("domin.by", "sign", "eval", "poli", "n", "pick.one"))
3
4 vc <- 4
5 md <- 3
6 tc <- 5
7 datafun <- seq(2, 14, by=1)
8 for (k in datafun) {
9   for (replicas in 1:5) {
10     obj <- list()
11     clusterExport(cluster , c("vc", "md", "tc"))
12     obj <- parSapply(cluster , 1:k, prop)
13
14     minim <- (runif(k) > 0.5)
15     sign <- (1 + -2 * minim)
16     n <- 200 # cuantas soluciones aleatorias
17     sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
18     val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
19     clusterExport(cluster , c("tc", "obj", "sol", "eval", "k"))
20     val <- parSapply(cluster , 1:n, verify)
21     val <- t(val)
22
23     mejor1 <- which.max(sign[1] * val[,1])
24     mejor2 <- which.max(sign[2] * val[,2])
25     cual <- c("max", "min")
26     xl <- paste("Primer objetivo (", cual[minim[1] + 1], ")")
27     yl <- paste("Segundo objetivo (", cual[minim[2] + 1], ")")
28
29     for (i in 1:n) {
30       d <- logical()
31       for (j in 1:n) {
32         d <- c(d, domin.by(sign * val[i,], sign * val[j,], k))
33       }
34       cuantos <- sum(d)
35       dominadores <- c(dominadores, cuantos)
36       no.dom <- c(no.dom, cuantos == 0) # nadie le domina
37     }
38     frente <- subset(val, no.dom) # solamente las no dominadas
39     dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
40     Select <- frente[c(round(runif(round(dim(frente)[1]/2), min = 1, max = dim(frente)[1]))),]
41     Select1 <- data.frame()
42     Select1 <- rbind(Select1, Select)
43
44     sub <- seq(1, k, by = 1)
45
46     for (i in vec) {
47       if((i+1) == is.element(i+1, vec)*(i+1)){
48         png(paste("p11_frente", k, "-", replicas, "_", i, "_", i+1, ".png", sep=""))
49         xt = paste("Objetivo ", i, " (", cual[minim[i] + 1], ")")
50         yt = paste("Objetivo ", i+1, " (", cual[minim[i+1] + 1], ")")
51         plot(val[,i], val[,i+1], xlab=xt, ylab=yt)
52         points(frente[,i], frente[,i+1], col="purple", pch=16, cex=1.5)
53         points(Select1[,i], Select1[,i+1], col="orange", pch=16, cex=1.5)
54         graphics.off()
55       }
56     }
57     data <- data.frame(pos=rep(0, n), dom=dominadores)

```

## 6. Resultados

En la figura 2 se puede observar el frente de Pareto (puntos morados) que es el frente original, y los subconjuntos del frente de Pareto (puntos naranjas) diversificados.

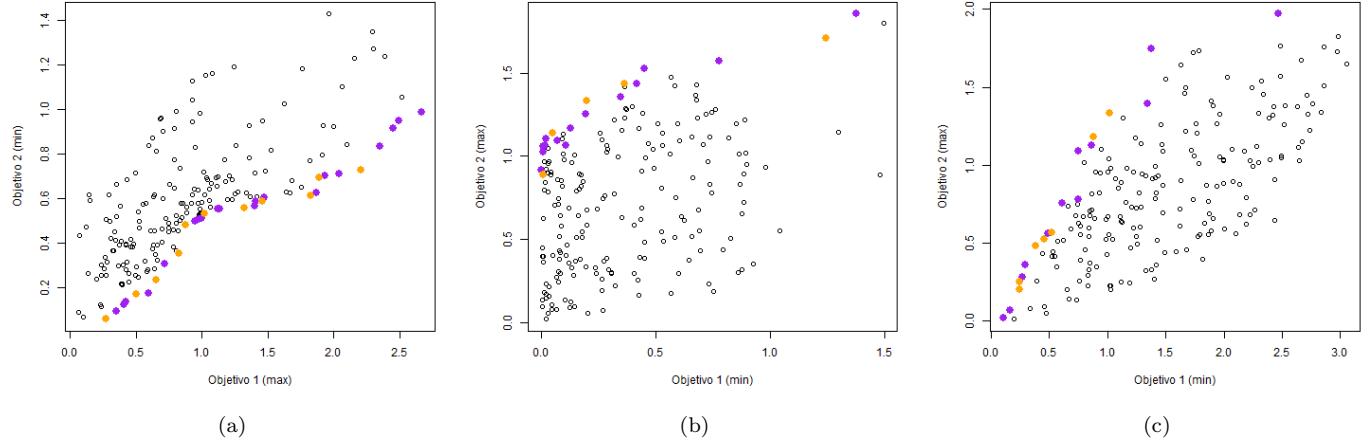


Figura 2: Diversificación de subconjuntos a través del frente de Pareto

## Referencias

- [1] Alexis Dinno. Package ‘dunn.test’, 2017. URL <https://cran.r-project.org/web/packages/dunn.test/dunn.test.pdf>.
- [2] Liliana Saus. Práctica 11, 2018. URL <https://github.com/pejli/simulacion/tree/master/p11>.
- [3] Elisa Schaeffer. Práctica 11: Frentes de Pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.

# Práctica 12: Red neuronal

1445183

7 de junio de 2019

## 1. Retroalimentación

Se cambiaron los valores para los colores, además se mejoró el reto1.

# Práctica 12: red neuronal

1445183

7 de mayo de 2019



## 1. Objetivo

Estudiar de manera sistemática el desempeño de la red neuronal para diez dígitos en función de las tres probabilidades asignadas a la generación de dígitos (`ngb`) en el código proporcionado por esta práctica [1] variando a las tres en un experimento factorial adecuado.

## 2. Descripción

El código se paraleliza desde el principio, usando tres núcleos de los cuatro posibles.

```
1 cluster <- makeCluster(detectCores() - 1)
2
3
4 binario <- function(d, 1) {
5   b <- rep(FALSE, 1)
6   while (1 > 0 | d > 0) {
7     b[1] <- (d %% 2 == 1)
8     l <- l - 1
9     d <- bitwShiftR(d, 1)
10  }
11  return(b)
12}
13 decimal <- function(bits, 1) {
14  valor <- 0
15  for (pos in 1:1) {
16    valor <- valor + 2^(1 - pos) * bits[pos]
17  }
18  return(valor)
19}
20.
21.
22.
23.
24.
25 clusterExport(cluster, c( "neuronas", "binario", "decimal", "modelos", "tope", "k", "dim", "n"))
26.
27.
28 #PRUEBA
29 contadores <- parSapply(cluster, 1:prueba, function(x){
30   d <- sample(0:tope, 1)
31   pixeles <- runif(dim) <- modelos[d + 1,] # fila 1 contiene el cero, etc.
32   correcto <- binario(d, n)
33   salida <- rep(FALSE, n)
34   for (i in 1:n) {
35     w <- neuronas[i,]
```

```

36     deseada <- correcto[i]
37     resultado <- sum(w * pixeles) >= 0
38     salida[i] <- resultado
39   }
40   r <- min(decimal(salida, n), k)
41   return(r==d)}
42 datos<-rbind(datos, data.frame(Replica=replicas, Negro=pn, Gris=pg, Blanco=pb, Porcentaje=(sum
43   (contadores)/prueba)*100))
44 }
45 }
46 }
47 }

```

Se genera un archivo *CSV* con los datos proporcionados por la práctica para indicar la ubicación de los pixeles y se vincula al código en la rutina de generación de pixeles (`ngb`), después se varían las probabilidades para `ngb` como se muestra en el siguiente código haciendo uso de `for` con 20 repeticiones cada una y obteniendo los respectivos porcentajes para cada combinación.

```

1 #repeticiones
2 replica<-20
3
4 tmax<-5000
5 entrenamiento <- ceiling(0.7 * tmax)
6 prueba <- tmax - entrenamiento
7 datos<- data.frame( Replica= integer(), Negras=integer(), Grises=integer(), Blancas=integer(),
8   Porcentaje=integer())
9
10 #archivo csv
11 modelos <- read.csv("digitos.modelo.csv", sep=" ", header=FALSE, stringsAsFactors=F)
12
13 #variar probabilidades
14 for (PN in c(0.995,0.92,0.002)) {
15   for(PG in c(0.92,0.002,0.995)){
16     for(PB in c(0.002,0.995,0.92)){
17       modelos[modelos=="n"] <- pn # pixeles negros en plantillas
18       modelos[modelos=="g"] <- pg # pixeles grises en plantillas
19       modelos[modelos=="b"] <- pb # pixeles blancos en plantillas
20       for(replicas in 1:replica){
21         print(replicas)
22
23         tasa <- 0.15
24         contadores <- vector()
25         n <- floor(log(k-1, 2)) + 1
26         neuronas <- matrix(runif(n * dim), nrow=n, ncol=dim) # perceptrones
27
28       #ENTRENAMIENTO
29       for (t in 1:entrenamiento) { # entrenamiento
30         d <- sample(0:tope, 1)
31         pixeles <- runif(dim) <- modelos[d + 1,]
32         correcto <- binario(d, n)
33         for (i in 1:n) {
34           w <- neuronas[i,]
35           deseada <- correcto[i]
36           resultado <- sum(w * pixeles) >= 0
37           if (deseada != resultado) {
38             ajuste <- tasa * (deseada - resultado)
39             tasa <- tranqui * tasa
40             neuronas[i,] <- w + ajuste * pixeles
41
42

```

43            }  
44            }  
45            }

### 3. Resultados *NextHS...}*

*Sí en Negro*

En la figura 1a se puede observar que existe mayor porcentaje cuando la probabilidad de *Negro* es mayor (crecendo al 1).

En la figura 1b el mayor porcentaje es cuando la probabilidad es mayor en *Gris*.

En la figura 1c el mayor porcentaje se encuentra a cualquier probabilidad de *Negro* (preferentemente cercano al 0) y mayor probabilidad de *Gris*.

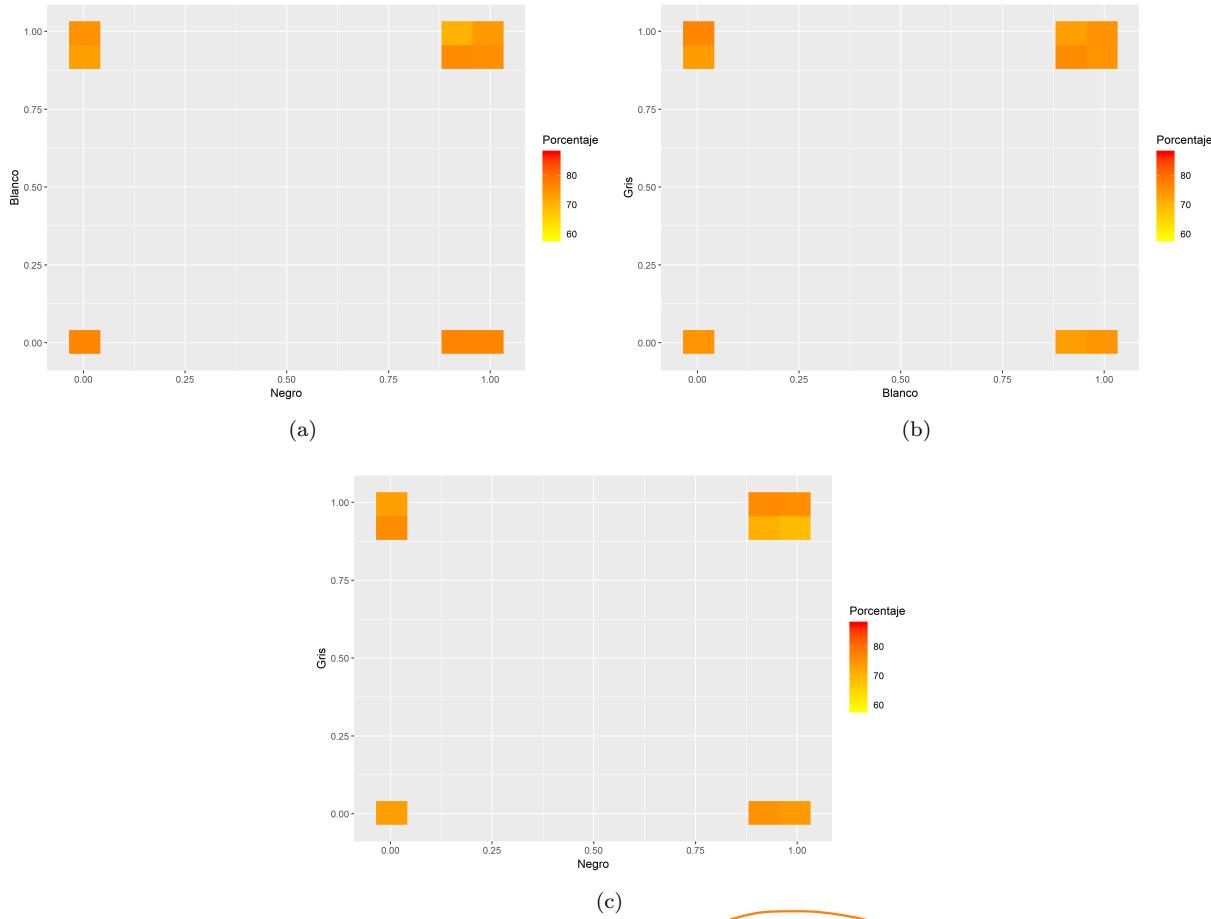
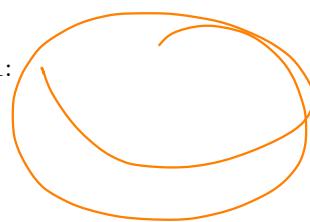


Figura 1:



## 4. Conclusión

Los pixeles que dominan la formación de caracteres son los pixeles *Negro* y *Gris* dado a que los porcentajes de que se reconociera correctamente los dígitos fueron mayores para estos pixeles sin importar las variaciones y combinaciones de probabilidades.

## Reto 1

El primer reto consiste en extender y entrenar la red neuronal para que reconozca además por lo menos doce símbolos ASCII adicionales, aumentando la resolución de las imágenes a  $5 \times 7$  de lo original de  $3 \times 5$ , estos se obtuvieron tomando de referencia a *Saus* [?].

```
1 modelos <- read.csv("digitos.reto.csv", sep=" ", header=FALSE, stringsAsFactors=F)
2 modelos[modelos=='n'] <- 0.995 # pixeles negros en plantillas
3 modelos[modelos=='g'] <- 0.92 # pixeles grises en plantillas
4 modelos[modelos=='b'] <- 0.002 # pixeles blancos en plantillas
5
6 r <- 7
7 c <- 5
8 dim <- r * c
9
10 n <- 49
11 w <- ceiling(sqrt(n))
12 h <- ceiling(n / w)
13
14 letras <- c(0:9, "L", "I", "C", "H", "T", "E", "F", "U", "P", "A", "W", "M")
15
16 png("plantilla.png", width=1600, height=2000)
17 par(mfrow=c(w, h), mar = c(0,0,7,0))
18 suppressMessages(library("sna"))
19
20 for (j in 1:n) {
21   d <- sample(0:21, 1)
22   pixeles <- runif(dim) < modelos[d + 1,] # fila 1 contiene el cero, etc.
23   imagen <- matrix(pixeles, nrow=r, ncol=c, byrow=TRUE)
24   plot.sociomatrix(imagen, drawlab=FALSE, diaglab=FALSE,
25                     main=paste(letras[d+1], ""), cex.main=5)
26 }
27 graphics.off()
```

## 5. Resultados

En la figura 2 se observan los números y letras codificadas donde solo 18 caracteres fueron reconocidos de manera satisfactoria.

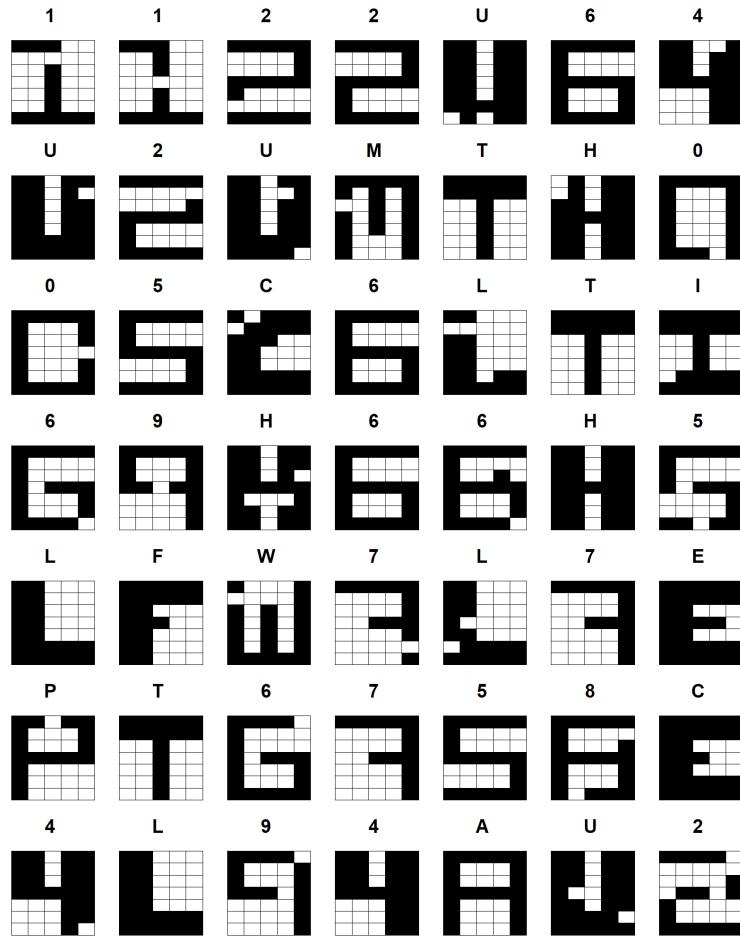


Figura 2: Caracteres de red neuronal extendida

## 6. Conclusión

En este caso no se pudo entrenar debidamente la red neuronal bajo estas nuevas condiciones, observando la figura 2 se puede deducir que para una red neuronal extendida se necesita mayor tiempo de entrenamiento para que posteriormente pueda hacer un reconocimiento correcto.

## Referencias

- [1] Elisa Schaeffer. Práctica 11: Frentes de Pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p12.html>.

# Práctica 12: red neuronal

1445183

5 de junio de 2019

## 1. Objetivo

Estudiar de manera sistemática el desempeño de la red neuronal para diez dígitos en función de las tres probabilidades asignadas a la generación de dígitos (`ngb`) en el código proporcionado por esta práctica [2] variando a las tres en un experimento factorial adecuado.

## 2. Descripción

El código se paraleliza desde el principio, usando tres núcleos de los cuatro posibles.

```
1 cluster <- makeCluster(detectCores() - 1)
2
3
4 binario <- function(d, 1) {
5   b <- rep(FALSE, 1)
6   while (1 > 0 | d > 0) {
7     b[1] <- (d %% 2 == 1)
8     l <- l - 1
9     d <- bitwShiftR(d, 1)
10  }
11  return(b)
12}
13 decimal <- function(bits, 1) {
14  valor <- 0
15  for (pos in 1:1) {
16    valor <- valor + 2^(1 - pos) * bits[pos]
17  }
18  return(valor)
19}
20.
21.
22.
23.
24.
25 clusterExport(cluster, c("neuronas", "binario", "decimal", "modelos", "tope", "k", "dim", "n"))
26
27
28 #PRUEBA
29 contadores <- parSapply(cluster, 1:prueba, function(x){
30   d <- sample(0:tope, 1)
31   pixeles <- runif(dim) <- modelos[d + 1,] # fila 1 contiene el cero, etc.
32   correcto <- binario(d, n)
33   salida <- rep(FALSE, n)
34   for (i in 1:n) {
35     w <- neuronas[i,]
```

```

36     deseada <- correcto[i]
37     resultado <- sum(w * pixeles) >= 0
38     salida[i] <- resultado
39   }
40   r <- min(decimal(salida, n), k)
41   return(r==d)}
42 datos<-rbind(datos, data.frame(Replica=replicas, Negro=pn, Gris=pg, Blanco=pb, Porcentaje=(sum
43   (contadores)/prueba)*100))
44 }
45 }
46 }
47 }

```

Se genera un archivo *CSV* con los datos proporcionados por la práctica para indicar la ubicación de los pixeles y se vincula al código en la rutina de generación de pixeles (`ngb`), después se varían las probabilidades para `ngb` como se muestra en el siguiente código haciendo uso de `for` con 20 repeticiones cada una y obteniendo los respectivos porcentajes para cada combinación.

```

1 #repeticiones
2 replica<-20
3
4 tmax<-5000
5 entrenamiento <- ceiling(0.7 * tmax)
6 prueba <- tmax - entrenamiento
7 datos<- data.frame( Replica= integer(), Negras=integer(), Grises=integer(), Blancas=integer(),
8   Porcentaje=integer())
9
10 #archivo csv
11 modelos <- read.csv("digitos.modelo.csv", sep=" ", header=FALSE, stringsAsFactors=F)
12
13 #variar probabilidades
14 for (PN in c(0.995,0.92,0.99)) {
15   for(PG in c(0.92,0.080,0.995)){
16     for(PB in c(0.002,0.05,0.1)){
17
18       modelos[modelos=='n'] <- pn # pixeles negros en plantillas
19       modelos[modelos=='g'] <- pg # pixeles grises en plantillas
20       modelos[modelos=='b'] <- pb # pixeles blancos en plantillas
21       for(replicas in 1:replica){
22         print(replicas)
23
24         tasa <- 0.15
25         contadores <-vector()
26         n <- floor(log(k-1, 2)) + 1
27         neuronas <- matrix(runif(n * dim), nrow=n, ncol=dim) # perceptrones
28
29
30       #ENTRENAMIENTO
31       for (t in 1:entrenamiento) { # entrenamiento
32         d <- sample(0:tope, 1)
33         pixeles <- runif(dim) <- modelos[d + 1,]
34         correcto <- binario(d, n)
35         for (i in 1:n) {
36           w <- neuronas[i ,]
37           deseada <- correcto[i]
38           resultado <- sum(w * pixeles) >= 0
39           if (deseada != resultado) {
40             ajuste <- tasa * (deseada - resultado)
41             tasa <- tranqui * tasa
42             neuronas[i ,] <- w + ajuste * pixeles

```

43 }  
44 }  
45 }

### 3. Resultados

En la figura I se puede ver que la relación blanco con gris es la que más se distingue a través de las diferentes proporciones, en cambio en las demás se puede ver el contraste muy marcado.

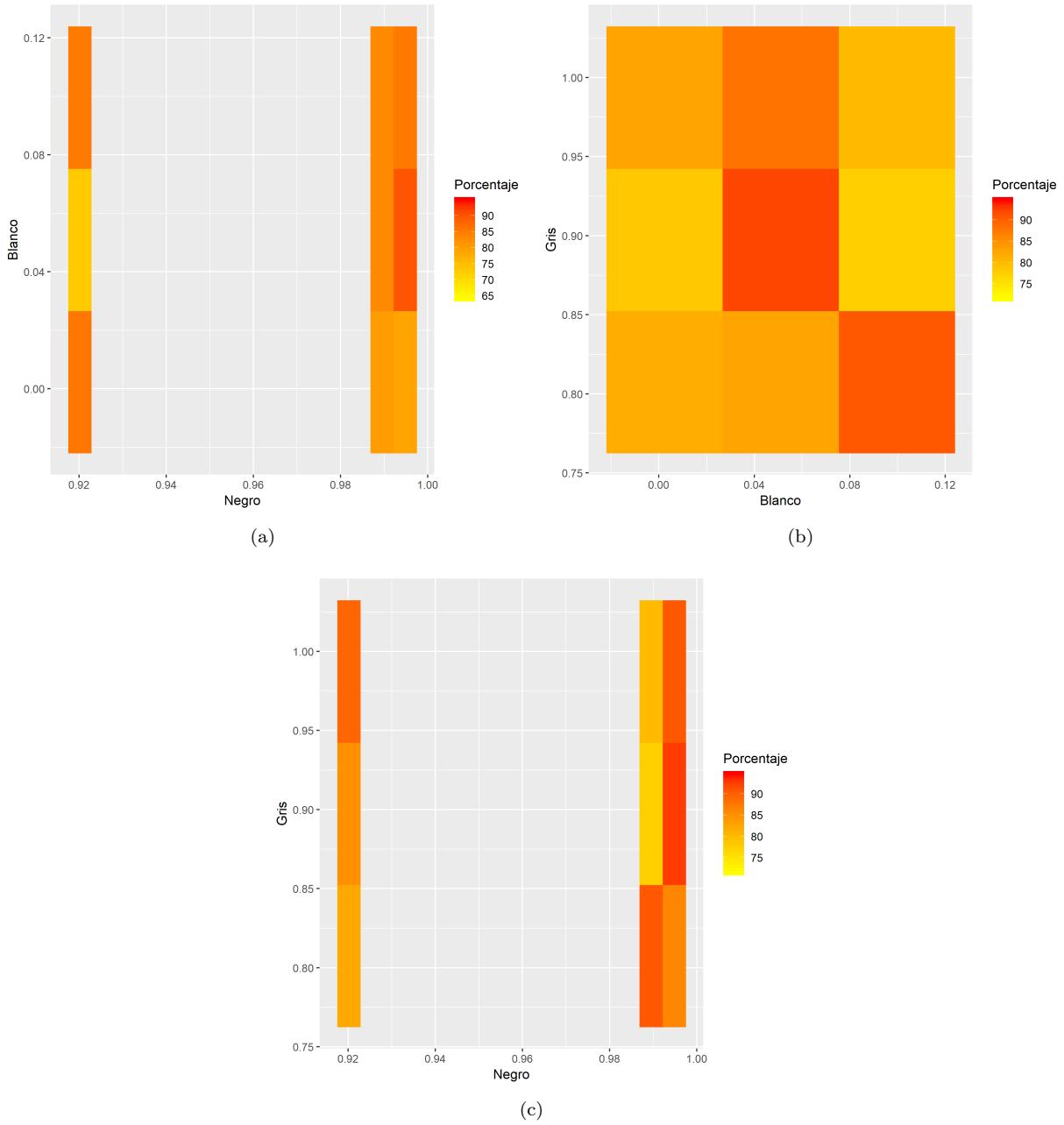


Figura 1:

## 4. Conclusión

Los pixeles que dominan la formación de caracteres son los pixeles *Gris* y *Blanco* dado a que los porcentajes de que se reconociera correctamente los dígitos fueron mayores para estos pixeles sin importar las variaciones y combinaciones de probabilidades.

## Reto 1

El primer reto consiste en extender y entrenar la red neuronal para que reconozca además por lo menos doce símbolos ASCII adicionales, aumentando la resolución de las imágenes a  $5 \times 7$  de lo original de  $3 \times 5$ , el código se modificó para agregar las letras que el código debe de leer [1].

```
1 binario <- function(d, 1) {
2   b <- rep(FALSE, 1)
3   while (1 > 0 | d > 0) {
4     b[1] <- (d %% 2 == 1)
5     l <- l - 1
6     d <- bitwShiftR(d, 1)
7   }
8   return(b)
9 }
10
11 decimal <- function(bits, 1) {
12   valor <- 0
13   for (pos in 1:1) {
14     valor <- valor + 2^(1 - pos) * bits[pos]
15   }
16   return(valor)
17 }
18
19 modelos <- read.csv("digitos.reto.csv", sep=" ", header=FALSE, stringsAsFactors=F)
20 modelos[modelos=='n'] <- 0.995
21 modelos[modelos=='g'] <- 0.92
22 modelos[modelos=='b'] <- 0.002
23
24
25
26 r <- 7
27 c <- 5
28 dim <- r * c
29
30
31
32 tasa <- 0.15
33 tranqui <- 0.9
34 tope<-21
35 letras <- c(0:9, "L", "I", "C", "H", "T", "E", "F", "U", "P", "A", "W", "M")
36
37
38 k <- length(letras)
39 contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
40 rownames(contadores) <- 0:tope
41 colnames(contadores) <- c(0:tope, NA)
42
43 n <- floor(log(k-1, 2)) + 1
44 neuronas <- matrix(runif(n * dim), nrow=n, ncol=dim) # perceptrones
45
46 for (t in 1:5000) { # entrenamiento
47   d <- sample(0:tope, 1)
48   pixeles <- runif(dim) < modelos[d + 1,]
49   correcto <- binario(d, n)
50   salida <- rep(FALSE, n)
51   for (i in 1:n) {
52     w <- neuronas[i,]
53     deseada <- correcto[i]
54     resultado <- sum(w * pixeles) >= 0
55     if (deseada != resultado) {
56       ajuste <- tasa * (deseada - resultado)
57       tasa <- tranqui * tasa
58       neuronas[i,] <- w + ajuste * pixeles
```

```

59 }
60 }
61 }
62 }
63 for (t in 1:300) { # prueba
64   d <- sample(0:tope, 1)
65   pixeles <- runif(dim) < modelos[d + 1,] # fila 1 contiene el cero, etc.
66   correcto <- binario(d, n)
67   salida <- rep(FALSE, n)
68   for (i in 1:n) {
69     w <- neuronas[i,]
70     deseada <- correcto[i]
71     resultado <- sum(w * pixeles) >= 0
72     salida[i] <- resultado
73   }
74   r <- min(decimal(salida, n), k) # todos los no-existentes van al final
75   contadores[d+1, r+1] <- contadores[d+1, r+1] + 1
76 }
77 print(contadores)

```

## 5. Resultados

La figura 5 es la plantilla de los número y letras generados a partir del documento *CSV* 1 y se puede observar el resultado de los contadores donde no fueron bien identificados los caracteres generados.

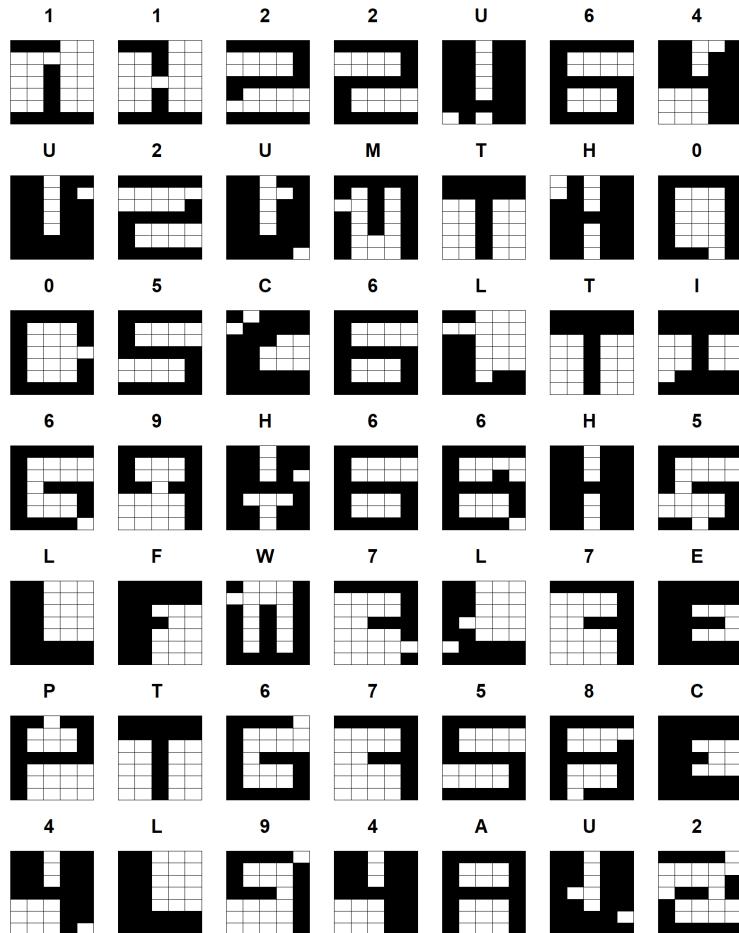


Figura 2: Caracteres de red neuronal extendida

	0	1	2	3	4	5	6	7	8	9	L	I	C	H	T	E	F	U	P	A	W	M	<NA>
0	8	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	11	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0	0	14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	2	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	2	2	2
5	0	0	0	0	0	9	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	2	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	2	2
7	5	0	0	0	0	0	9	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	6	1	0	1	0	3	0	8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	7	0	0	0	0	0	0	3	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
10	8	7	1	0	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
11	9	0	0	0	0	0	5	0	2	0	0	0	0	7	0	0	0	0	0	0	0	0	0
12	L	2	0	4	0	0	0	0	0	1	0	2	0	0	0	1	0	0	0	0	0	0	0
13	I	0	0	0	0	0	0	0	0	0	10	0	0	1	0	1	0	0	0	0	0	0	0
14	C	0	0	0	0	0	0	0	0	11	0	0	0	2	0	1	0	0	0	0	0	0	0
15	H	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	6	0	1	0	3
16	T	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	10	3	0	0	0	0	0
17	E	0	0	0	0	0	0	0	0	0	0	0	0	0	7	1	0	3	0	0	0	0	0
18	F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21
19	U	0	1	0	0	0	4	0	0	0	0	0	0	0	3	0	0	0	0	0	0	1	2
20	P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0
21	A	0	0	0	2	0	0	0	0	1	0	0	1	0	0	0	9	0	0	0	0	0	1
22	W	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	1
23	M	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	3	0	12	

## 6. Conclusión

En este caso no se pudo entrenar debidamente la red neuronal bajo estas nuevas condiciones, observando la figura 5 se puede deducir que para una red neuronal extendida se necesita mayor tiempo de entrenamiento para que posteriormente pueda hacer un reconocimiento correcto.

## Referencias

- [1] Liliana Saus. Práctica 12, 2018. URL <https://github.com/pejli/simulacion/tree/master/p12>.
- [2] Elisa Schaeffer. Práctica 11: Frentes de Pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p12.html>.

# Proyecto final

1445183

7 de junio de 2019

## 1. Retroalimentación

Se mejoraron detalles en el formato, además de los colores en la gráfica de cinética.

5783

# Estudio de la cinética de un adsorbente catiónico

Dulce Esperanza Carrasco Castillo

4 de junio de 2019

**4 + 4 + 4 + 5 + 1 + 4 = 22 pts**

## Resumen

Este trabajo describe la cinética de un adsorbente catiónico, se varía la fuerza de atracción del adsorbente (carga catiónica) de 0.02, 0.05 y 0.1 para los pasos 2, 5, 10, 15, 20, 30, 40, 50. Usando el software Rproject se grafican los resultados de número de partículas cargadas adsorbidas vs velocidad de adsorción (pasos), obteniendo que la fuerza de atracción 0.05 es la que obtuvo la mayor adsorción de partículas en menor cantidad de pasos, se realizó un ANOVA como prueba estadística para el paso diez donde los valores no son significativos, es decir, cualquier fuerza de adsorción aplicada (0.02, 0.05, 0.1) es eficiente para la adsorción.

Palabras clave: Adsorción, Cinética, Simulación, Interacción, Cargas.

## 1. Introducción

Un adsorbente es un sólido que tiene la capacidad de retener en su superficie componentes presentes en un medio líquido o gaseoso, son comúnmente usados para la eliminación de contaminantes en un medio. El adsorbente mayormente utilizado es el carbón activado el cual se puede sintetizar de cualquier compuesto orgánico llevado a pirólisis, también los adsorbentes poliméricos son muy comunes.

Existen tres tipos de adsorción, la química cuando el adsorbente forma enlaces con el adsorbato, la física que es regida por fuerzas de Van Der Waals y la adsorción por intercambio que es cuando los iones de cierta sustancia se atraen a la superficie del adsorbente por fuerzas electrostáticas.

Para determinar la eficiencia de un adsorbente es

necesario analizar los aspectos cinéticos y termodinámicos<sup>[2]</sup>, el análisis cinético de un adsorbente nos indica la capacidad que tiene el adsorbente, es decir, la eficiencia en cuanto a rendimiento de adsorción.

## 2. Antecedentes

Se sabe que el uso de adsorbentes para eliminación de contaminantes en medios gaseosos y líquidos son muy estudiados y existe gran cantidad de artículos enfocados a estos, más en el área ambiental<sup>[1]</sup>. En el caso de carbón activado como adsorbente las investigaciones van enfocadas más a la fuente del carbón activado para su capacidad de adsorción, en cambio los adsorbentes de intercambio iónico también son de gran estudio ya que evitan las reacciones en

el medio, por lo cual solo adsorben las partículas de cierta carga y se mantienen en la superficie del adsorbente.

### 3. Herramientas

La simulación se realizó en una laptop ASUS modelo X507M con un procesador Intel Celeron N4000. Haciendo uso del software *R Project versión 3.5.3 (2019-03-11)*. Como base se usó la práctica 9 de R paralelo: simulación y análisis de datos de *W.H. Shaeffer*.

### 4. Experimentación

Se crearon cien partículas cargadas aleatoriamente de -5 a 5, el algoritmo se modificó de tal forma de darle carga positiva al adsorbente de esta manera que las partículas con carga negativa fueran atraídas hacia el adsorbente. Se varió la fuerza de atracción de las partículas con valores de 0.02, 0.05 y 0.1, con 50 iteraciones y 3 réplicas para cada una, para medir la cinética se tomaron en cuenta los pasos con valores de 2, 5, 10, 15, 20, 30, 40 y 50, de manera de obtener la cantidad de partículas adsorbidas en cada paso, como prueba estadística se uso del ANOVA tomando los resultados del paso 10 para cada valor de fuerza.

### 5. Resultados y discusión

En la figura 1 se pueden ver las partículas adsorbidas para cada valor de fuerza de atracción en el paso 10 de la cinética donde en la fuerza de atracción 0.02 es donde menos adsorbe.

Se puede observar en la figura 2 que en donde hubo más partículas adsorbidas fue para la fuerza de atracción de 0.05, siguiendo de 0.02 y por último 0.1. También se puede observar que para la fuerza de atracción de 0.02 la adsorción de partículas incrementó considerablemente a partir del paso 15,

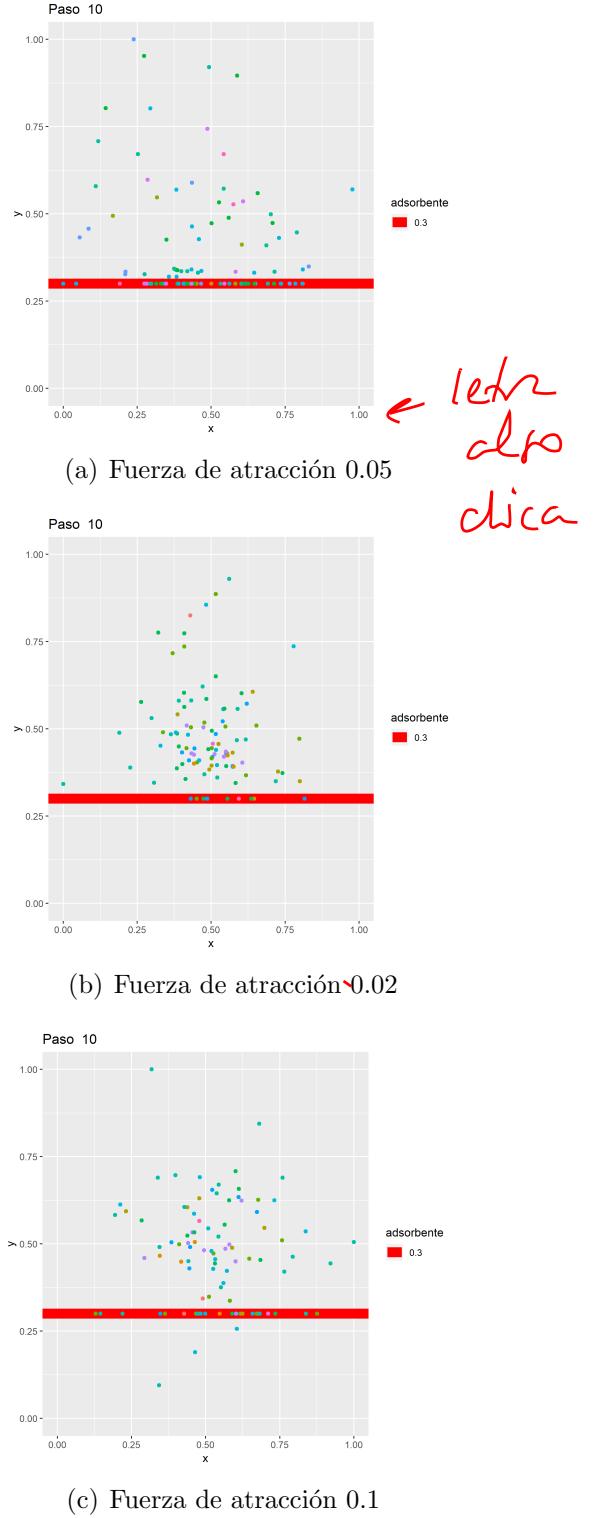


Figura 1: Partículas adsorbidas en el paso 10.

en cambio para la fuerza de atracción de 0.05 fue una adsorción gradual.

Para la fuerza 0.1 se puede decir que la cantidad de

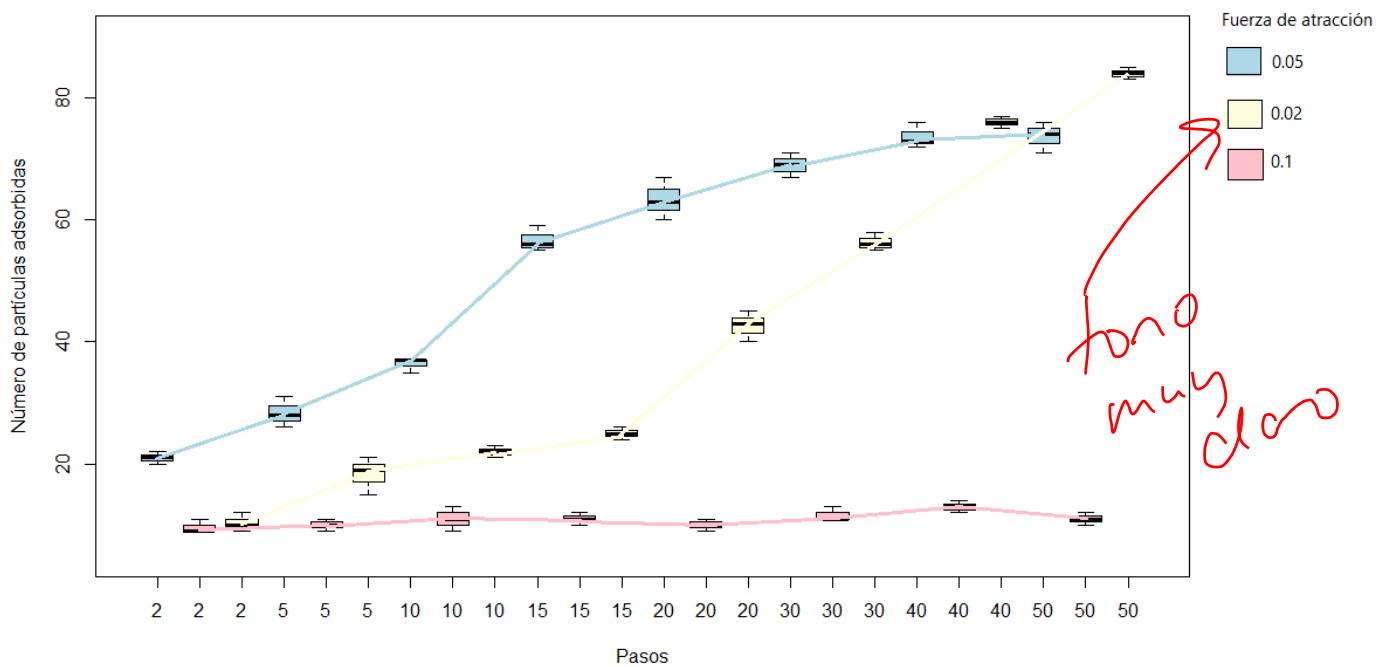


Figura 2: Cinética del adsorbente catiónico.

partículas adsorbidas fueron casi las mismas en todos los pasos, de manera que lo que adsorbe al inicio es lo más que podrá adsorber a través del tiempo, esto es debido a la alta fuerza de atracción, es decir, a la alta carga catiónica del adsorbente, al concentrarse gran cantidad de cargas negativas cerca del adsorbente, esta misma fuerza en conjunto hace que se repelen entre ellas y no se adsorban, se podría pensar que la adsorción sería más eficiente con mayor cantidad de cargas positivas en el adsorbente, pero aquí se puede apreciar que no.

Como resultados del ANOVA se obtuvo un valor  $F$  de 5.143208497 y un valor  $P$  de 0.004723224, tomando un valor de significancia de 0.05, se puede decir que las fuerzas afectan significativamente al comportamiento del adsorbente.

## 6. Conclusiones

La cantidad de carga catiónica (fuerza de atracción) en un adsorbente catiónico determina si el adsorbente tendrá eficiencia, esto se puede saber a través de un análisis cinético del adsorbente, donde se sabe la capacidad de adsorción del mismo. En este caso se obtuvo que a una carga catiónica de 0.05 la adsorción de partículas fue gradual a través del tiempo, pero a cargas cationicas grandes el efecto de adsorción no es efectivo.

~~Conclusiones~~

Se podría hacer más análisis cinéticos y variar las fuerzas con otros valores, incluir algún modelo matemático como base de la experimentación, incluso se podría tomar en cuenta la masa en el efecto de adsorción y la porosidad del adsorbente para ver de que manera afecta al rendimiento.

## Agradecimientos

Se agradece al ~~Dr.~~ Martín Rosas por ayudar en la idea principal de este proyecto, también a Marcos Guajardo ~~compañero de clase~~ por asesorarme en el algoritmo de la simulación.

## Referencias

- [1] H. Qiu and L. Lv. Critical review in adsorption kinetic models. *journal of zhejiangcatinic and anionic dye adsorption by agricultural solid wastes: A comprehensive review. Journal of Zhejiang University-SCIENCE*, 10:716–724, 2009.
- [2] M. Salleh and D. Mahmoud. Cationic and anionic dye adsorption by agricultural solid wastes: A comprehensive review. *Desalination*, 280:1–13, 2011.

# Estudio de la cinética de un adsorbente catiónico

Dulce Esperanza Carrasco Castillo

7 de junio de 2019

---

## Resumen

Este trabajo describe la cinética de un adsorbente catiónico, se varía la fuerza de atracción del adsorbente (carga catiónica) de 0.02, 0.05 y 0.1 para los pasos 2, 5, 10, 15, 20, 30, 40, 50. Usando el software Rproject se grafican los resultados de número de partículas cargadas adsorbidas vs velocidad de adsorción (pasos), obteniendo que la fuerza de atracción 0.05 es la que obtuvo la mayor adsorción de partículas en menor cantidad de pasos, se realizó un ANOVA como prueba estadística para el paso diez donde los valores no son significativos, es decir, cualquier fuerza de adsorción aplicada (0.02, 0.05, 0.1) es eficiente para la adsorción.

Palabras clave: Adsorción, Cinética, Simulación, Interacción, Cargas.

---

## 1. Introducción

Un adsorbente es un sólido que tiene la capacidad de retener en su superficie componentes presentes en un medio líquido o gaseoso, son comúnmente usados para la eliminación de contaminantes en un medio. El adsorbente mayormente utilizado es el carbón activado el cual se puede sintetizar de cualquier compuesto orgánico llevado a pirólisis, también los adsorbentes poliméricos son muy comunes.

Existen tres tipos de adsorción, la química cuando el adsorbente forma enlaces con el adsorbato, la física que es regida por fuerzas de Van Der Waals y la adsorción por intercambio que es cuando los iones de cierta sustancia se atraen a la superficie del adsorbente por fuerzas electrostáticas.

Para determinar la eficiencia de un adsorbente es

necesario analizar los aspectos cinéticos y termodinámicos [2], el análisis cinético de un adsorbente nos indica la capacidad que tiene el adsorbente, es decir, la eficiencia en cuanto a rendimiento de adsorción.

---

## 2. Antecedentes

Se sabe que el uso de adsorbentes para eliminación de contaminantes en medios gaseosos y líquidos son muy estudiados y existe gran cantidad de artículos enfocados a estos, más en el área ambiental [1]. En el caso de carbón activado como adsorbente las investigaciones van enfocadas más a la fuente del carbón activado para su capacidad de adsorción, en cambio los adsorbentes de intercambio iónico también son de gran estudio ya que evitan las reacciones en

el medio, por lo cual solo adsorben las partículas de cierta carga y se mantienen en la superficie del adsorbente.

### 3. Herramientas

La simulación se realizó en una laptop ASUS modelo X507M con un procesador Intel Celeron N4000. Haciendo uso del software *R Project versión 3.5.3 (2019-03-11)*. Como base se usó la práctica 9 de R paralelo: simulación y análisis de datos de la Dra. Shaeffer [3].

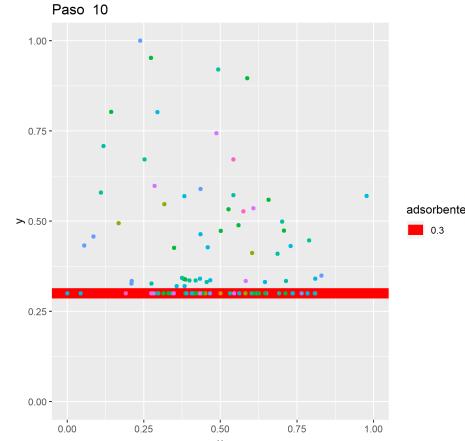
### 4. Experimentación

Se crearon cien partículas cargadas aleatoriamente de -5 a 5, el algoritmo se modificó de tal forma de darle carga positiva al adsorbente de esta manera que las partículas con carga negativa fueran atraídas hacia el adsorbente. Se varió la fuerza de atracción de las partículas con valores de 0.02, 0.05 y 0.1, con 50 iteraciones y 3 réplicas para cada una, para medir la cinética se tomaron en cuenta los pasos con valores de 2, 5, 10, 15, 20, 30, 40 y 50, de manera de obtener la cantidad de partículas adsorbidas en cada paso, como prueba estadística se uso del ANOVA tomando los resultados del paso 10 para cada valor de fuerza.

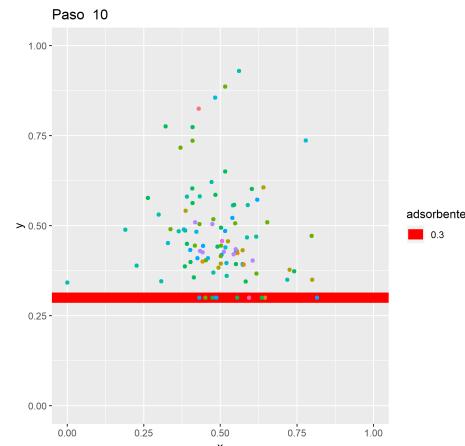
### 5. Resultados y discusión

En la figura 1 se pueden ver las partículas adsorbidas para cada valor de fuerza de atracción en el paso 10 de la cinética donde en la fuerza de atracción 0.02 es donde menos adsorbe.

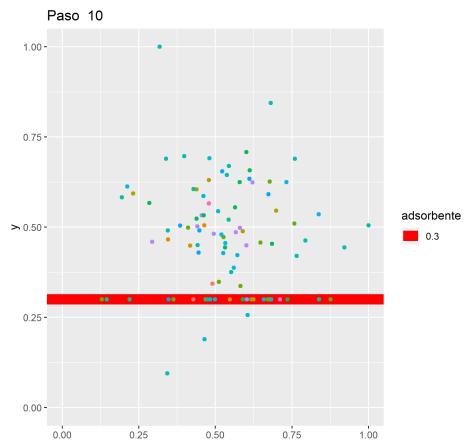
Se puede observar en la figura 2 que en donde hubo más partículas adsorbidas fue para la fuerza de atracción de 0.05, siguiendo de 0.02 y por último 0.1. También se puede observar que para la fuerza de atracción de 0.02 la adsorción de partículas incrementó considerablemente a partir del paso 15,



(a) Fuerza de atracción 0.05



(b) Fuerza de atracción 0.02



(c) Fuerza de atracción 0.1

Figura 1: Partículas adsorbidas en el paso 10.

en cambio para la fuerza de atracción de 0.05 fue una adsorción gradual.

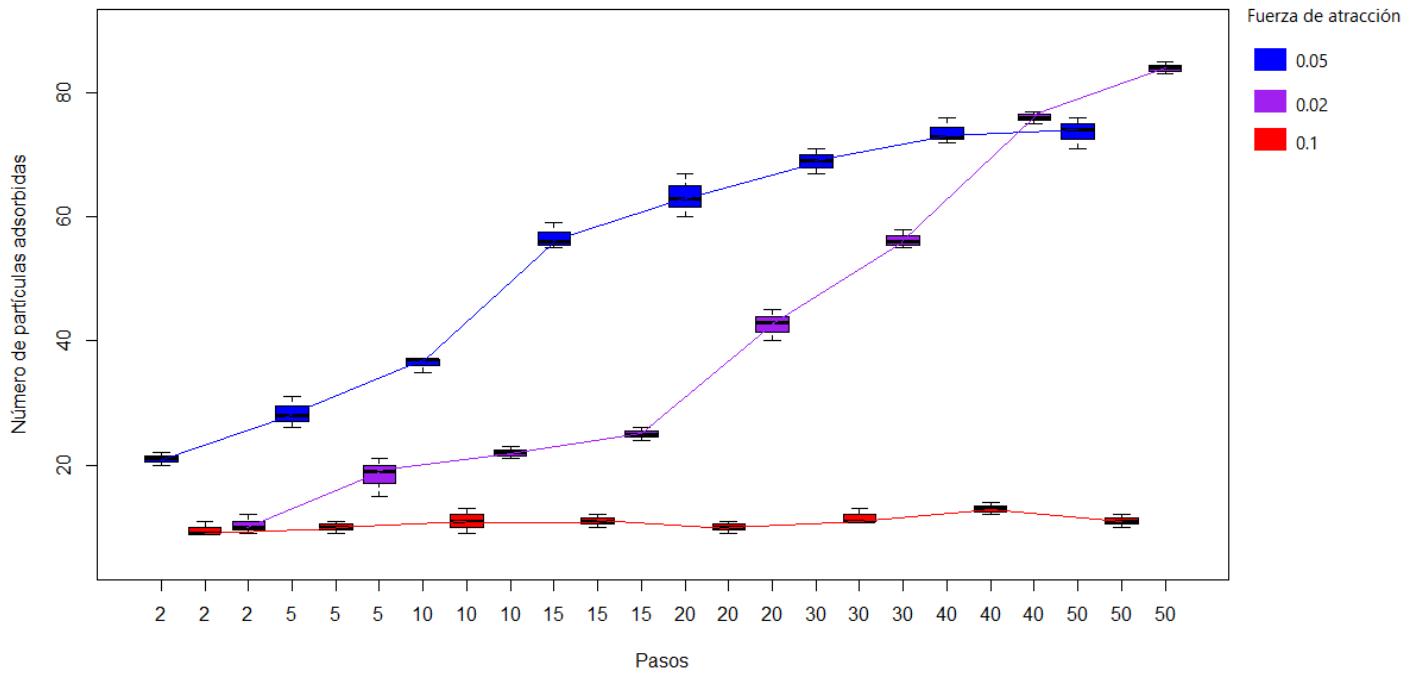


Figura 2: Cinética del adsorbente catiónico.

Para la fuerza 0.1 se puede decir que la cantidad de partículas adsorbidas fueron casi las mismas en todos los pasos, de manera que lo que adsorbe al inicio es lo más que podrá adsorber a través del tiempo, esto es debido a la alta fuerza de atracción, es decir, a la alta carga catiónica del adsorbente, al concentrarse gran cantidad de cargas negativas cerca del adsorbente, esta misma fuerza en conjunto hace que se repelen entre ellas y no se adsorban, se podría pensar que la adsorción sería más eficiente con mayor cantidad de cargas positivas en el adsorbente, pero aquí se puede apreciar que no.

Como resultados del ANOVA se obtuvo un valor  $F$  de 5.14 y un valor  $P$  de 0.0047 tomando un valor de significancia de 0.05, se puede decir que las fuerzas afectan significativamente al comportamiento del adsorbente.

## 6. Conclusiones

La cantidad de carga catiónica (fuerza de atracción) en un adsorbente catiónico determina si el adsorbente tendrá eficiencia, esto se puede saber a través de un análisis cinético del adsorbente, donde se sabe la capacidad de adsorción del mismo. En este caso se obtuvo que a una carga catiónica de 0.05 la adsorción de partículas fue gradual a través del tiempo, pero a cargas cationicas grandes el efecto de adsorción no es efectivo.

Se podría hacer más análisis cinéticos y variar las fuerzas con otros valores, incluir algún modelo matemático como base de la experimentación, incluso se podría tomar en cuenta la masa en el efecto de adsorción y la porosidad del adsorbente para ver de que manera afecta al rendimiento.

Agradecimientos Se agradece a Martín Rosas por ayudar en la idea principal de este proyecto, también a Marcos Guajardo por asesoría en el algorit-

mo de la simulación.

## Referencias

- [1] H. Qiu and L. Lv. Critical review in adsorption kinetic models. journal of zhejiangcatinic and anionic dye adsorption by agricultural solid wastes: A comprehensive review. *Journal of Zhejiang University-SCIENCE*, 10:716–724, 2009.
- [2] M. Salleh and D. Mahmoud. Cationic and anionic dye adsorption by agricultural solid wastes: A comprehensive review. *Desalination*, 280:1–13, 2011.
- [3] Elisa Schaeffer. Práctica 9: Interacciones entre partículas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.