



UANL

**UNIVERSIDAD AUTÓNOMA
DE NUEVO LEÓN**

**FACULTAD DE INGENIERÍA
MECÁNICA Y ELÉCTRICA**



FIME

**MAESTRÍA EN CIENCIAS DE LA INGENIERÍA CON
ORIENTACIÓN EN NANOTECNOLOGÍA**

Simulación Computacional de Nanomateriales

Portafolio de prácticas 1–12 y proyecto final

PRESENTA:

Ing. Ricardo Rosas Macías

INVESTIGADOR:

Dra. Satu Elisa Schaeffer

San Nicolás de los Garza, N.L.

Martes, 21 de Mayo de 2019

Práctica 1: movimiento Browniano

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la primera práctica que realice tuve problemas para concluirla debido a que no contaba con noción del software *R studio* y *LATEX*; fue la primera vez que trabajaba con ambos programas, por consecuencia eso disminuyó el tiempo para concluir la práctica. Por otro lado, me pareció muy buen tema a tratar; muy habitual en el campo de la nanotecnología y nanociencia.

La primera versión que entregue tenía errores en la presentación de resultados, pero conforme realice más prácticas, fui adquiriendo experiencia para reportar los resultados de una manera correcta. Asimismo en base a la experiencia adquirida, pude realizar ambos retos.

3194 3

Práctica 1. Movimiento Browniano

Ricardo Rosas Macías

23 de Enero de 2019

Resumen

Con fines educativos se realizó la práctica con el fin de observar la trayectoria que puede tener el movimiento de una nanopartícula o mejor conocido como movimiento Browniano. Con ayuda del software R nos permitió emular el movimiento de manera aleatoria, asimismo medir de manera cuantitativa su dirección Euclíadiana y distancia Manhattan.

[Palabras clave: Nanopartícula, Movimiento Browniano, dirección Euclíadiana, distancia Manhattan]

1. Introducción

Los movimientos pueden ser de muchos tipos distintos, pero en esta práctica nos limitamos a un caso sencillo donde la partícula mueve en pasos discretos, es decir, cada paso mide lo mismo, y las únicas posibles direcciones de movimiento son las direcciones paralelas a los ejes cardinales del sistema de coordenadas en el cual se realiza el movimiento. Vamos a utilizar pasos unitarios (es decir, el paso mide uno), teniendo como la posición inicial de la partícula el origen. [1]

2. Objetivo

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

2.1. Descripción

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

```

113 repetir <- 30
114 duracion <- 2**6 #Se cambio el exponente de 6 hasta 12
115 euct <- FALSE
116 library(parallel)

```

Figura 1: ParSapply

```

62 repetir <- 30
63 duracion <- 2**6 #Se cambio el exponente de 6 hasta 12

```

Figura 2: Boxplot

?

\label{\{dur\}}

3. Resultados

Para obtener los resultados se le realizaron modificaciones al código en el software R, como se muestra en la parte inferior. Se cambio los valores en el Script de parSapply1 y los valores de la rutina de Boxplot2, asimismo se logró obtener las graficas de la siguiente manera:

4. Conclusiones

en la figura \ref{\{dur\}}

De acuerdo a los diagramas de caja-bigote indica que los datos tienen más dispersión central, por lo que hay algunos datos atípicos que aparecen en forma de puntos en los gráficos que están fuera del rango intercuartílico. Por lo tanto hay más probabilidad de que los dos parámetros afectan al comportamiento de la nanopartícula.??

\texttt{\text{+} \{fig\}}

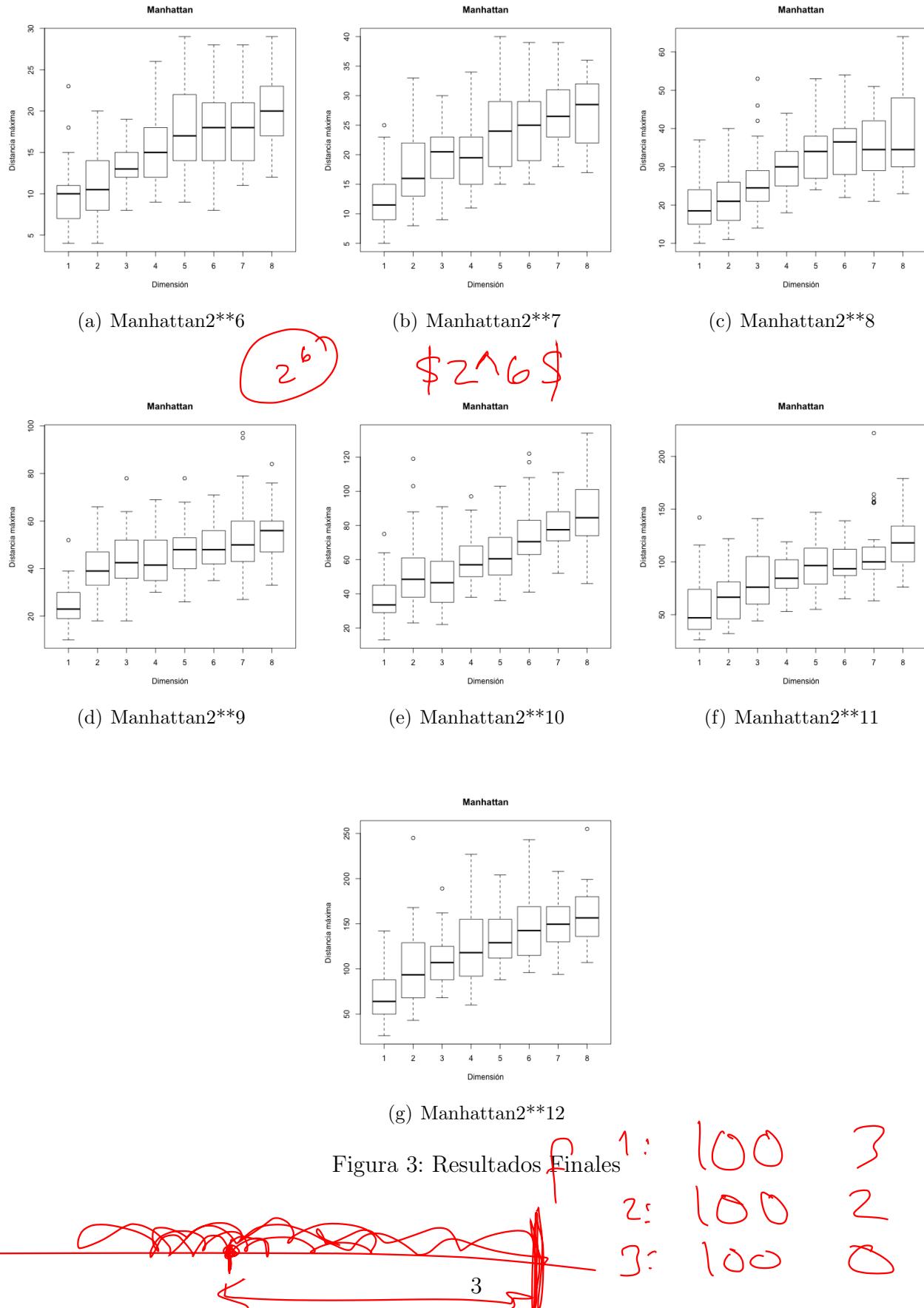
Referencias

[1] 01 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p1.html>.

```

auth = { ... },
title = { --- },
url = [
year:

```



Práctica 1. Movimiento Browniano

Ricardo Rosas Macías

21 de mayo de 2019

Resumen

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

1. Introducción

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

2. Objetivo

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

2.1. Descripción

Lo que se debe hacer es [5]:

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

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

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

3. Resultados y conclusiones

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

```

1 rep <- 30
2 pasos <- sapply(6:12, function(x) {2**x})
3 eucl <- FALSE
4
5 cluster <- makeCluster(detectCores() - 1)
6 clusterExport(cluster, "dur")
7 clusterExport(cluster, "eucl")
8 clusterExport(cluster, "pasos")
9 datos <- data.frame()
10 for (dur in pasos) {
11   for (dimension in 1:8) {
12     counter <- 0
13     clusterExport(cluster, "dimension")
14     resultado <- parSapply(cluster, 1:rep,
15       function(r) {
16         pos <- rep(0, dimension)
17         test <- rep(0, dimension)
18         mayor <- 0
19         counter <- 0
20         for (t in 1:duration) {
21           cambiar <- sample(1:dimension, 1)
22           cambio <- 1
23           if (runif(1) < 0.5) {
24             cambio <- -1
25           }
26           pos[cambiar] <- pos[cambiar] + cambio
27           if (eucl) {
28             d <- sum(sqrt(pos**2))
29           } else { # Manhattan
30             d <- sum(abs(pos))
31           }
32           if (d > mayor) {
33             mayor <- d
34           }
35           if (all(pos==test)) {
36             counter <- counter + 1

```

```

37 }
38     }
39     return(counter)
40   })
41   datos <- rbind(datos, resultado)
42 }
43 }
44 stopCluster(cluster)
45 Adata <- data.frame(datos)

```

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

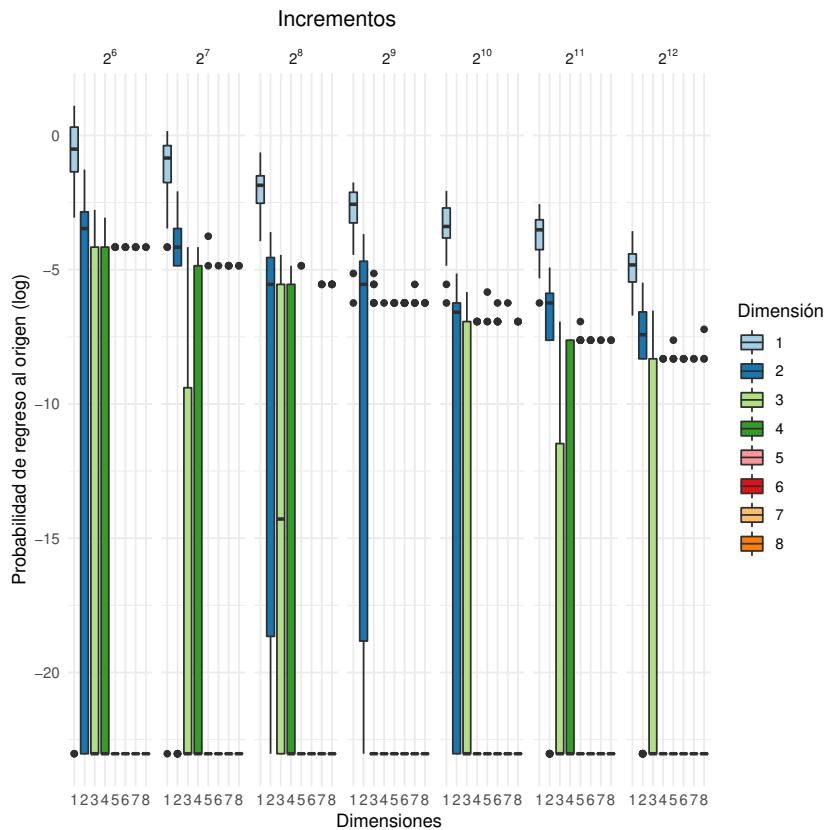


Figura 1: Distancia Manhattan de partícula en incrementos de 2^6 a 2^{12}

3.1. Reto 1

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

```

1 rep <- 1
2 dur <- 100
3 dim <- 8
4 orig <- rep(0, dim)
5 tab <- numeric(length(rep*dim))
6 elapsed <- numeric(length(dur))
7 i = 1
8 j = 1
9 anterior <- 0
10 timeT <- system.time(
11   for (dimension in dim:dim) {
12     timeDim <- system.time(
13       for (repeticion in 1:rep) {
14         cero <- 0
15         pos <- rep(0, dimension)
16         for (t in 1:dur) {
17           timeD <- system.time(
18             for (x in 1:1) {
19               cambiar <- sample(1:dimension, 1)
20               if (runif(1) < 0.5) {
21                 cambio <- 1
22               } else {
23                 cambio <- -1
24               }
25               pos[cambiar] <- pos[cambiar] + cambio
26               if (all(pos == orig)) {
27                 cero = cero + 1
28               }
29             })[3]
30           time <- (timeD + anterior)
31           elapsed[j] <- time
32           anterior <- time
33           j = j + 1
34         }
35         prob <- ((cero/dur)*100)
36         tab[i] <- prob
37         i = i + 1
38       })[3]
39   })

```

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

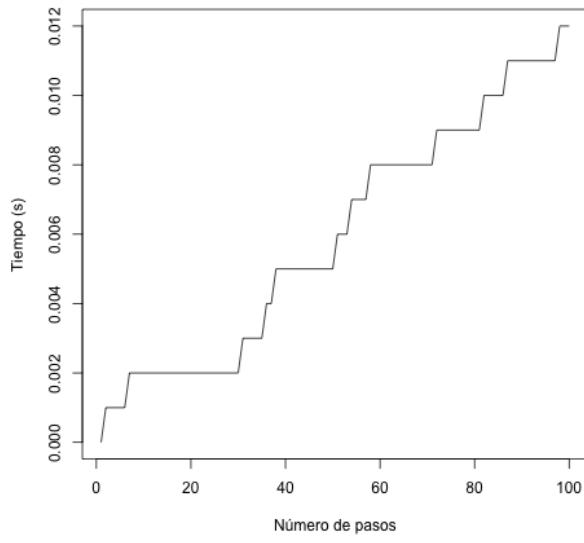


Figura 2: Porcentaje de soluciones

3.2. Reto 2

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

```

1 library(parallel)
2
3 PSprocess <- Sys.time()
4 repetir <- 30
5 dur <- 200
6 dimen <- 1:8
7
8 cluster <- makeCluster(detectCores() - 1)
9 clusterExport(cluster, "dur")
10 datos <- data.frame()
11
12 for (dim in dimen) {
13   clusterExport(cluster, "dim")
14   resultado <- parSapply(cluster, 1:repetir,
15                         function(x) {
16   pos <- rep(0, dim)
17   origen <- rep(0, dim)
18   contador <- 0
19   for (t in 1:dim) {
20     cambiar <- sample(1:dim, 1)
21     cambio <- 1
22     if (runif(1) < 0.5) {
23       cambio <- -1
24     }
25     pos[cambiar] <- pos[cambiar] + cambio
26     d <- dist(pos)
}

```

```

27         if (sum(pos == origen) == dim) {
28             contador <- contador + 1
29         }
30     }
31     return(contador)
32 }
33 )
34 datos <- rbind(datos, resultado)
35 }
36 stopCluster(cluster)
37
38 PEmprocess <- Sys.time()
39 TimeP <- PEmprocess - PSprocess
40 print(TimeP)
41
42 Sprocess <- Sys.time()
43 datos <- data.frame()
44 for (dim in dimen) {
45     resultado <- sapply(1:repetir,
46                         function(x) {
47                             pos <- rep(0, dim)
48                             origen <- rep(0, dim)
49                             contador <- 0
50                             for (t in 1:dur) {
51                                 cambiar <- sample(1:dim, 1)
52                                 cambio <- 1
53                                 if (runif(1) < 0.5) {
54                                     cambio <- -1
55                                 }
56                                 pos[cambiar] <- pos[cambiar] + cambio
57                                 d <- dist(pos)
58                                 if (sum(pos == origen) == dim) {
59                                     contador <- contador + 1
60                                 }
61                             }
62                         }
63             )
64 }
65 datos <- rbind(datos, resultado)
66 }
67
68 Eprocess <- Sys.time()
69 TimeSP <- Eprocess - Sprocess
70 print(TimeSP)
71
72 vstime <- data.frame(TimeSP, TimeP)
73 print(vstime)
74 ftc <- kruskal.test(vstime)
75 ftc$p.value
76 ftc$p.value < 0.05

```

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

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

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

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

Referencias

- [1] Jodie Burchell. Creating plots in r using ggplot2 - part 10: boxplots, 2016. URL <http://t-redactyl.io/blog/2016/04/creating-plots-in-r-using-ggplot2-part-10-boxplots.html>.
- [2] Astrid González. Exercise1, 2018. URL https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_1/.
- [3] Alboukadel Kassambara. ggplot2 box plot : Quick start guide - r software and data visualization, 2015. URL <http://www.sthda.com/english/wiki/ggplot2-box-plot-quick-start-guide-r-software-and-data-visualization>.
- [4] Ricardo Rosas Macías. Práctica 1: Movimiento browniano, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP1>.
- [5] Elisa Schaeffer. Práctica 1: Movimiento browniano, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p1.html>.
- [6] José Ángel Cavazos Contreras. P1, 2018. URL https://github.com/Xhangelx13x/SimNano_Xhan/blob/master/P1/Reporte_P1.

Práctica 2: autómata celular

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la segunda práctica al igual que en la primera se denotan fallos en el reporte de resultados, además de una ausencia de demostración cuantitativa; un soporte estadístico que permita discernir si el código cumple con los objetivos planteados. En esta práctica tarde un poco en comprender el objetivo debido a que no estaba muy familiarizado con este tema, pero la verdad me pareció interesante, igualmente creo que tiene una gran aplicación en la simulación de nanomateriales con propiedades antibacteriales; por ejemplo: Oro, Plata y Cobre.

No pude realizar el primer reto debido a que no comprendí como hacer que el modelo STL cumpla con los objetivos planteados y con lo que respecta al segundo reto, no entendí el como usar *Python3* para realizar el modelo 3D. En un futuro seguiré realizando código para irme familiarizando aún más los lenguajes que se pueden usar para simulación.

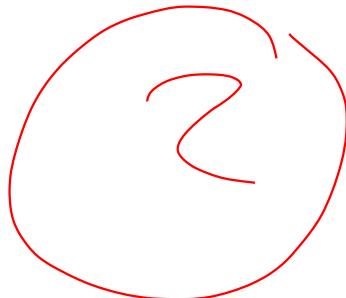
3194

Práctica 2: Autómata celular

Ricardo Rosas Macías

5 de febrero de 2019

Resumen



El experimento es un modelo matemático llamado autómata celular, este realiza una secuencia parecida a una máquina de Turing que puede ser representada matemáticamente con una matriz booleana para así obtener la interacción que esta tiene a través del tiempo.

[Palabras clave: Autómata celular, máquina de Turing, matriz booleana]

1. Introducción

Los autómatas celulares surgen en la década de 1940 con John Von Neumann, que intentaba modelar una maquina que fuera capaz de autoreplicarse, llegando así a un modelo matemático de dicha maquina con reglas complicadas sobre una red rectangular. Inicialmente fueron interpretados como conjunto de células que crecían, se reproducían y morían a medida que pasaba el tiempo. A esta similitud con el crecimiento de las células se le debe su nombre[1].

2. Objetivo

Con fines educativos se realizó la práctica con la finalidad de ver la supervivencia celular que esta tiene, asimismo con el procesamiento de datos permitirá mostrar la evolución celular que tiene el experimento.

2.1. Descripción

Lo que se debe hacer es [2]:

EVITA EL ROLLO

“Diseño y ejecución de un experimento para determinar el número de iteraciones que procede la simulación en una malla de 30 por 30 celdas hasta que se mueran todas, variando la probabilidad inicial de celda viva entre cero y uno en pasos de 0.10.”

3. Resultados

arreglos de listas de los resultados
En términos generales para lograr con el objetivo, se realizó los cambios en el código de manera que proporcionó las interacciones que tienen en la matriz.

```
p <= 0.1      0.0, 1.0  
for (p in seq(0.1, 0.9, by=0.1)) {library(parallel)  
dim <- 30
```

```
for (iteracion in 1:20) {  
  MMAN    CoIA
```

Asimismo con la ayuda del servicio de Giphy se creó una imagen.

4. Conclusiones

Referencias

- [1] David Alejandro Reyes Gómez. Descripción y aplicaciones de los autómatas celulares. *Verano de Investigación 2011*, pages 3–4, 2011.
- [2] Elisa Schaeffer. *Algo que no me acuerdo*, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p1.html>.

ponle aguas para que sean más fáciles del giphy

Práctica 2. Autómata celular

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

El experimento es un modelo matemático llamado autómata celular, este realiza una secuencia parecida a una máquina de Turing que puede ser representada matemáticamente con una matriz booleana para así obtener la interacción que esta tiene a través del tiempo.

2. Objetivo

El interés en el trabajo presente es examinar la supervivencia celular, durante la evolución celular que tiene el experimento. Asimismo, observar cómo la probabilidad afecta en la supervivencia.

2.1. Descripción

Se busca [2]:

“Diseñar y ejecutar un experimento para determinar el número de iteraciones que procede la simulación en una malla de 30 por 30 celdas hasta que se mueran todas, variando la probabilidad inicial de celda viva entre cero y uno en pasos de 0.10.”

3. Resultados y conclusiones

Para realizar el código se tomo lo anteriormente reportado [1][3]. En las líneas de código de la parte inferior se determinaron los parámetros con los que se ejecuta para variar la probabilidad de supervivencia.

```
1 dim <- 30
2 num <- dim^2
3 limit <- 10
4 repeatexp <- 30
5 probability <- seq(from=0.1, to=0.9 ,by=0.10)
6
7 results <- data.frame()
8
```

```

9 | for(p in probability){
10| iterations <- list()
11| actual <- matrix(1 * (runif(num) < p) , nrow=dim , ncol=dim)
12| suppressMessages(library("sna"))
13| for (rep in 1:repeatexp){
14|   if(sum(actual) == 0){
15|     iterations <- c(iterations , 0)
16|     iterations <- unlist(iterations)
17|   } else if(sum(actual) == num){
18|     iterations <- c(iterations , 0)
19|     iterations <- unlist(iterations)
20|   }
21|
22|   iterations <- c(iterations , iteracion)
23|   iterations <- unlist(iterations)
24| }
25}
26 results <- rbind(results , iterations)
27 system("convert -delay 50 -size 300x300 p2_Repl_t*.png -loop 0 AC.gif")

```

Además, con la paquetería *ggplot2* se obtuvo la visualización de código en la figura [1], que muestra la distribución de los datos en donde se aprecia la densidad de probabilidad. Por consiguiente, la probabilidad 0.3, 0.4 y 0.5 tienen la misma expectativa de vida; de tal manera que no tienen límite de iteraciones y los agentes tienen menor auspicio a morir. Por otro lado, la probabilidad de 1 tiene la mayor posibilidad de estar viva, ya que esta tiene menor iteraciones con los vecinos.

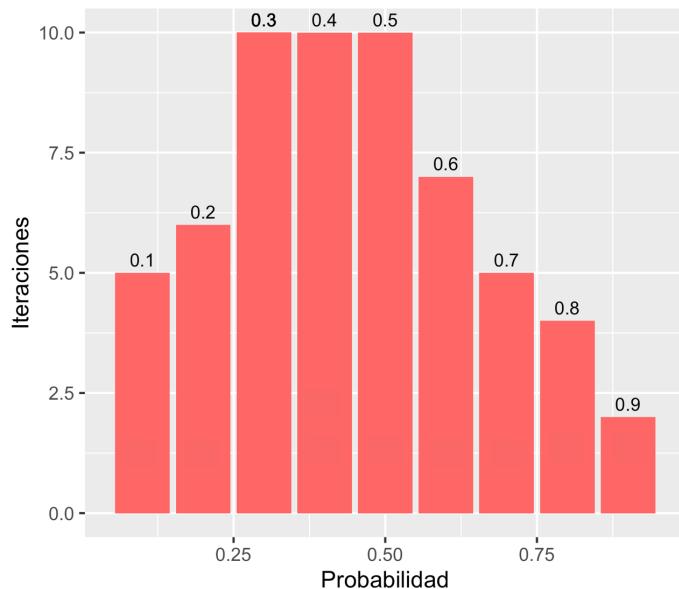


Figura 1: Probabilidad respecto a la iteración

Para observar el fenómeno descrito anteriormente, se creó un gif de las imágenes proporcionadas del experimento; en las líneas de la parte inferior del código, de modo que en la figura [2] se puede ver el inicio de los agentes en la matriz y la evolución rápida de los agentes respecto a sus vecinos.

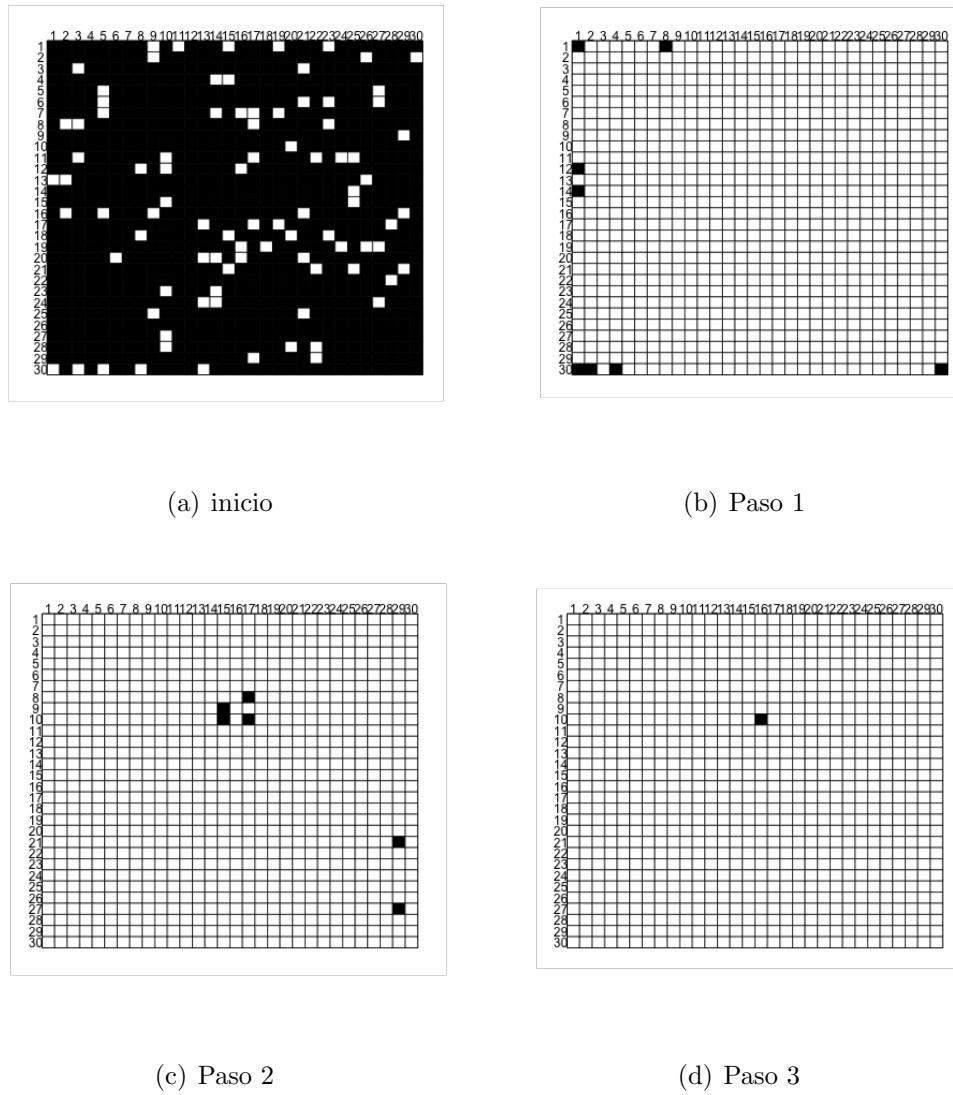


Figura 2: Matriz de evolución de agentes

Referencias

- [1] Ricardo Rosas Macías. Práctica 2. autómata celular, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP2>.
 - [2] Elisa Schaeffer. Práctica 2: Autómata celular, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p2.html>.
 - [3] Marco Antonio Guajardo Vigil. Celulasautonomas, 2019. URL <https://sourceforge.net/p/simulaciondesistemas/code/ci/master/tree/P2/>.

Práctica 3: teoría de colas

Ricardo Rosas Macías

21 de mayo de 2019

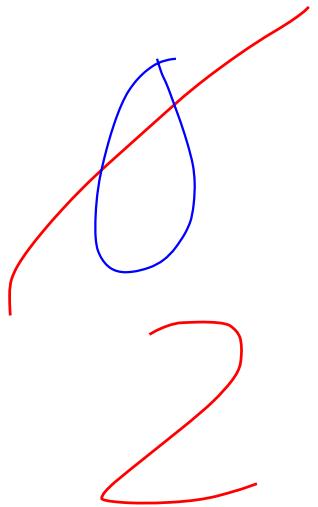
1. Reporte de aprendizaje

En la tercer práctica me costo trabajo hacer que el código importara los datos que nos proporcionaba la página web; dónde descargue los primos, de igual manera tuve algunos ligeros problemas para poner las gráficas enseguida del texto; esto ocasionó que el documento no tuviera resultados, por lo que se me asigno una calificación baja. Además, la falta de líneas de código en un cuadro de *listings* originó la idea que no se realizó la práctica.

Es preciso mencionar que la última imagen de la página web de la practica, ejemplifica de una manera muy sencilla lo que realiza el código de la página web.

Por último, en esta práctica logre aprender el manejo de la librería *ggplot2*, como se puede observar en el documento final; en donde empece a realizar cambios en colores. No pude realizar los retos, intente pero no tuve un resultado favorecedor, por lo que creo que en un futuro me ayudaría el leer y practicar algunos razonamientos para estimular el pensamiento matemático.

394



Práctica 3: teoría de colas

Ricardo Rosas Macías

12 de febrero de 2019

1. Introducción

La teoría de colas es un área de las matemáticas que estudia el comportamiento de líneas de espera. Los trabajos que están esperando ejecución en un cluster esencialmente forman una línea de espera. Medidas de interés que ayudan caracterizar el comportamiento de una línea de espera incluyen, por ejemplo, el tiempo total de ejecución. En esta práctica **vamos a estudiar** el efecto del orden de ejecución de trabajos y el número de núcleos utilizados en esta medida[1].

2. Objetivo

En términos generales para la lograr con el objetivo, se realizó los cambios en el código de manera que proporcionó el comportamiento que tienen la ejecución del trabajo de los núcleos al realizar una tarea.

2.1. Descripción

Lo que se debe hacer es [1]:

“Examinar cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al cluster, utilizando como datos de entrada un vector que contiene primos grandes; descargados de <https://primes.utm.edu/lists/small/millions/> y no-primos con un mismo número de dígitos. Investiga también el efecto de la proporción de primos y no primos en el vector igual como la magnitud de los números incluidos en el vector con pruebas estadísticas adecuadas..”

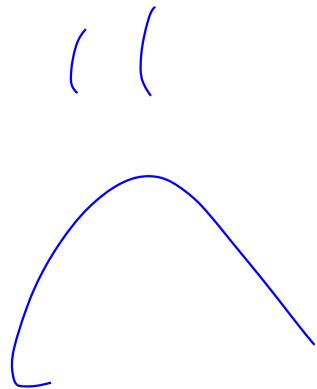
3. Resultados y conclusiones

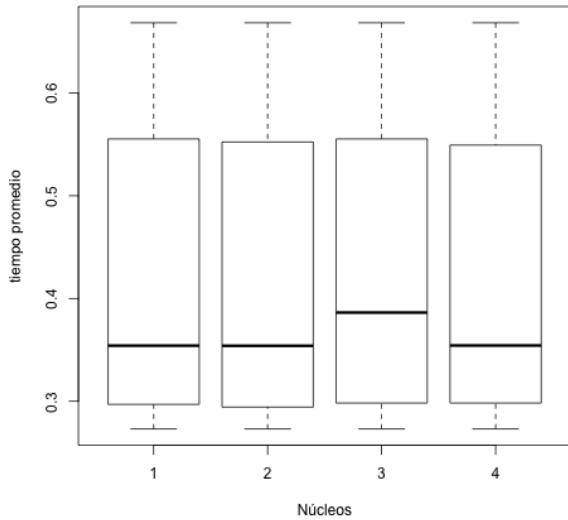
Realice la compilación del código de manera que me proporcionó los resultados finales?? Como se puede observar en las siguientes gráficas en donde el cambio de núcleos para realizar las tareas no es significativo, se denota el cambio al hacerlas 3 núcleos?? en donde la media de tiempo promedio sube un poco pero permanece debajo del segundo cuartil. En cuanto el orden ?? se muestra que se tarda menos

en analizar los datos de manera ascendente; como aparecen en el documento de texto que se descargo de la página web, sin embargo su proporc?? muestra algunos cambios en la matriz respecto al tiempo del ciclo. Finalmente podemos observar el numero de dígitos?? en el cual se distingue claramente que se tarda mucho más al realizar el análisis de números de 2 digitos.

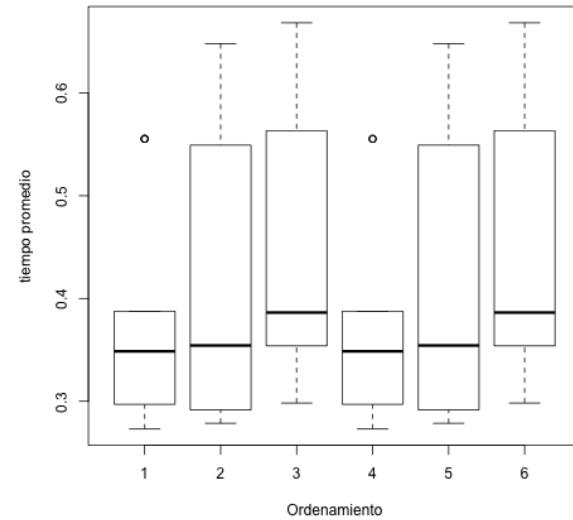
Referencias

- [1] Elisa Schaeffer. Práctica 3: teoría de colas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.

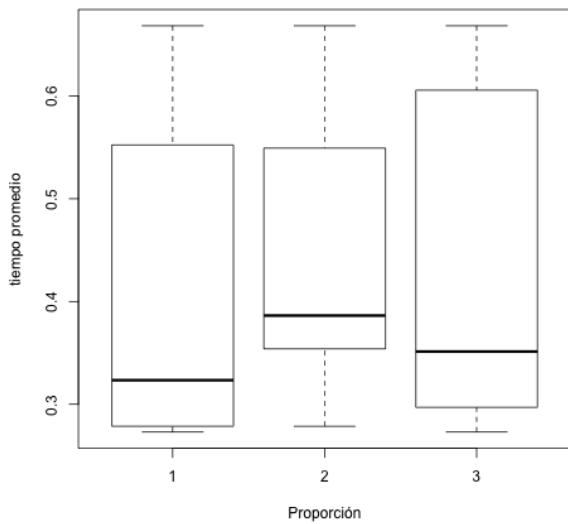




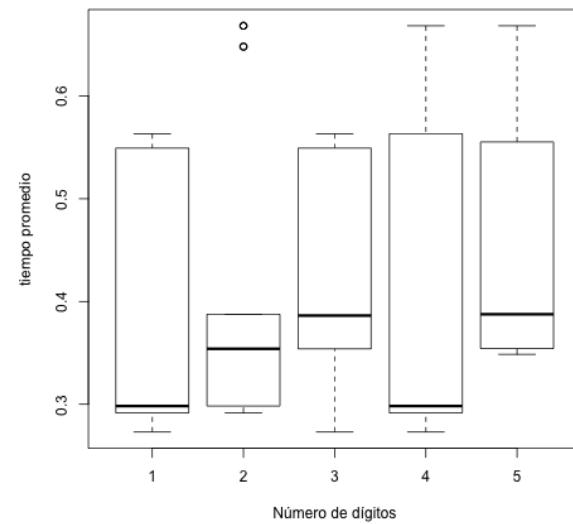
(a) Núcleos



(b) Orden



(c) Proporción



(d) Dígitos

Figura 1: Resultados finales

Práctica 3: teoría de colas

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

La teoría de colas o líneas de espera, es un análisis matemático que permite estudiar el comportamiento sistemático de un procedimiento que tiene la capacidad de procesar muchos datos o tareas, sin aletargamiento que arruine la obtención de los parámetros de interés.

2. Objetivo

En la práctica se busca examinar el arreglo de las tareas del experimento, obtener los tiempos que tardan en realizarse y su duración al ejecutarse con un número de núcleos determinado.

2.1. Descripción

Lo que se debe hacer es [3]:

“Examinar cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al cluster, utilizando como datos de entrada un vector que contiene primos grandes; descargados de <https://primes.utm.edu/lists/small/millions/> y no-primos con un mismo número de dígitos. Investiga también el efecto de la proporción de primos y no primos en el vector igual como la magnitud de los números incluidos en el vector con pruebas estadísticas adecuadas.”

3. Resultados y conclusiones

En términos generales para la lograr con el objetivo, se realizaron cambios en el código [1][2] de tal forma que proporcione el comportamiento que tiene la ejecución del experimento con la proporción de primos y no primos.

```
1 pd <- read.csv("primes1.txt", sep="", header = FALSE) #Primos descargados  
2 pd$V1 <- NULL  
3 pd$V10 <- NULL
```

```

4 pdm <- as.matrix(pd)
5 primos <- as.vector(t(pdm))
6
7 hasta<-20011
8 desde<-5381
9 hasta<-11827
10 replicas<-30
11 primo <- function(n) {
12   for (i in 2:(n-1)) {
13     if ((n > i && n %%i) == 0) { # residuo es cero
14       return(FALSE)
15     }
16   }
17   return(n)
18 }
19 primos <- numeric() # un vector vacio
20 for (n in desde:hasta) {
21   primos <- c(primos, primo(n)) # combinar vectores
22 }
23
24 noprimo <- function(n) {
25   for (i in 2:(n-1)) {
26     if ((n > i && n %%i) == 0) { # residuo es cero
27       return(n)
28     }
29   }
30   return(FALSE)
31 }
32 noprimos <- numeric() # un vector vacio
33 for (m in desde:hasta) {
34   noprimos <- c(noprimos, noprimo(m)) # combinar vectores
35 }
36
37 PC<-primos[which(primos>0)] # Primos crecientes
38 NPC<-noprimos[which(noprimos>0)] # No primos crecientes
39 PD<- sort(a, decreasing = TRUE) # Primos decrecientes
40 NPD<-sort(l, decreasing = TRUE) # No primos decrecientes
41 PA<-sample(a) # Primos aleatorios
42 NPA<-sample(d) # No primos aleatorios
43
44 e <- sample(a, size=420 ,replace = TRUE, prob = NULL)
45 f<-sample(l ,size=420, replace = TRUE,prob=NULL)
46
47 primx <- function(n) {
48   if (n == 1 || n == 2) {
49     return(TRUE)
50   }
51   if (n %%2 == 0) {
52     return(FALSE)
53   }
54   for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
55     if ((n %%i) == 0) {
56       return(FALSE)
57     }
58   }

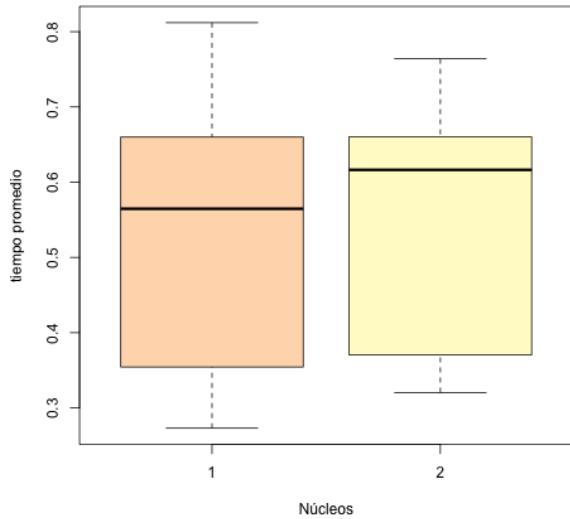
```

```

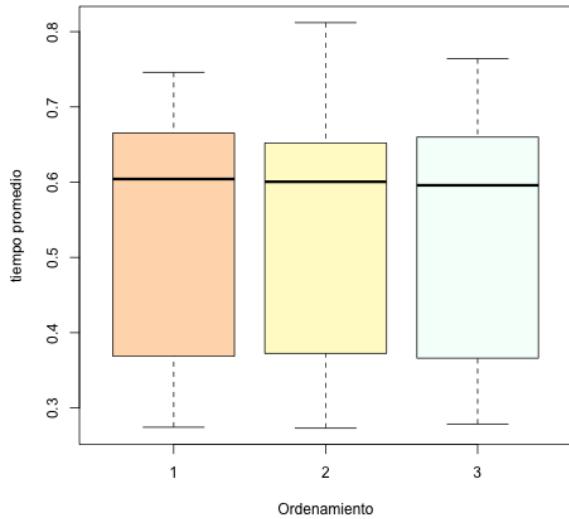
59     return (TRUE)
60 }
61
62 suppressMessages(library(doParallel))
63 registerDoParallel(makeCluster(detectCores()-2))
64 PCt <- numeric()
65 PDt <- numeric()
66 PAt <- numeric()
67 NPDt <- numeric()
68 NPCt <- numeric()
69 NPAt <- numeric()
70
71 for (r in 1:replicas) {
72   PCt <- c(at, system.time(foreach(n = a, .combine=c) %dopar % primx(n))[3])
73   PDt <- c(bt, system.time(foreach(n = b, .combine=c) %dopar % primx(n))[3])
74   PAt <- c(vt, system.time(foreach(n = v, .combine=c) %dopar % primx(n))[3])
75   NPDt <- c(dt, system.time(foreach(n = d, .combine=c) %dopar % primx(n))[3])
76   NPCt <- c(lt, system.time(foreach(n = l, .combine=c) %dopar % primx(n))[3])
77   NPAt <- c(wt, system.time(foreach(n = w, .combine=c) %dopar % primx(n))[3])
78 }
79 stopImplicitCluster()
80
81 mean(PCt)
82 mean(PDt)
83 mean(NPDt)
84 mean(NPCt)
85 mean(PAt)
86 mean(NPAt)
87 nucleos <- rep(1,20)
88 nucleos<-c(nucleos ,rep(2,20))
89 digitos<-rep(5,10)
90 digitos<-c(digitos ,rep(4,10))
91 digitos<-c(digitos ,rep(3,10))
92 digitos<-c(digitos ,rep(2,10))
93 digitos<-c(digitos ,rep(1,10))
94 proportion<- rep(1,3)
95 proportion<-c(proportion ,rep(2,3))
96 proportion<-c(proportion ,rep(3,3))
97 p<-c(1, 2, 3)
98 ordenamiento<-rep(p, 20)

```

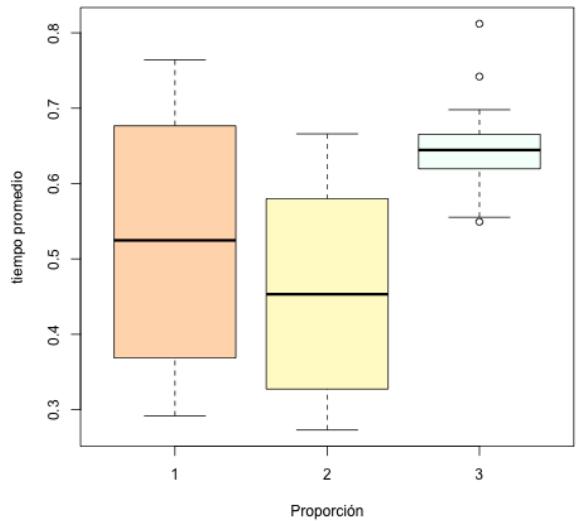
Se realizaron gráficos de la simulación como se muestra en la figura 1, en donde se puede observar en la figura 1(a) el cambio de núcleos para realizar las tareas es ligeramente significativo, la media de tiempo promedio sube un poco pero permanece debajo del segundo cuartil. En cuanto el orden, en la figura 1(b) se muestra que se tarda menos en analizar los datos de manera ascendente; como aparecen en el documento de texto que se descargo de la página web. Sin embargo, en la figura 1(c) muestra una proporción, donde hay algunos cambios en la matriz respecto al tiempo del ciclo. Finalmente podemos observar en la figura 1(d) el número de dígitos en el cual se distingue claramente que se tarda mucho más al realizar el análisis de números de 2 dígitos.



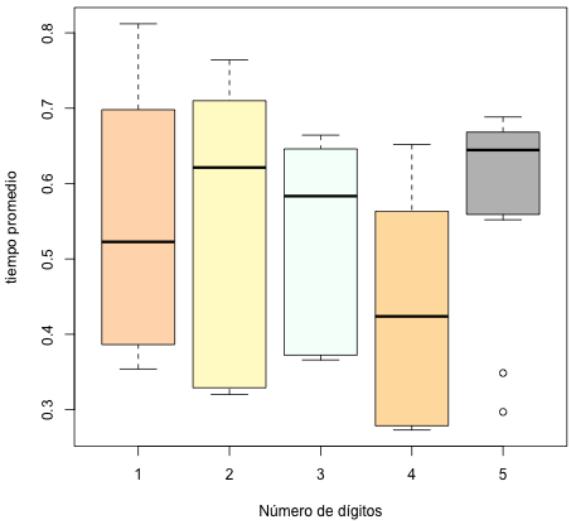
(a) Núcleos



(b) Orden



(c) Proporción



(d) Dígitos

Figura 1: Resultados de ejecución del experimento

Referencias

- [1] Ricardo Rosas Macías. Práctica 3: teoría de colas, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP3>.
- [2] Liliana Saus. p3 teoría de colas, 2018. URL <https://github.com/pejli/simulacion/tree/master/P3>.
- [3] Elisa Schaeffer. Práctica 3: teoría de colas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.

Práctica 4: diagramas de Voronoi

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la cuarta práctica entendí perfectamente el objetivo, puesto que relacione a la Teselación de Voronoi con la microestructura de un metal que ha sido preparado por pulido mecánico; en el cual se observa un límite de grano, asimismo tome a las semillas como centros de nucleación.

Este tema tiene una gran aplicación en nanomateriales, creo que uno de ellos podría ser para el segundo reto, en donde estaría bien ver el comportamiento de la propagación de la grieta debido a la rotación de nanogranos. Para más información al respecto, dejo el link: <https://pubs.acs.org/doi/10.1021/nl061775u>

Por último cabe destacar que en esta práctica hubo un buen razonamiento y aplicación de lo anteriormente visto; debido al uso de la distancia Manhattan para obtener la distancia de la grieta. En el documento final se realizaron los cambios en los errores marcados, además se agregó las líneas del código que estaban ausentes en el documento anterior y una justificación estadística que permitió observar de una manera cuantitativa el comportamiento del experimento.

Práctica 4: diagramas de Voronoi

Ricardo Rosas Macías

19 de febrero de 2019



1. Introducción

El mapa de Voronoi o Teselación de Voronoi es un conjunto de regiones en el plano es la división de dicho plano en regiones de tal forma, que a cada punto se asigna una región del plano formada por los puntos que son más cercanos a él que a cualquier otro de los otros objetos.¹ Dicho de otra manera, lo que hace dicho diagrama es dividir el plano en tantas regiones como puntos u tengamos de tal forma que a cada punto le asignemos la región formada por todo lo que está más cerca de él que de ningún otro.”[1].

2. Objetivo

En términos generales para la lograr el objetivo, se realizó los cambios en el código de manera que proporcionó el comportamiento que tiene la grieta al realizar cambios en las variables.

2.1. Descripción

Lo que se debe hacer es [2]:

“Examinar de manera sistemática el efecto del número de semillas y del tamaño de la zona en la distribución de los largos de las grietas que se forman con dos medidas para el largo

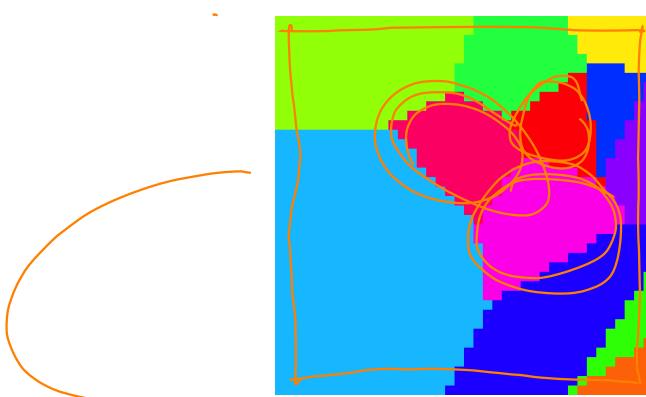


Figura 1: Teselación de Voronoi

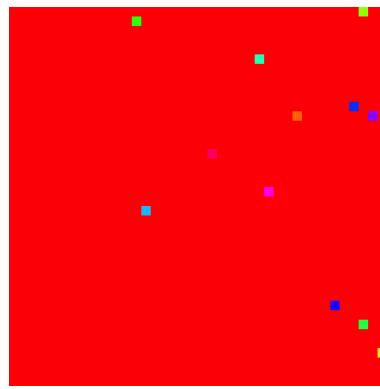


Figura 2: Crecimiento de semillas

de una grieta: el número de celdas que contiene la grieta y la mayor distancia Manhattan entre la grieta y el borde del cuadro.”

3. Resultados y conclusiones

Se llevó acabo el experimento con las siguientes condiciones: para el acomodo de las semillas se realizó de manera aleatoria en conjuntos de 10 – 40 en intervalos de 10, de la misma manera la cuadricula se ejecutó en 50 – 200 en intervalos de 50. Con los cuales se realizó un epresentación grafica con la serie de datos obtenidos de la corrida, por consiguiente se puede observar un decremento en la distancia manhattan y grieta al tener una cuadricula de 150 en relación al tamaño de la zona en comparación con las demás cuadriculas que en muestran una afinidad normal entre la grieta y el borde del cuadro 3. Por otro lado, la compilación del código proporcionó el crecimiento de la grieta en una serie de diagramas de Voronoi 4.

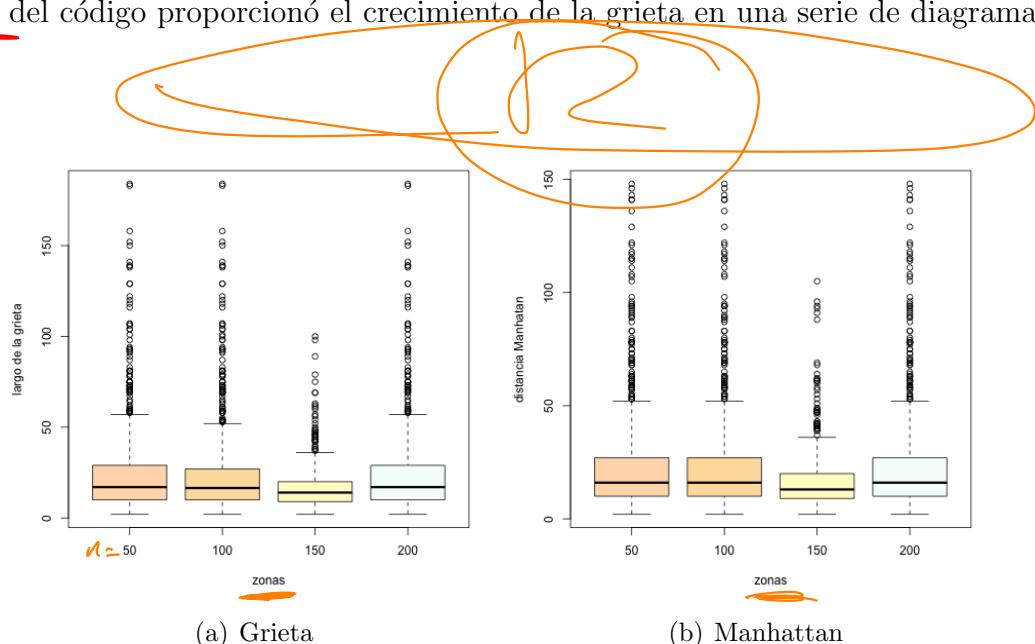


Figura 3: Zonas

$$n-d_M$$

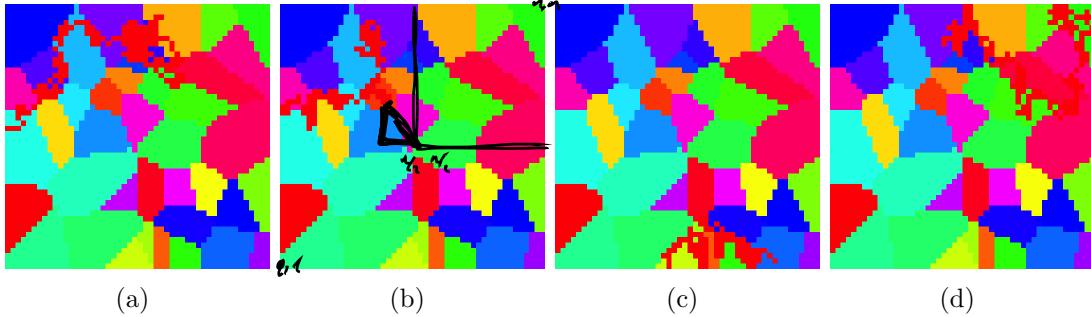


Figura 4: Diagramas de Voronoi

En la figura 5 y 6 se aprecia la afinidad de disminución en las zonas con un mayor tamaño. Por lo que la variación de los datos repercuten significativamente en el largo de la grieta final.

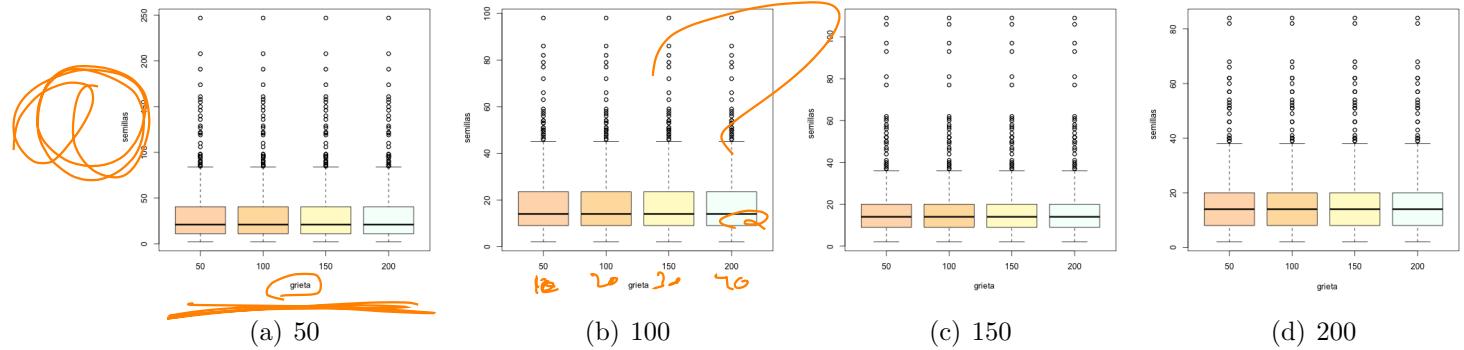


Figura 5: Largo de la grieta

Referencias

- [1] Clara Grima. El diagrama de Voronoi, la forma matemática de dividir el mundo, 1980. URL https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101_noticia.html.
- [2] Elisa Schaeffer. Práctica 4: diagramas de voronoi [SV]. 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p4.html>.

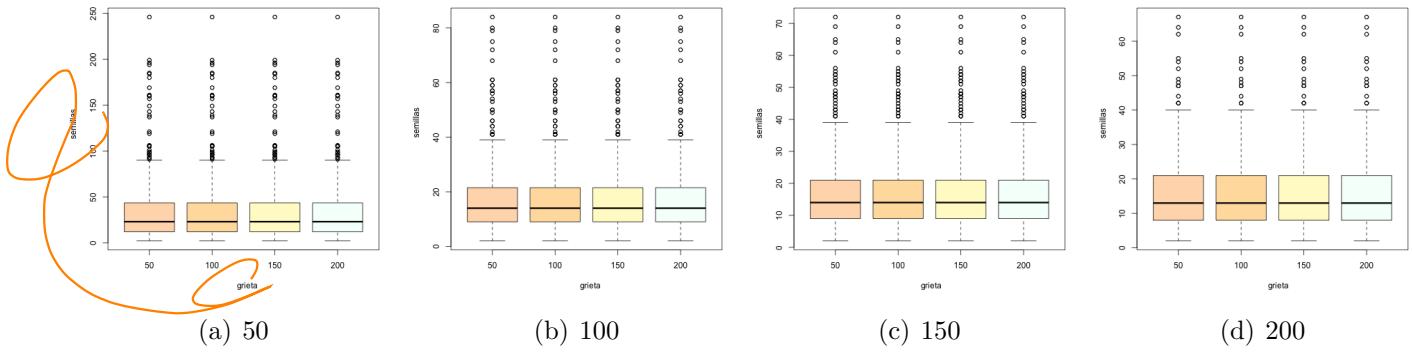


Figura 6: Distancia Manhattan

Práctica 4: diagramas de Voronoi

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

Los diagramas de Voronoi [2] o también conocidos como polígonos de Thiessen, son una observación básica en la Geometría Computacional, estos son un agregado de puntos en un plano, que tiene una región delimitada por el tamaño de los vecinos; por ende cada punto almacena toda la información acorde a la cercanía entre los demás puntos de su alrededor.

2. Objetivo

El objetivo principal de la práctica es determinar el comportamiento de la grieta respecto a la variación de los parámetros que generan la Teselación de Voronoi, asimismo examinar de una manera cuantitativa la varianza de los factores que alteran el tamaño de la grieta.

2.1. Descripción

Lo que se debe hacer es [4]:

“Examinar de manera sistemática el efecto del número de semillas y del tamaño de la zona en la distribución de los largos de las grietas que se forman con dos medidas para el largo de una grieta: el número de celdas que contiene la grieta y la mayor distancia Manhattan entre la grieta y el borde del cuadro.”

3. Resultados y conclusiones

Para llevar acabo el experimento se realizó cambios en el código [3] con ayuda de la literatura [1]. En las líneas de código que se muestran debajo, en donde el acomodo de las semillas se realizó de manera aleatoria en conjuntos de 10 – 50 en intervalos de 10, de la misma manera la cuadricula se ejecutó en 50 – 300 en intervalos de 50. En la parte inferior se plantea la ejecución de un análisis de varianza; de este modo proporcionó el comportamiento que tiene la grieta al realizar cambios en las variables. Además se hizo un Gif para tener una visualización del cambio del tamaño de la grieta.

```

1 suppressMessages(library(doParallel))
2 registerDoParallel(makeCluster(detectCores() - 1))
3 Manhattan <- TRUE
4 N <- c(50, 300, 50)
5 K <- c(10, 50, 10)
6 Info <- data.frame()
7 Largo <- c()
8 Nys <- c()
9 Kys <- c()
10 for (n in N) {
11   for (k in K) {
12     zona <- matrix(rep(0, n * n), nrow = n, ncol = n)
13     x <- rep(0, k)
14     y <- rep(0, k)
15
16     for (semilla in 1:k) {
17       while (TRUE) {
18         fila <- sample(1:n, 1)
19         columna <- sample(1:n, 1)
20         if (zona[fila, columna] == 0) {
21           zona[fila, columna] = semilla
22           x[semilla] <- columna
23           y[semilla] <- fila
24           break
25         }
26       }
27     }
28   celdas <- foreach(p = 1:(n * n), .combine=c) %dopar% celda(p)
29   stopImplicitCluster()
30   voronoi <- matrix(celdas, nrow = n, ncol = n, byrow=TRUE)
31   rotate <- function(x) t(apply(x, 2, rev))
32   Lmin <- n
33   vp <- data.frame(numeric(), numeric())
34   for (dx in -1:1) {
35     for (dy in -1:1) {
36       if (dx != 0 | dy != 0) {
37         vp <- rbind(vp, c(dx, dy))
38       }
39     }
40   }
41   names(vp) <- c("dx", "dy")
42   vc <- dim(vp)[1]
43
44   registerDoParallel(makeCluster(detectCores() - 1))
45   largos <- foreach(r = 1:200, .combine=c) %dopar% propaga(r)
46   stopImplicitCluster()
47   Sum <- c(n, k, summary(largos))
48   Info <- rbind(Info, Sum)
49   Largo <- c(Largo, largos)
50   Nys <- c(Nys, rep(n, 200))
51   Kys <- c(Kys, rep(k, 200))
52   colnames(Info) <- c("n", "k", "Min", "Q1", "Median", "Mean", "Q3", "Max")
53 }
54 }
```

```

55 Nys <- factor(Nys)
56 Kys <- factor(Kys)
57 summary(aov(Largo ~ Nys * Kys))
58 system("convert -delay 50 -size 300x300 p4g_*.png -loop 0 Voronoi.gif")

```

Con la ejecución del código se obtuvo la Teselación de Voronoi que es mostrada en la figura 1(a), en la cual se puede observar la posición inicial de las semillas. Asimismo se hicieron diagramas de Voronoi con la grieta en ellos, se tomó los mas representativos que son mostrados en la figura 1(b), 1(c) y 1(d).

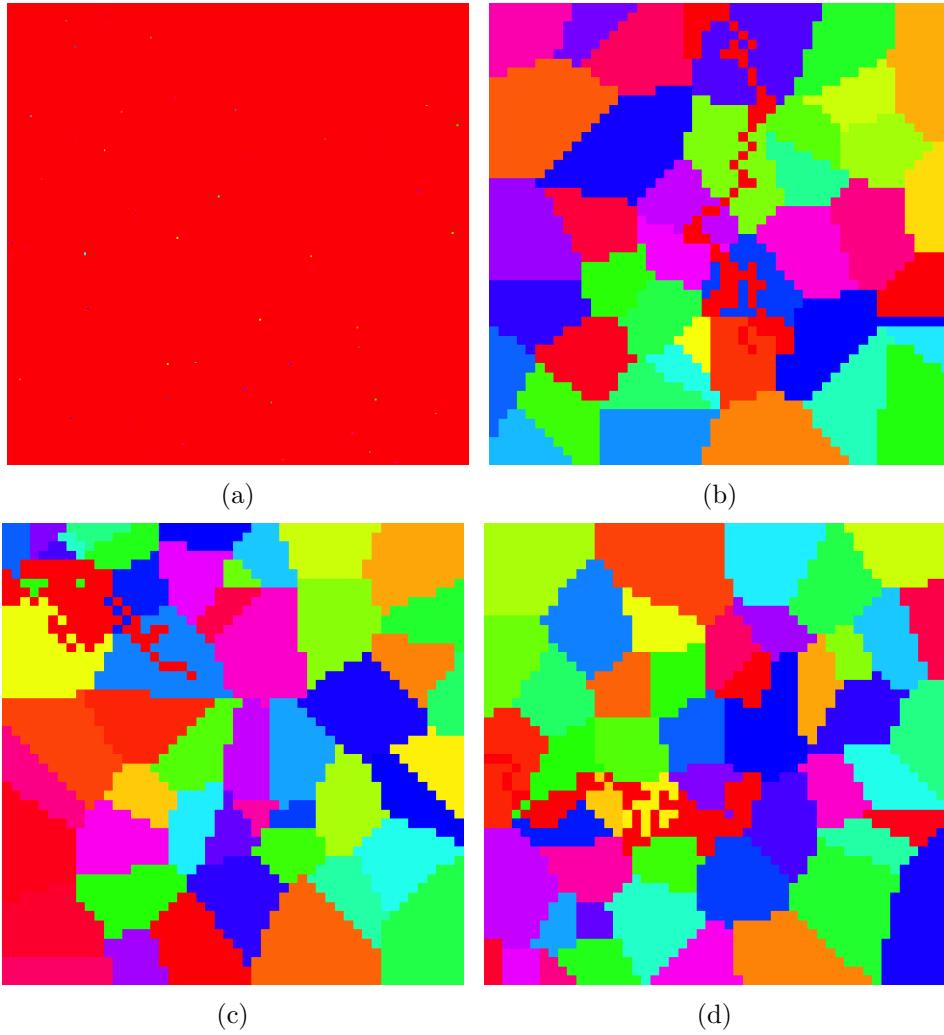


Figura 1: Diagramas de Voronoi del experimento

En la figura 2(b) y 2(a) se aprecia la afinidad de disminución en las zonas con un mayor tamaño. Por lo que la variación de los datos repercute significativamente en el largo de la grieta final. Por consiguiente se puede observar un decremento en la distancia manhattan y grieta al tener una cuadricula de 50 en relación al tamaño de la zona en comparación con las demás cuadriculas que en muestran una afinidad normal entre la grieta y el borde del cuadro.

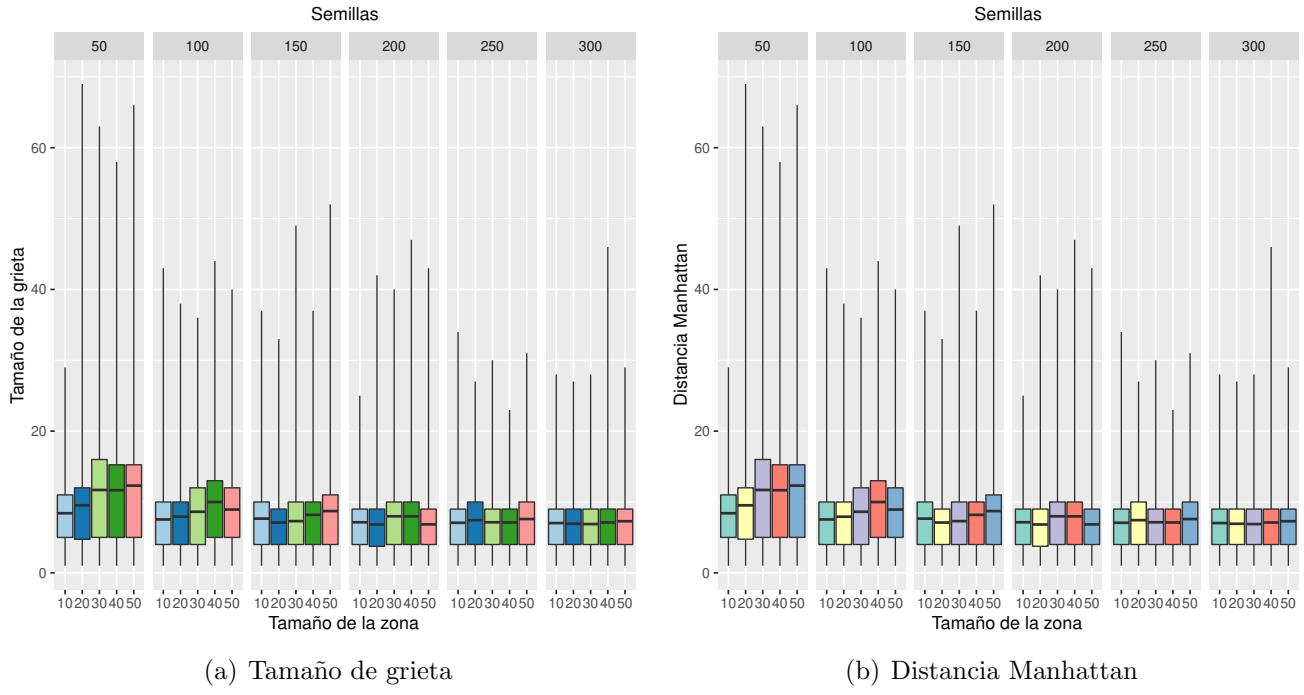


Figura 2: Resultados de ejecución del experimento

Por último, como se mencionó anteriormente, se realizó un análisis ANOVA factorial que permitió determinar la varianza del tamaño de la grieta de acuerdo a su distancia Manhattan y semillas usadas, estos datos son mostrados en el cuadro [1]. En el que se puede observar que se tiene un nivel debajo del 0.05 en $Pr(> F)$, de modo que ocasiona una ligera diferencia en las medias, que muestran una diferencia significativa en el crecimiento de las grietas respecto a la disminución de las semillas.

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

	Gl	Suma Cuad.	Media Cuad.	Valor F	Pr(>F)
Tamaño de zona	5	9580	1916.1	52.523	<2e-16
Semillas	4	1481	370.2	10.149	3.47e-08
Tamaño Zona vs Semillas	20	2155	107.8	2.954	1.06e-05
Residuales	5970	217792	36.5		

Referencias

- [1] Astrid González. Exercise 4, 2018. URL https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_4/.
- [2] Clara Grima. El diagrama de Voronoi, la forma matemática de dividir el mundo, 1980. URL https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101_noticia.html.
- [3] Ricardo Rosas Macías. Práctica 4: diagramas de Voronoi, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP4>.
- [4] Elisa Schaeffer. Práctica 4: diagramas de Voronoi, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p4.html>.

Práctica 5: método Monte-Carlo

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

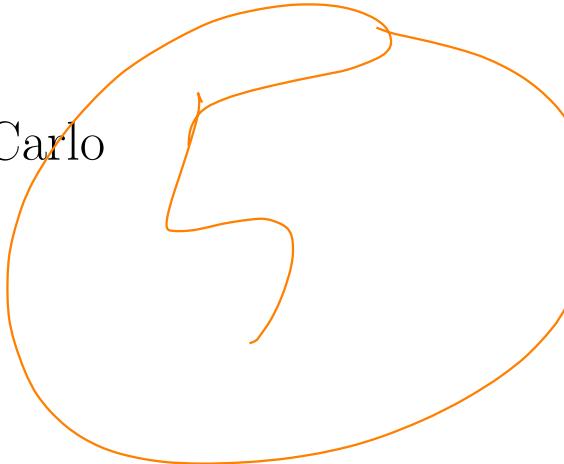
En la quinta práctica tuve algunas dificultades para entender el objetivo principal, por lo que me di a la tarea de buscar información de una fuente externa. En esta, logré hacer el reto por primera vez pero para el segundo reto tuve problemas para obtener los datos del boletín de epidemiología. Por otra parte, se realizaron cambios ligeros para el documento final.

Cabe señalar que no sabía mucho sobre este tema, pero la verdad me pareció muy interesante la aplicación que ha tenido por grandes científicos como Fermi; quien lo uso para muestreos estadísticos de su experimentación. En un futuro seguiré explorando las aplicaciones de este método, debido a que me gustaría indagar más y realizar un código para observar la variación de tamaño en la cinética de la síntesis de nanopartículas; para tener una mejor aproximación del tamaño deseado.

Práctica 5: método Monte-Carlo

Ricardo Rosas Macías

25 de febrero de 2019



1. Introducción

El método Monte Carlo es una herramienta matemática para calcular las probabilidades mediante una serie de números aleatorios, en virtud de ello nos permite identificar cuantos puntos se necesitan para estimar el valor de interés. “Es idóneo para situaciones en las cuales algún valor o alguna distribución no se conoce y resulta complicado de determinar de manera analítica”[3].

2. Objetivo

En términos generales, se realizó cambios en el código de modo que proporcionó el estimado en función de los decimales, para determinar el tamaño de muestra necesario para la aproximación del valor real.

2.1. Descripción

Lo que se debe hacer es [3]:

“Determinar el tamaño de muestra requerido por cada lugar decimal de precisión del estimado obtenido para la integral, comparando con la calculadora virtual Wolfram Alpha[2] de uno hasta siete decimales, asimismo representar el resultado como una sola gráfica. De igual manera para el primer reto, implementar la estimación del valor de π de Kurt[1] con paralelismo, determinar la relación matemática entre el número de muestras obtenidas y la precisión obtenida en términos de la cantidad de lugares decimales correctos”

3. Resultados y conclusiones

En la figura 1 podemos observar la integral $\int_3^7 f(x)dx$ respecto a la función: $f(x) = \frac{1}{\exp(x) + \exp(-x)}$. Asimismo con los datos se realizó una normalización ($g(x) = 2f(x)/\pi$) para obtener la distribución arbitraria 1. El resultado final del código proporcionado arrojó un valor aproximado que posteriormente se comparó con el valor real proporcionado de la calculadora Wolfram[2].

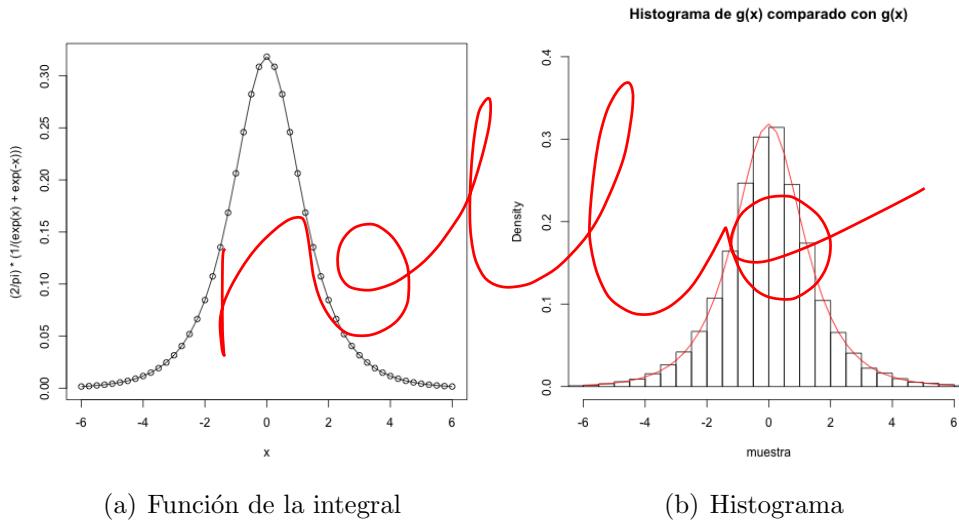


Figura 1: Resultado de muestra pseudoaleatoria

En la figura 2 se observa la aproximación al valor real, en donde la secuencia de la muestra del experimento fue 300 realizando cambios en su tamaño; 50, 100, 500, 1000, 5000 y 10000, con 10 repeticiones para cada una. Esto nos permitió conocer la aproximación al valor real: 0.0488340. De manera que la gráfica nos muestra que al incrementar el tamaño de muestra la precisión incrementa en el resultado final, por lo tanto a 10000 la muestra tendrá mayor exactitud en los decimales del valor real, así como poca variación de los números aleatorios.

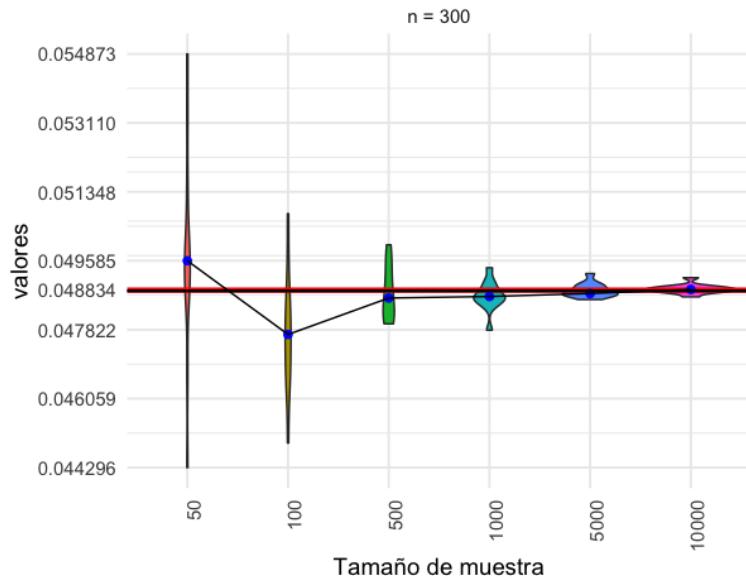


Figura 2: Aproximación al valor real

De acuerdo a la ponderación anterior, para el primer reto se realizó el experimento a 10 repeticiones y con una secuencia de 300; en el cual las condiciones permitieron observar la aproximación de π (3.14159265359), de modo que en la gráfica a de la figura 3 podemos notar que en comparación a los

resultados anteriores hay una diferencia al tener un tamaño de muestra menor, este presenta una mejor aproximación al valor, como es el caso de la muestra 500; la cual denota un ligero cambio porcentual obtenido, así como su tamaño de error es ligeramente desapercibido en contraposición de las demás muestras.

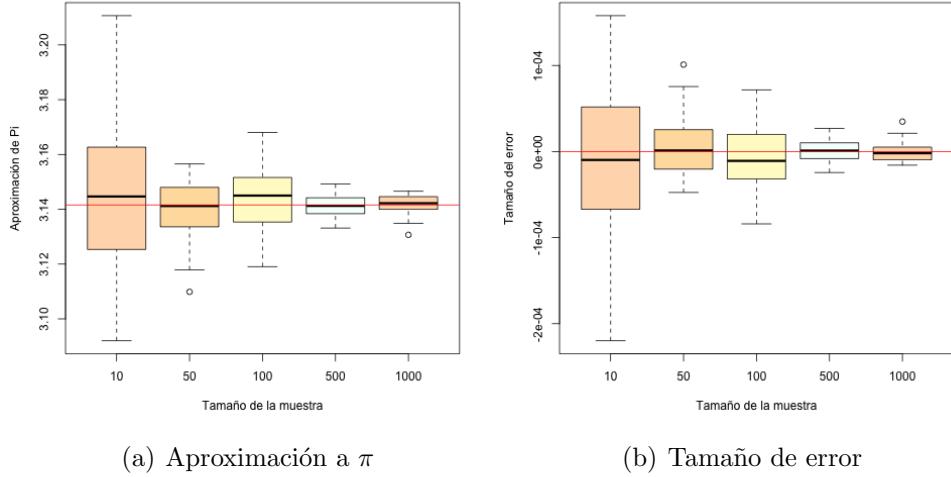


Figura 3: Estimación al valor de π

Referencias

- [1] Will Kurt. 6 neat tricks with monte carlo simulations - count bayesie; probably a probability blog, 2015. URL <https://www.countbayesie.com/blog/2015/3/3/6-amazing-trick-with-monte-carlo-simulations>.
- [2] Wolfram Research. Mathematica, 2019. URL [https://www.wolframalpha.com/input/?i=integrate+\(2%2Fpi\)+**+\(1%2F\(exp\(x\)%2Bexp\(-x\)\)\)+from+-infty+to+infty](https://www.wolframalpha.com/input/?i=integrate+(2%2Fpi)+**+(1%2F(exp(x)%2Bexp(-x)))+from+-infty+to+infty).
- [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

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

El método Monte Carlo es una herramienta matemática para calcular las probabilidades mediante una serie de números aleatorios, en virtud de ello nos permite identificar cuántos puntos se necesitan para estimar el valor de interés.

2. Objetivo

En términos generales, se realizó cambios en el código de modo que proporcionó el estimado en función de los decimales, para determinar el tamaño de muestra necesario para la aproximación del valor real.

2.1. Descripción

Lo que se debe hacer es [3]:

“Determinar el tamaño de muestra requerido por cada lugar decimal de precisión del estimado obtenido para la integral, comparando con la calculadora virtual Wolfram Alpha [2] de uno hasta siete decimales, asimismo representar el resultado como una sola gráfica. De igual manera para el primer reto, implementar la estimación del valor de π de Kurt [1] con paralelismo, determinar la relación matemática entre el número de muestras obtenidas y la precisión obtenida en términos de la cantidad de lugares decimales correctos”

3. Resultados y conclusiones

Para obtener los resultados se tomaron en base a la integral $\int_3^7 f(x)dx$ respecto a la función: $f(x) = \frac{1}{\exp(x) + \exp(-x)}$. Asimismo con los datos se realizó una normalización ($g(x) = 2f(x)/\pi$) para obtener la distribución arbitraria, como se muestra en las siguientes líneas del código.

¹ desde < 3
² hasta < 7

```

3 Pedazos <- 300
4 cuantos <- 100
5 muestra <- c(50, 100, 500, 1000, 5000, 10000)
6 wolfram<-0.048834 # Valor real
7 repeticiones <- 20
8 parte <- function() {
9   val <- generador(pedazo)
10  return(sum(val >= desde & val <= hasta))
11 }
12 Info <- data.frame()
13 for (pedazo in Pedazos) {
14   for (cuantos in muestra){
15     for (vez in 1:repeticiones){
16       suppressMessages(library(doParallel))
17       clust <- makeCluster(2)
18       registerDoParallel(clust)
19       montecarlo <- foreach(i = 1:cuantos , .combine=c) %dopar% parte()
20       stopCluster(clust)
21       integral <- sum(montecarlo) / (cuantos * pedazo)
22       piInt <- (pi / 2) * integral
23       pI <- c(piInt, cuantos, as.integer(pedazo))
24       print(cuantos)
25       Info <- rbind(Info , pI)
26     }
27   }
28 }
29 colnames(Info) <- c("val" , "rep" , "esp")
30 Info$esp <- as.integer(Info$esp)
31 Info$rep <- as.factor(Info$rep)
32 print(Info)
33
34 agregando <- function(Espacio){
35   Espacio <- as.factor(Espacio)
36   return(paste("n = ", Espacio , sep = ""))
37 }
38 valmin <- (wolfram-50)

```

El resultado final del código proporcionado muestra un valor aproximado que posteriormente se comparó con el valor real proporcionado de la calculadora Wolfram [2]. En la figura 1 se observa la aproximación al valor real, en donde la secuencia de la muestra del experimento fue 300 realizando cambios en su tamaño; 50, 100, 500, 1000, 5000 y 10000, con 10 repeticiones para cada una. Esto nos permitió conocer la aproximación al valor real: 0.0488340. De manera que la gráfica nos muestra que al incrementar el tamaño de muestra la precisión incrementa en el resultado final, por lo tanto a 10000 la muestra tendrá mayor exactitud en los decimales del valor real, así como poca variación de los números aleatorios.

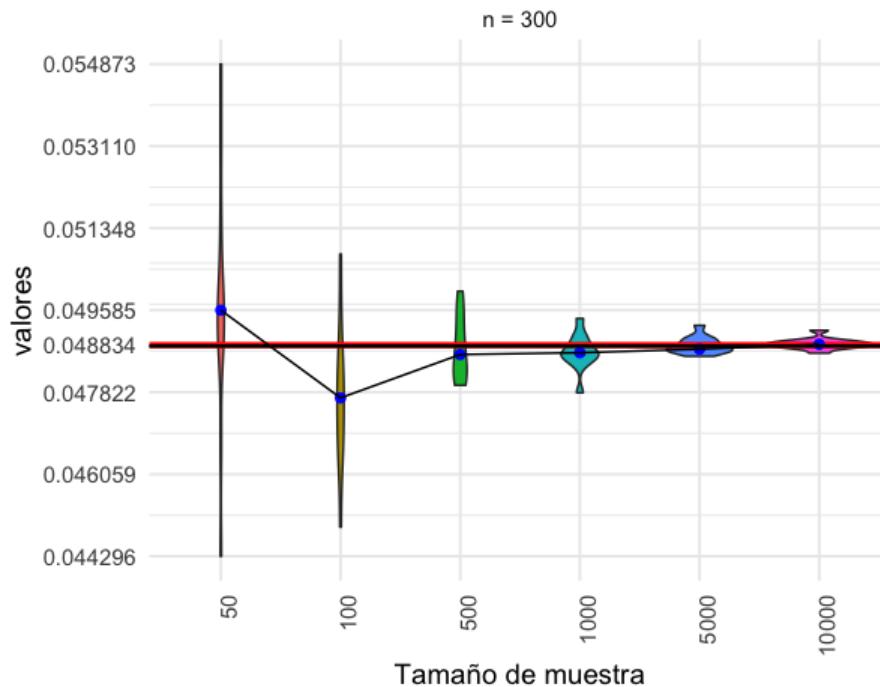


Figura 1: Aproximación al valor real

3.1. Reto 1

De acuerdo a la ponderación anterior, para el primer reto se realizó el experimento a 10 repeticiones y con una secuencia de 300; en el cual las condiciones permitieron observar la aproximación de π (3.14159265359), como se muestra en las siguientes líneas de código.

```

1 inicio <- -0.1
2 final <- -inicio
3 pi <- 3.14159265359
4 muestra <- c(10,50,100,500,1000)
5 repeticiones <- 30
6
7 calcpi=function() {
8   xs <- runif(replicas , min= inicio , max= final)
9   ys <- runif(replicas , min= inicio , max= final)
10  in.circle <- xs^2 + ys^2 <= inicio ^2
11  mc.pi <- (sum(in.circle)/replicas)*4
12  return(mc.pi)
13 }
14 suppressMessages(library(doParallel))
15 registerDoParallel(makeCluster(detectCores() - 1))
16
17 resultados=data.frame()
18
19 for(replicas in muestra) {
20   for(i in 1:repeticiones) {
21     montecarlo <- foreach(i = 1:300, .combine=c) %dopar% calcpi()
22     mc.pi=sum(montecarlo)/300

```

```

23 diferencia=(pi-mc.pi)/(pi*100)
24 pii <- pi
25 resultados=rbind(resultados,c(replicas,i,pii,mc.pi,diferencia))
26 }
27 }
28 names(resultados)=c("muestra","replicas","Valor de Pi","aprox.pi","error")

```

De modo que en la gráfica a de la figura 2 podemos notar que en comparación a los resultados anteriores hay una diferencia al tener un tamaño de muestra menor, este presenta una mejor aproximación al valor, como es el caso de la muestra 500; la cual denota un ligero cambio porcentual obtenido, así como su tamaño de error es ligeramente desapercibido en contraposición de las demás muestras.

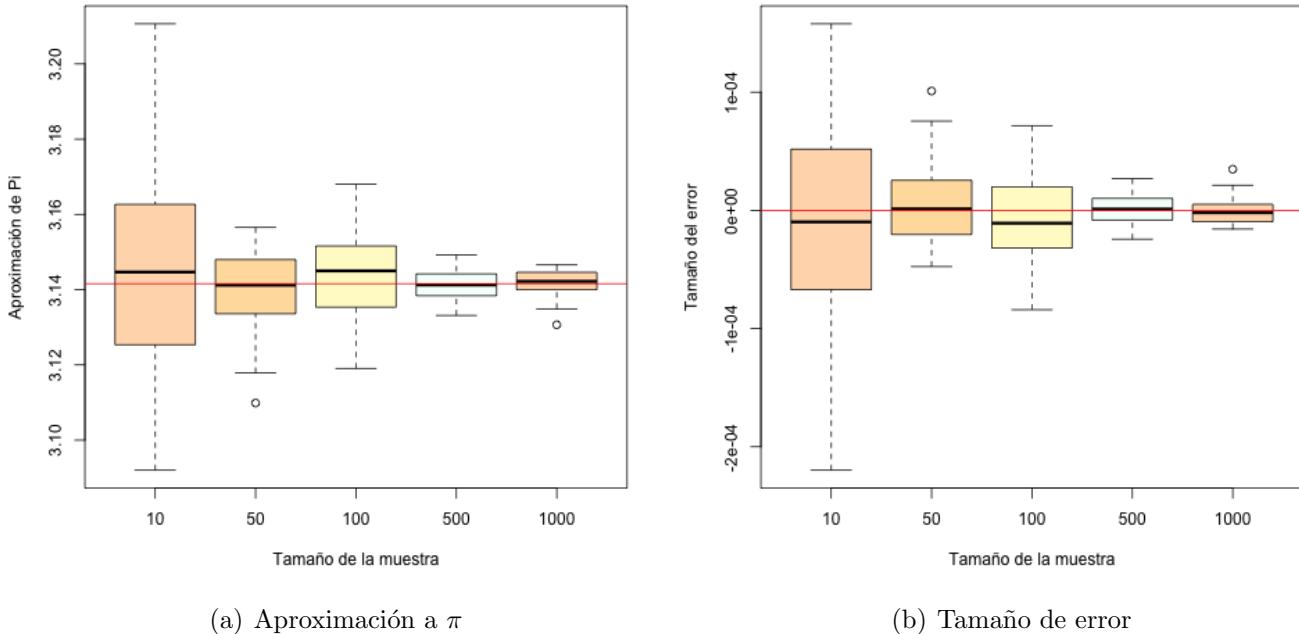


Figura 2: Estimación al valor de π

Referencias

- [1] Will Kurt. 6 neat tricks with Monte Carlo simulations - count Bayesie; probably a probability blog, 2015. URL <https://www.countbayesie.com/blog/2015/3/3/6-amazing-trick-with-monte-carlo-simulations>.
- [2] Wolfram Research. Mathematica, 2019. URL [https://www.wolframalpha.com/input/?i=integrate+\(2%2Fpi\)+**+\(1%2F\(exp\(x\)%2Bexp\(-x\)\)\)+from+-infty+to+infty](https://www.wolframalpha.com/input/?i=integrate+(2%2Fpi)+**+(1%2F(exp(x)%2Bexp(-x)))+from+-infty+to+infty).
- [3] Elisa Schaeffer. Práctica 5: método Monte-Carlo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p5.html>.

Práctica 6: sistema multiagente

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la sexta práctica, se logra apreciar una mejora en las líneas del código; se muestra un error que repercutió en un par de valores, que posteriormente se logró cambiar para el documento final. En adición, en esta práctica tuve buena comprensión del tema, aunque no logre terminar los retos debido a falta de lógica matemática de mi parte. Por lo que en un futuro seguiré leyendo y practicando para el desarrollo de razonamiento matemático.

3194



Práctica 6: sistema multiagente

Ricardo Rosas Macías

4 de marzo de 2019

1. Introducción

Un sistema multiagente es un método en donde los agentes interactúan los unos con los otros con ayuda de la inteligencia artificial, de tal modo que coordinan su comportamiento para llevar acabo una tarea de forma automática o resolver un problema complejo rápidamente.

2. Objetivo

En términos generales, se realizó cambios en el código de modo que permite vacunar a los agentes desde el inicio del experimento. Asimismo se cuantificó el porcentaje máximo de infectados en la ejecución del código.

2.1. Descripción

Lo que se debe hacer es [1]:

“Vacunar con probabilidad p_v a los agentes al momento de crearlos de tal forma que están desde el inicio en el estado R y ya no podrán contagiarse ni propagar la infección. Estudia el efecto estadístico del valor de p_v en (de cero a uno en pasos de 0.1) el porcentaje máximo de infectados durante la simulación.”

\$P-V\$

3. Resultados y conclusiones

Las ponderaciones para el experimento de 50 agentes y 5 repeticiones. Se realizaron cambios

Para realizar acabo el experimento se tomaron los valores predeterminados del código del sitio web [1].

```
1 l <- 1.5
2 n <- 50
3 pi <- 0.05
4 pr <- 0.02
5 v <- 1 / 30
```

6 | r \leftarrow 0.1

\$L \times L \$

Asimismo se tomo las ponderaciones para la variación del experimento de 50 agentes en 5 repeticiones para la ejecución. En el cual los agentes se posicionaron de manera aleatoria en un rectángulo de medidas $L \times L$ en el cual todos sus extremos crean un continuo; como si fuese un toroide. Adicionalmente se programo para que el experimento se ejecutara con los agentes vacunados, como se muestra en el inicio del código.

```
1 for (pv in pV) {  
2   for (rep in 1:replicas) {  
3     agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(),  
4       estado = character(), amigo = NULL)  
5     for (i in 1:n) {  
6       if (runif(1) < pv){  
7         e <- "R"  
8       } else if (runif(1) < pi){  
9         e <- "I"  
10      } else{  
11        e <- "S"  
12      }
```

Por otra parte se ejecutó este código para obtener el efecto estadístico, de modo que se obtengan los datos, para posteriormente ser gráficados.

```
1 vac <- data.frame()  
2 mxinf <- max(epidemia)  
3 porcentaje <- 100 * mxinf / n  
4 vac <- rbind(vac, c(pv, rep, mxinf, porcentaje))  
5 print(pv)
```

En la figura 1 podemos apreciar que en cuanto más probabilidad de vacunados de cero a uno en pasos de 0.1 el experimento tendrá un mejor control en el máximo de infección de los agentes. Además se puede apreciar una distribución uniforme y decreciente en los datos de la epidemia, como se muestra en la gráfica.

Referencias

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

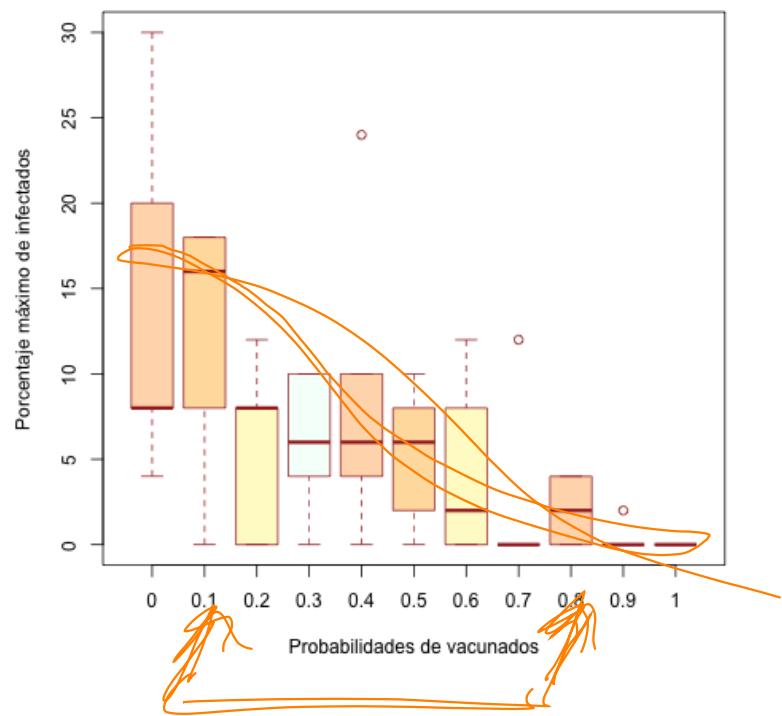


Figura 1: Efecto estadístico

Avergiar: pas de
altz edibetak

Práctica 6: Sistema multiagente

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

Un sistema multiagente es un método en donde los agentes interactúan los unos con los otros con ayuda de la inteligencia artificial, de tal modo que coordinan su comportamiento para llevar acabo una tarea de forma automática o resolver un problema complejo rápidamente.

2. Objetivo

En términos generales, se realizó cambios en el código de modo que permite vacunar a los agentes desde el inicio del experimento. Asimismo se cuantificó el porcentaje máximo de infectados en la ejecución del código.

2.1. Descripción

Lo que se debe hacer es [2]:

“Vacunar con probabilidad p_v a los agentes al momento de crearlos de tal forma que están desde el inicio en el estado R y ya no podrán contagiarse ni propagar la infección. Estudia el efecto estadístico del valor de p_v en (de cero a uno en pasos de 0.1) el porcentaje máximo de infectados durante la simulación.”

3. Resultados y conclusiones

Para realizar acabo el experimento se tomaron los valores predeterminados del código del sitio web [2], como se muestra en la lista siguiente.

```
1 l <- 1.5
2 n <- 50
3 pi <- 0.05
4 pr <- 0.02
5 v <- 1 / 30
```

```

6 r <- 0.1
7 tmax <- 100

```

Con ayuda de la paquetería *listings* se colocaron las líneas de código modificada █. Asimismo se tomo las ponderaciones para la variación del experimento de 50 agentes en 30 repeticiones para la ejecución. En el cual los agentes se posicionaron de manera aleatoria en un rectángulo de medidas $L \times L$ en el cual todos sus extremos crean un continuo; como si fuese un toroide. Adicionalmente se programo para que el experimento se ejecutara con los agentes vacunados, como se muestra en el inicio del código

```

1 pV <- c(0, 1, 0.1)
2 IMax <- c()
3 for(pv in pV) {
4   for(rep in 1:30) {
5     agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(),
6       estado = character())
7     for (i in 1:n) {
8       e <- "S"
9       if(runif(1) < pv) {
10         e <- "R"
11       } else {
12         if (runif(1) < pi) {
13           e <- "I"
14         }
15       }
16       levels(agentes$estado) <- c("S", "I", "R")
17       agentes <- rbind(agentes, data.frame(x = runif(1, 0, 1), y = runif(1, 0, 1), dx =
18         runif(1, -v, v),
19         dy = runif(1, -v, v), estado = e))
20     }
21   }
22 }
23 Vacu <- data.frame(Probvac = seq(pV, 30), Imaxi = (IMax/n)*100)

```

Por otra parte se ejecutó este código para obtener el efecto estadístico, de modo que se obtengan los datos, para posteriormente ser gráficados.

```

1 IMax <- c(IMax, max(epidemia))
2 }
3 }
4 Vacu <- data.frame(Probvac = seq(pV, 30), Imaxi = (IMax/n)*100)

```

La ejecución del experimento mostró los resultados del análisis estadístico, que fue realizado mediante la prueba *Kruskal-Wallis*, el cual evidencia que los datos tienen diferencias entre medianas estadísticamente significativas; debido a nivel de significación en donde el valor de p es menor, como se muestra en el cuadro █.

Cuadro 1: Resultados de prueba Kruskal Wallis

Chi-squared	df	p-value
62.453	10	1.241e-09

Por último, en la figura [1] podemos apreciar que en cuanto más probabilidad de vacunados de 0 – 1 en pasos de 0.1 el experimento tendrá un mejor control en el máximo de infección de los agentes. Además se puede apreciar una distribución uniforme y decreciente en los datos de la epidemia, como se muestra en la figura [1].

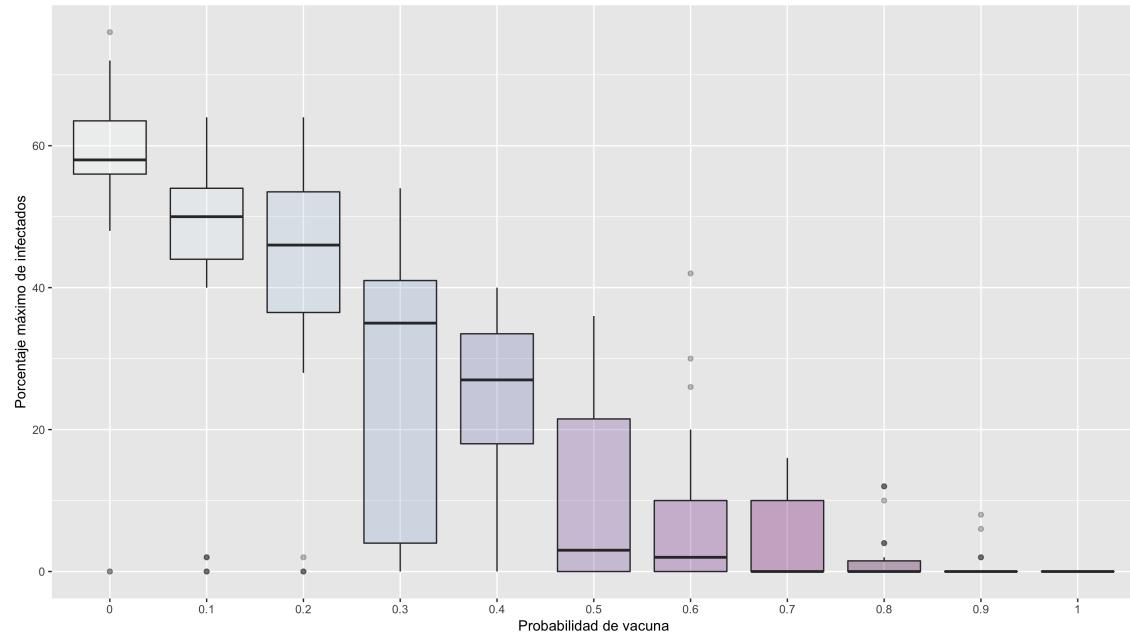


Figura 1: Efecto estadístico

Referencias

- [1] Ricardo Rosas Macías. Práctica 6: Sistema multiagente, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP6>.
- [2] Elisa Schaeffer. Práctica 6: Sistema multiagente, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.

Práctica 7: búsqueda local

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la séptima práctica, tuve menores errores en comparación de las prácticas anteriores; esto demuestra que hay un avance favorecedor en la curva de aprendizaje. En el documento final se puede observar que se mejoró el reporte de resultados, asimismo se logró realizar el primer reto, debido a que gracias a la literatura existente logré comprender y darme una idea para realizar las líneas de código que cumplieran con el objetivo planteado.

Por otra parte, para la comprensión de la práctica leí los recursos bibliográficos que muestra la página web, pero al final recurrió a una búsqueda literaria en *Google Scholar*; de manera que me proporcionara más información para abordar el tema.

Práctica 7: búsqueda local

Ricardo Rosas Macías

18 de marzo de 2019

3199

1. Introducción

La búsqueda local usa el método heurístico, que le permite encontrar la mejor solución de todas las rutas de resultado posibles; a través de descubrir la posición exacta del agente le deja obtenerla en un tiempo corto.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web[2], de manera que el agente en el experimento mantiene un movimiento 3D pero con una visualización en 2D que permite observar la posición óptima en la secuencia del experimento.

2.1. Descripción

Lo que se debe hacer es [2]:

“La finalidad del experimento es maximizar la función bidimensional del ejemplo $g(x, y)$, con restricciones $-3 \leq x, y \leq 3$, con la misma técnica del ejemplo unidimensional. La posición actual es un par x y y se ocupan dos movimientos aleatorios, Δx y Δy , cuyas combinaciones posibles proveen ocho posiciones vecino, de los cuales aquella que logra el mayor valor para g es seleccionado.”

3. Resultados y conclusiones

En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajaría con la función g .

```
1 resultados<-data.frame()
2 g <- function(x, y) {
3   return(((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y^2 - 20 * y)/100)
4 }
```

```

5
6 low <- -3
7 high <- 3
8 step <- 0.01
9 replicas <- 15

```

Posteriormente se generó el código que de manera tuviera movimientos de izquierda o derecha, así como con un patrón de arriba o abajo, como se muestra en las líneas de código. Asimismo, se realizó una combinación de dichos movimientos de modo de recreación de un eje (x, z) para el movimiento del agente dentro de la zona creada $-3 \leq x, y \leq 3$.

```

1 replica <- function(t) {
2   curr <- runif(1, min = low, max = high)
3   actual <- runif(1, min = low, max = high)
4   best <- curr
5   firstbest <- actual
6   for (tiempo in 1:t) {
7     delta <- runif(1, 0, step)
8     omega <- runif(1, 0, step)
9     left <- curr - delta
10    right <- curr + delta
11    up <- actual + omega
12    down <- actual - omega

```

cur, up, down, best, firstbest, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

Optimización

En las líneas finales se puede ver que esta indicado una proximidad máxima local para los agentes, lo cual permitirá que seleccione el mejor de los vecinos y así este converja en un punto óptimo.

```

1 for (pot in 2:4) {
2   tmax <- 10^pot
3   resultados <- foreach(i = 1:replicas, .combine="rbind") %dopar %replica(tmax)
4   valores <- outer(resultados[,1], resultados[,2], g)
5   mejor <- which.max(valores)
6   dimnames(valores) <- list(resultados[,1], resultados[,2])
7   funcion <- melt(valores)
8   names(funcion) <- c("x", "y", "z")
9   zona <- levelplot(z ~ x * y, data = funcion)

```

Se obtuvieron representaciones gráficas con la ayuda de la paquetería lattice[1] de las 15 replicas simultáneas, se seleccionó las mejores representaciones; como se muestra en la figura1, en donde se puede observar que las posiciones iniciales son proporcionadas de manera aleatoria, de modo que con las repeticiones y con un incremento de 10000 en los pasos el resultado se refino hasta obtener la posición óptima del agente, como se puede observar en la repetición 5.

text

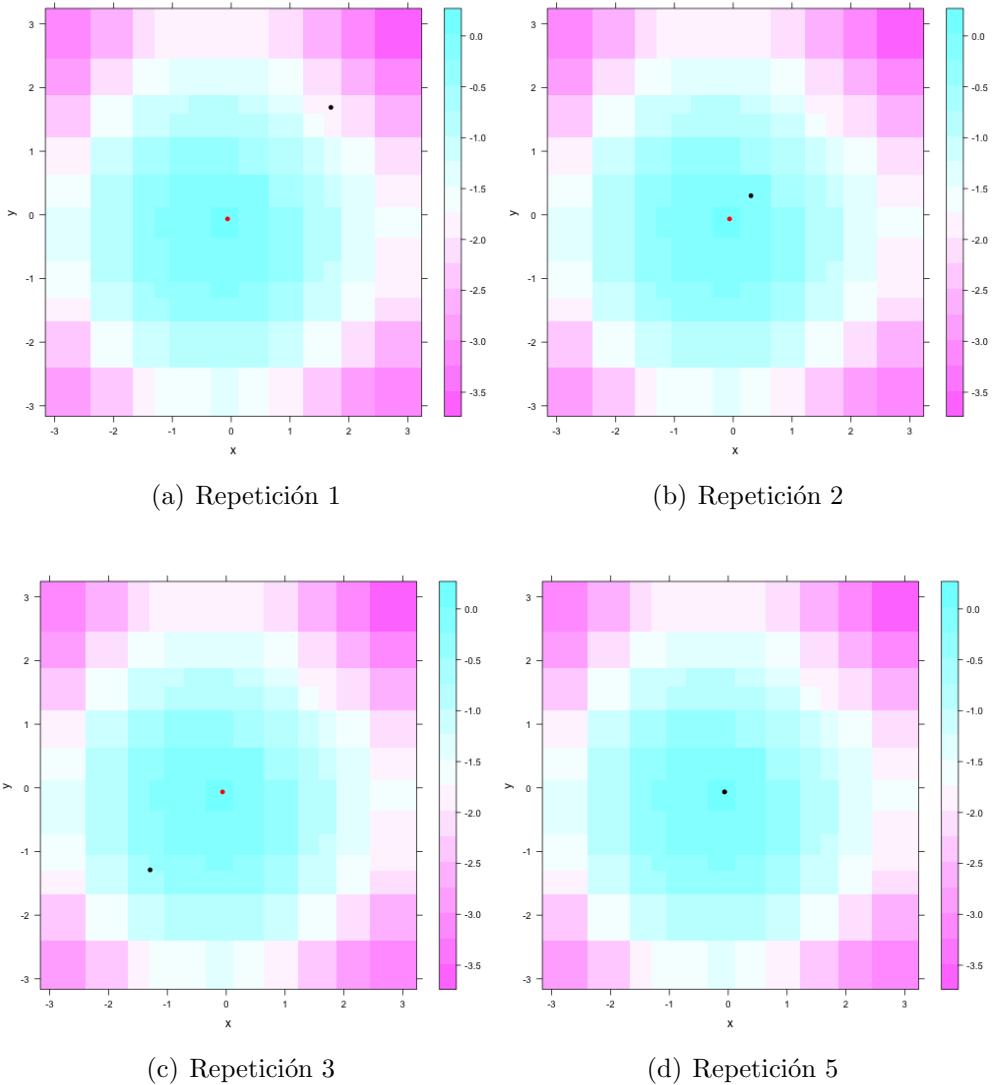


Figura 1: Proyección plana del experimento a 10000 pasos

Referencias

- [1] Deepayan Sarkar. Lattice: trellis graphics for r, 2011. URL <http://lattice.r-forge.r-project.org>.
- [2] Elisa Schaeffer. Práctica 7: búsqueda local, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p7.html>.

Práctica 7: búsqueda local

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

La búsqueda local usa el método heurístico, que le permite encontrar la mejor solución de todas las rutas de resultado posibles; a través de descubrir la posición exacta del agente que le deja obtenerla en un tiempo corto.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [4][1], de manera que el agente en el experimento mantiene un movimiento 3D pero con una visualización en 2D que permite observar la posición óptima en la secuencia del experimento.

2.1. Descripción

Lo que se debe hacer es [4]:

“La finalidad del experimento es maximizar la función bidimensional del ejemplo $g(x, y)$, con restricciones $-3 \leq x, y \leq 3$, con la misma técnica del ejemplo unidimensional. La posición actual es un par x y y se ocupan dos movimientos aleatorios, Δx y Δy , cuyas combinaciones posibles proveen ocho posiciones vecino, de los cuales aquella que logra el mayor valor para g es seleccionado.

El primer reto es cambiar la regla del movimiento de una solución x (un vector de dimensión arbitraria) a la siguiente a la de recocido simulado: para optimizar una función $f(x)$, se genera para la solución actual x un sólo vecino $x' = x + \Delta x$ (algún desplazamiento local). Se calcula $\delta = f(x') - f(x)$ (para minimizar; maximizando la resta se hace al revés). Si $\delta > 0$, siempre se acepta al vecino x' como la solución actual ya que representa una mejora. Si $\delta < 0$, se acepta a x' con probabilidad $\exp(-\frac{\delta}{T})$ y rechaza en otro caso. Aquí T es una temperatura que decrece en aquellos pasos donde se acepta una empeora; la reducción se logra multiplicando el valor actual de T con $\xi < 1$, como por ejemplo 0.995. Examina los efectos estadísticos del valor inicial de T y el valor de ξ en la calidad de la solución, es decir, qué tan bajo (para minimizar; alto para maximizar) el mejor valor termina siendo.”

3. Resultados y conclusiones

En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajaría con la función g .

```
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 low <- -3
5 high <- -low
6 step <- 0.25
7 replicas <- 15
```

Posteriormente se generó el código que de manera tuviera movimientos de izquierda o derecha, así como con un patrón de arriba o abajo, como se muestra en las líneas de código. Asimismo, se realizó una combinación de dichos movimientos de modo de recreación de un eje z para el movimiento del agente dentro de la zona creada de $-3 \leq x, y \leq 3$.

```
1 replica <- function(t){
2   puntosxy<- c()
3   curr <- c( x = runif(1, min = low , max = high) , y = runif(1, min = low , max = high))
4   best <- curr
5   for (tiempo in 1:t) {
6     delta <- runif(1, 0, step)
7     omega <- runif(1, 0, step)
8     left <- curr + c(-delta,0) # Eje izquierdo
9     right <- curr + c(delta,0) # Eje Derecho
10    up <- curr + c(0,-delta) # Eje arriba
11    down <- curr + c(0,delta) # Eje abajo
12    puntos <- c(left , right , up , down)
13
14   for(k in 1:8){
15     if(puntos[k] < (-3)){
16       puntos[k] <- puntos[k]+3
17     }
18     if(puntos[k] > 3){
19       puntos[k] <- puntos[k]-3
20     }
21   }
22   vecx <- c()
23   vecy <- c()
24   for(p in 1:8){
25     if(p %%2 == 0){
26       vecy <- c(vecy , puntos[p])
27     } else{
28       vecx <- c(vecx , puntos[p])
29     }
30   }
31   valg <- c()
32   for(q in 1:4){
33     valg <- c(valg , g(vecx[q] , vecy[q]) )
34   }
35   dm <- which.max(valg)
```

```

36     curr <- c(vecx[dm], vecy[dm])
37     puntosxy <- c(puntosxy, vecx[dm], vecy[dm])
38   }
39   return(puntosxy)
40 }
41
42 resultado <- c()
43 for(q in 1:5){
44   resultado <- c(resultado, replica(100))
45 }
46
47 vx <- c()
48 vy <- c()
49 for(p in 1:1000){
50   if(p %% 2 == 0){
51     vy <- c(vy, resultado[p])
52   } else{
53     vx <- c(vx, resultado[p])
54   }
55 }
```

En las líneas finales se puede ver que esta indicado una proximidad máxima local para los agentes, lo cual permitirá que seleccione el mejor de los vecinos y así este converja en un punto óptimo.

```

1 for (pot in 2:4) {
2   tmax <- 10^pot
3   resultados <- foreach(i = 1:replicas, .combine="rbind") %dopar% replica(tmax)
4   valores <- outer(resultados[,1], resultados[,2], g)
5   mejor <- which.max(valores)
6   dimnames(valores)<-list(resultados[,1], resultados[,2])
7   funcion <- melt(valores)
8   names(funcion) <- c("x", "y", "z")
9   zona <- levelplot(z ~ x * y, data = funcion)
```

Se obtuvieron representaciones gráficas con la ayuda de la paquetería *lattice* [3] de las 15 réplicas simultáneas, se seleccionó las mejores representaciones; como se muestra en la figura 1, en donde se puede observar que las posiciones iniciales son proporcionadas de manera aleatoria, de modo que con las repeticiones y con un incremento de 10000 en los pasos el resultado se refinó hasta obtener la posición optima del agente, como se puede observar en la figura 1(d).

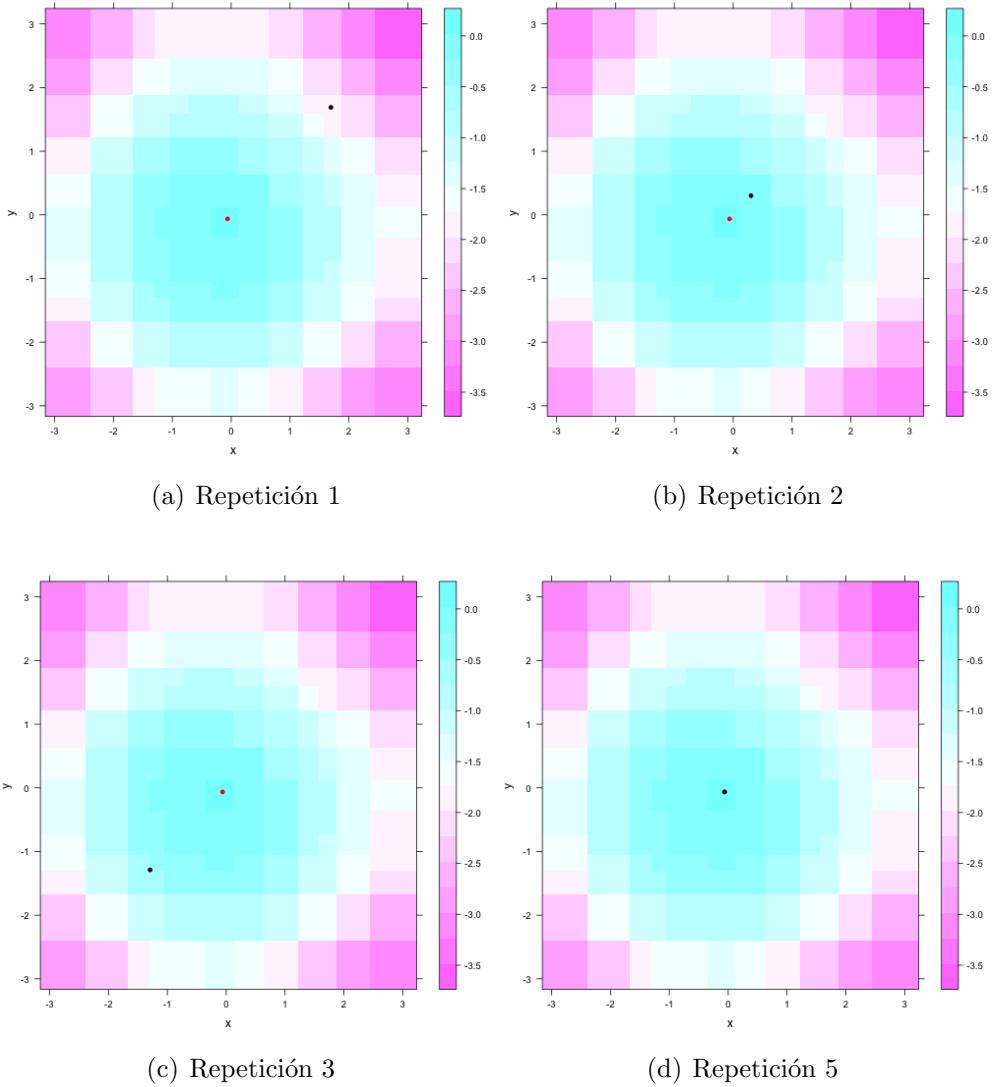


Figura 1: Proyección plana del experimento a 10000 pasos

3.1. Reto1

Para realizar el reto 1, se tomo como ejemplo lo anteriormente reportado [2], como se muestra en las líneas siguientes.

```

1 g <- function(x, y) {
2   a<- (((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y^2 - 20 * y)/100)
3   return(a)
4 }
5 valeng <- c()
6 temperaturas <- c(5,10,15,20,40)
7 for (tem in temperaturas){
8   low <- -3
9   high <- -low
10  step <- 0.25

```

```

11 replicas <- 15
12 t <- tem
13 ep <- 0.80
14 replica <- function(t){
15   curr <- c(runif(1, low, high), runif(1, low, high))
16   best <- curr
17   for (tiempo in 1:t) {
18     delta <- runif(1, 0, step)
19     x1 <- curr + c(-delta,0)
20     x2 <- curr + c(delta ,0)
21     y1 <- curr + c(0,-delta)
22     y2 <- curr + c(0,delta)
23     puntos <- c(x1,x2,y1,y2)
24     for(k in 1:8){
25       if(puntos[k] < (-5)){
26         puntos[k] <- puntos[k]+10
27       }
28       if(puntos[k] > 5){
29         puntos[k] <- puntos[k]-10
30       }
31     }
32     vecx <- c()
33     vecy <- c()
34     for(p in 1:8){
35       if(p %%2 == 0){
36         vecy <- c(vecy ,puntos[p])
37       }else{
38         vecx <- c(vecx ,puntos[p])
39       }
40     }
41     u <- sample(1:4,1)
42     x.p <- c(vecx[u],vecy[u])
43     delt <- g(x.p[1],x.p[2]) - g(curr[1],curr[2])
44     if(delt > 0){
45       curr <- x.p
46     }else{
47       if(runif(1)< exp((delt) / (t * ep))){
48         curr <- x.p
49         if(t == 1){
50           t <-t
51         }else{
52           t <- t-1
53         }
54       }
55     }
56     if(g(curr[1],curr[2]) > g(best[1],best[2])){
57       best <- curr
58     }
59   }
60   return(best)
61 }
62 tmax <- 100
63 resultados <- c()
64 for(indi in 1:100){
65   resultados <- c(resultados , replica(tmax))

```

```

66 }
67 vecx <- c()
68 vecy <- c()
69 aux <- 200
70 for(p in 1:aux){
71   if(p %%2 == 0){
72     vecy <- c(vecy , resultados [p])
73   } else{
74     vecx <- c(vecx , resultados [p])
75   }
76 }
77 valores <- c()
78 for(q in 1:100){
79   valores <- c(valores , g(vecx [q] , vecy [q]) )
80 }
81 valeng <- c(valeng , valores)
82 }

```

En la figura 2 se puede observar el valor del recocido a diferentes temperaturas a un valor de ξ de 0.1, en donde se puede determinar que la temperatura no altera el resultado final, debido a que hay una diferencia despreciable como se muestra en las cajas bigote.

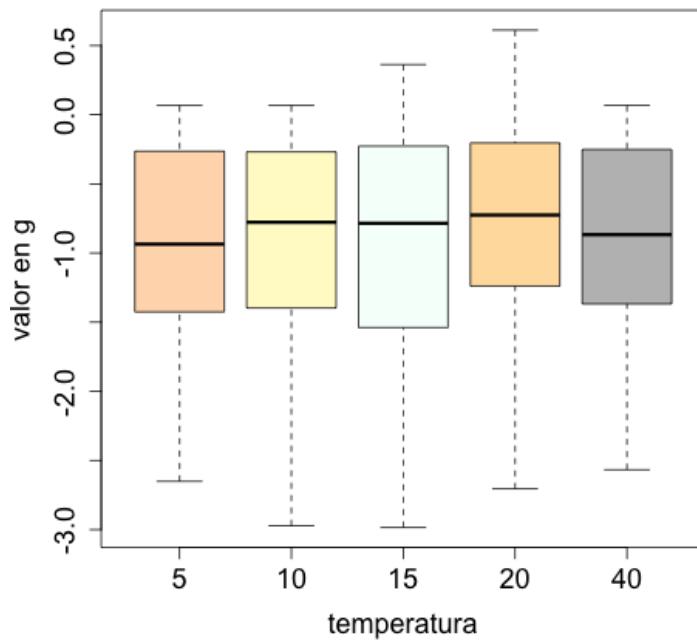


Figura 2: Resultados de recocido simulado

Referencias

- [1] Ricardo Rosas Macías. Práctica 7: búsqueda local, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP7>.
- [2] Ricardo Parga. Práctica 7: búsqueda local, 2018. URL <https://github.com/RParga/R-simulation/tree/master/p7>.
- [3] Deepayan Sarkar. Lattice: trellis graphics for r, 2011. URL <http://lattice.r-forge.r-project.org>.
- [4] Elisa Schaeffer. Práctica 7: búsqueda local, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p7.html>.

Práctica 8: modelo de urnas

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

La octava práctica, comprendí con mayor facilidad ya que este es un fenómeno que sucede muchas veces en las nanopartículas; estas se logran juntarse con ayuda de un coagulante para realizar procesos de filtración o recuperación de sustancias líquidas o nanopartículas precipitadas. En el primer documento entregado tuve problemas para plasmar las ideas en el código, por lo cual en el documento final se puede observar una buena práctica del uso de conocimiento adquirido; en su mayor parte para mostrar una mejor visualización de los resultados obtenidos.



Práctica 8: modelo de urnas

Ricardo Rosas Macías

25 de marzo de 2019

3194

1. Introducción

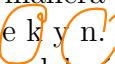
El modelo de urnas arriba los fenómenos coalescencia y fragmentación, de modo en que une las partículas que se encuentran divididas en dos o más pedazos más pequeños; siempre siendo un entero. Este experimento tiene una reacción parecida a los agentes coagulantes en donde las partículas más pequeñas se agrupan para formar una más grande y así poder ser separada de una manera más fácil.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [1], de manera que al paralelizar se vuelva más eficiente realizando varias tareas al mismo tiempo y así reducir el tiempo total de la ejecución del código, asimismo se busca visualizar el cambio al generar repeticiones del experimento.

2.1. Descripción

Lo que se debe hacer es [1]:

“La finalidad del experimento es paralelizar tanto como resulte eficientemente posible en esta simulación y medir cuánto tiempo se logra ahorrar, de igual manera evaluar si el ahorro es estadísticamente significativo para diferentes combinaciones de k y n .


Para el primer reto, se determinó en cuál momento (en términos del número de iteraciones de la simulación) la fracción de éstas alcance un máximo. Calcula y gráfica esta fracción a lo largo de la simulación. Realiza 30 réplicas para determinar si el número de paso en el cual se alcanza el máximo tiene un comportamiento sistemático.”

3. Resultados y conclusiones

En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajaría, asimismo se ejecuto el comando de paralelizar para que el experimento fuera más eficiente. Además

se realizó una comparación de la ejecución del experimento en donde se ponderaron diferentes valores para las partículas que están representadas como "k" para los cúmulos definidos con la variante "n"

```

1 library(testit) # para pruebas, recuerda instalar antes de usar
2 library(parallel)
3 tiempo1<-data.frame()
4
5 k <- c(100, 1000, 10000)
6 n <- c(10000, 100000, 1000000)
7
8 cluster <- makeCluster(detectCores() - 1)
9
10 rotura <- function(x) {
11   return(1 / (1 + exp((c - x) / d)))
12 }

```

Como se menciona en el apartado anterior, para observar cambios significativos en el tiempo del experimento se varió los valores de las partículas (k) y cúmulos (n) respectivamente, de manera que los datos proporcionados se muestran en la gráfica 1, en donde se puede observar claramente que los tiempos al paralelizar son mejores a comparación de la ejecución normal. Por lo tanto es un factor determinante para optar por el uso de dicha función.

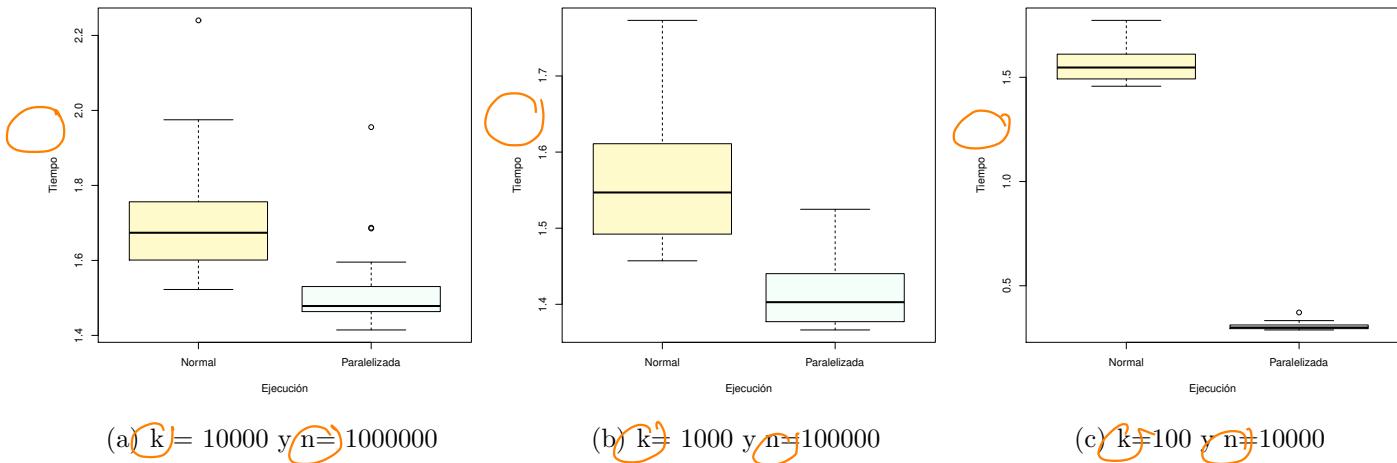


Figura 1: Tiempo del experimento

Para el primer reto se busca que el experimento se pueda replicar por 30 veces como se determinó en las siguientes líneas del código. Asimismo se generó la variante ~~x~~^{k1} para minimizar los valores en los cúmulos y de esta manera determinar si este tiene un comportamiento sistemático.

```

1 k <- 10000
2 n <- 1000000
3 k1 <- c(500, 1000, 10000, 100000)
4 replicas = 30
5 tiempos = data.frame(replica=numeric(), k=numeric(), time=numeric())

```

```

1 cluster <- makeCluster(detectCores() - 1)
2
3 for(k in k1){
4   n <- 30*k
5   digitos <- floor(log(duracion, 10)) + 1
6
7   for(replica in 1:replicas){
8     tempo = proc.time()[3]
9     originales <- rnorm(k)

```

De tal modo que en la gráfica 2 se muestra la variación del máximo de las repeticiones de la ejecución, de igual manera en la figura 3 se puede observar en como se tiene un comportamiento sistemático.

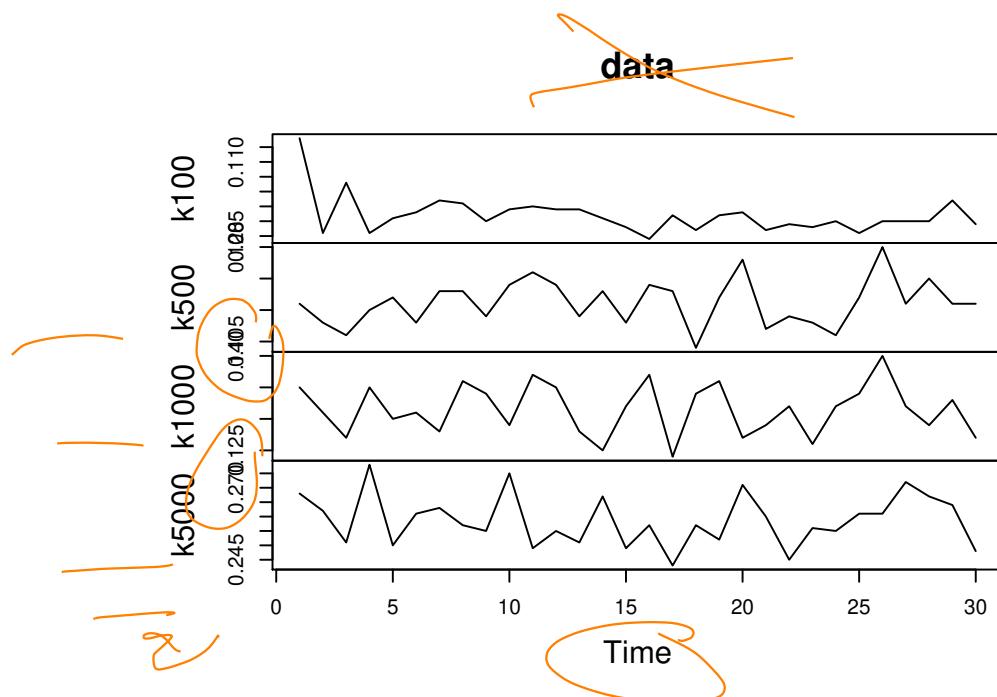


Figura 2: Simulación de 30 réplicas

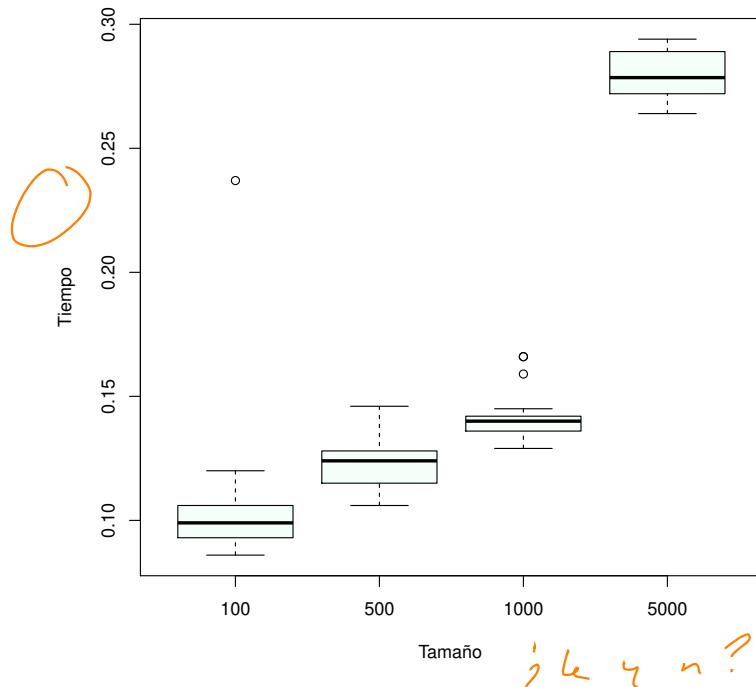


Figura 3: Tiempo del experimento

Referencias

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

Práctica 8: modelo de urnas

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

El modelo de urnas arriba los fenómenos coalescencia y fragmentación, de modo en que une las partículas que se encuentran divididas en dos o más pedazos más pequeños; siempre siendo un entero. Este experimento tiene una reacción parecida a los agentes coagulantes en donde las partículas más pequeñas se agrupan para formar una más grande y así poder ser separada de una manera más fácil.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [2], de manera que al paralelizar se vuelva más eficiente realizando varias tareas al mismo tiempo y así reducir el tiempo total de la ejecución del código, asimismo se busca visualizar el cambio al generar repeticiones del experimento.

2.1. Descripción

La finalidad del experimento es [2]:

“Paralelizar tanto como resulte eficientemente posible en esta simulación y medir cuánto tiempo se logra ahorrar, de igual manera evaluar si el ahorro es estadísticamente significativo para diferentes combinaciones de partículas y cúmulos.

Para el primer reto, se determinó en cuál momento (en términos del número de iteraciones de la simulación) la fracción de éstas alcance un máximo. Calcula y gráfica esta fracción a lo largo de la simulación. Realiza 30 réplicas para determinar si el número de paso en el cual se alcanza el máximo tiene un comportamiento sistemático.”

3. Resultados y conclusiones

En base al trabajo anteriormente reportado [1], se realizó el código que se muestra en la parte inferior. En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajaría, asimismo se ejecutó una parallelización para que el experimento fuera más eficiente. Además se realizó

una comparación de la ejecución del experimento en donde se ponderaron diferentes valores para las partículas que están representadas como “k” y para los cúmulos definidos con la variante “n”, como se muestra en el cuadro 1.

Cuadro 1: Combinación de valores de variables del experimento

Nombre	n	k
C1	100	10000
C2	1000	100000
C3	10000	1000000

```

1 k <- c(100, 1000, 10000)
2 n <- c(10000, 100000, 1000000)
3
4 suppressMessages(library(doParallel))
5 registerDoParallel(clust)
6 cluster <- makeCluster(detectCores() - 1)
7 clusterExport(cluster, "romperse")
8 clusterExport(cluster, "rotura")
9 clusterExport(cluster, "funromperse")
10 clusterExport(cluster, "c")
11 clusterExport(cluster, "d")
12 clusterExport(cluster, "assert")
13 clusterExport(cluster, "fununirse")
14 clusterExport(cluster, "union")
15 clusterExport(cluster, "unirse")
16 TimeF <- NULL
17 TimeF <- microbenchmark(exect2 = {
18   originales <- rnorm(k)
19   cumulos <- originales - min(originales) + 1
20   cumulos <- round(n * cumulos / sum(cumulos))
21   assert(min(cumulos) > 0)
22   diferencia <- n - sum(cumulos)
23   if (diferencia > 0) {
24     for (i in 1:diferencia) {
25       p <- sample(1:k, 1)
26       cumulos[p] <- cumulos[p] + 1
27     }
28   } else if (diferencia < 0) {
29     for (i in 1:-diferencia) {
30       p <- sample(1:k, 1)
31       if (cumulos[p] > 1) {
32         cumulos[p] <- cumulos[p] - 1
33       }
34     }
35   }
36 }
37 for (paso in 1:duration) {
38   assert(sum(cumulos) == n)
39   cumulos <- integer()
40
41   for (i in 1:dim(freq)[1]) { # fase de rotura

```

```

43     urna <- freq[i,]
44     if (urna$tam > 1) { # no tiene caso romper si no se puede
45       cumulos <- c(cumulos, romperse(urna$tam, urna$num))
46     } else {
47       cumulos <- c(cumulos, rep(1, urna$num))
48     }
49   }
50 }
51 }, times = 10, unit = "s")
52 stopCluster(clust)
53 print(Ptime)
54 print(TimeF)
55 tiempos <- rbind(Ptime, TimeF)
56 tiempos <- data.frame(tiempos)
57 tiempos$time <- tiempos$time/10**9
58 tiempos$expr <- as.factor(tiempos$expr)

```

Como se menciona en el apartado anterior, para observar cambios significativos en el tiempo del experimento se vario los valores de las partículas y cúmulos respectivamente, de manera que los datos proporcionados se muestran en la gráfica 1, en donde se puede observar claramente que los tiempos al paralelizar son mejores a comparación de la ejecución normal. Por lo tanto es un factor determinante para optar por el uso de dicha función.

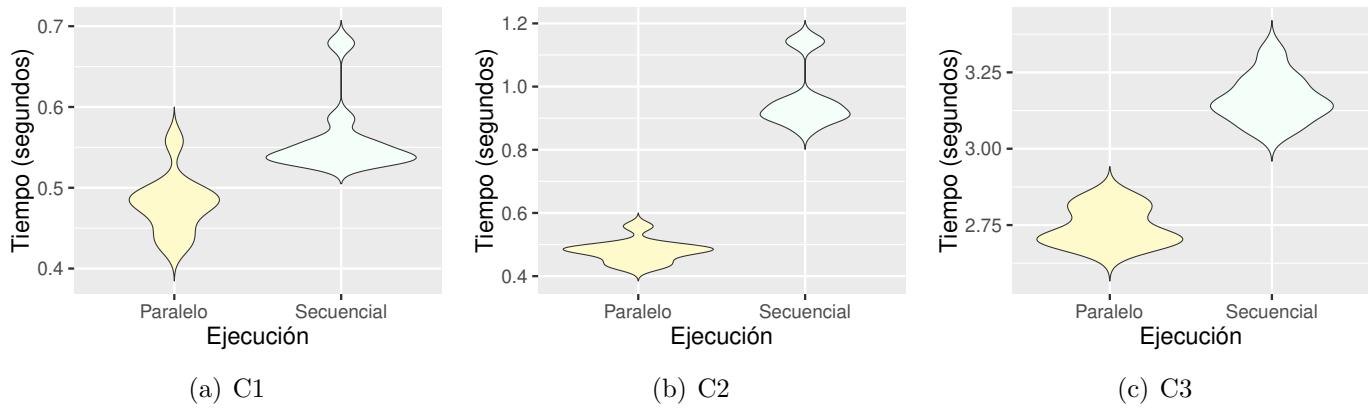


Figura 1: Tiempo del experimento

3.1. Reto 1

Para el primer reto se creó un código en el cual se pueda replicar por 30 veces, como se muestra en las líneas de la parte inferior. Asimismo se usaron las combinaciones anteriores para determinar si el experimento tiene un comportamiento sistemático.

```

1 k <- c(100, 1000, 10000)
2 n <- c(10000, 100000, 1000000)
3
4 funromperse <- function(i){
  return(as.vector(romperse(as.numeric(freq[i,][1]), as.numeric(freq[i,][2]))))
5

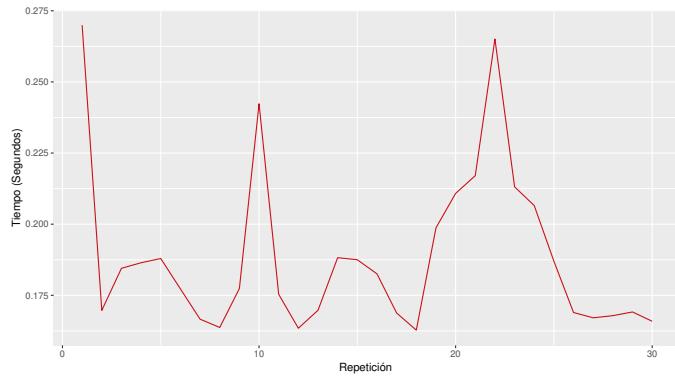
```

```

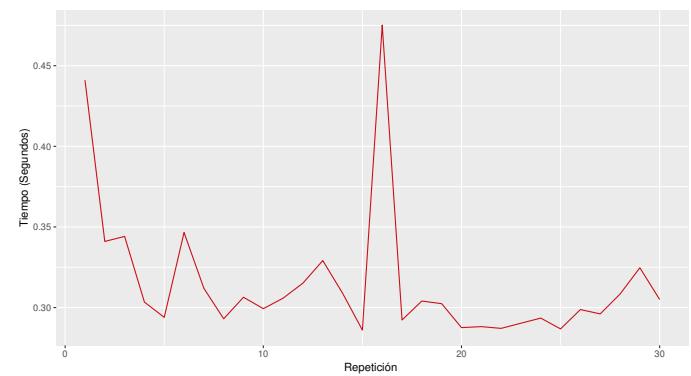
6  }
7  fununirse <- function(i){
8    return(unirse(as.numeric(freq[i,][1]), as.numeric(freq[i,][2])))
9  }
10
11 cluster <- makeCluster(detectCores()-1)
12 clusterExport(cluster, "romperse")
13 clusterExport(cluster, "rotura")
14 clusterExport(cluster, "funromperse")
15 clusterExport(cluster, "c")
16 clusterExport(cluster, "d")
17 clusterExport(cluster, "assert")
18 clusterExport(cluster, "fununirse")
19 clusterExport(cluster, "union")
20
21
22 vartime <- data.frame()
23 for (repeticiones in 1:30) {
24   tiempoinicial <- Sys.time()
25   for (paso in 1:duration) {
26     assert(sum(cumulos) == n)
27     cumulos <- integer()
28     clusterExport(cluster, "freq")
29     cumulos <- as.vector(parSapply(cluster, 1:(dim(freq)[1]), funromperse))
30     a <- c()
31     for(i in 1:length(cumulos)){
32       a <-c(a, cumulos[[i]])
33     }
34     cumulos <- a
35
36   clusterExport(cluster, "freq")
37   cumulos <- as.vector(parSapply(cluster, 1:(dim(freq)[1]), fununirse))
38   a <- c()
39   for(i in 1:length(cumulos)){
40     a <-c(a, cumulos[[i]])
41   }
42   cumulos <- a

```

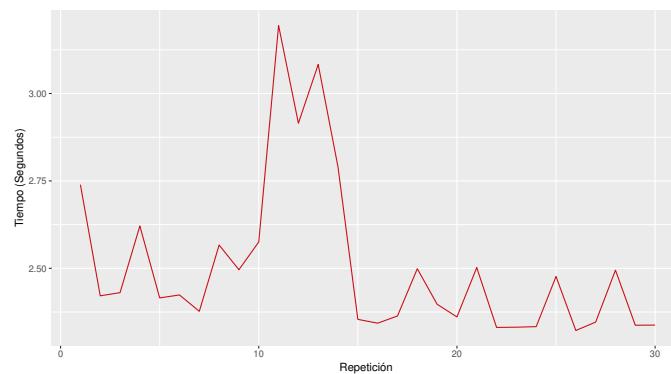
En la figura 2 se muestra la variación del máximo de las ejecuciones, las cuales demuestran una tendencia a disminuir la variación en el tiempo de ejecución de las replicas, debido al incremento en los valores de la combinación de partículas y cúmulos. De manera que se puede determinar que el experimento tiene un comportamiento sistemático.



(a) Tiempo de C1



(b) Tiempo de C2



(c) Tiempo de C3

Figura 2: Iteraciones de la simulación

Referencias

- [1] Astrid González. Exercise 8, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/files/Practica%208/>.
- [2] Elisa Schaeffer. Práctica 8: modelo de urnas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p8.html>.

Práctica 9: interacciones entre partículas

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la novena práctica tuve algunos mal entendidos con la fuerza que realice; hice que las partículas tuvieran una fuerza de tracción, en vez de tener una fuerza de tipo gravitacional, puesto que malinterpreté el texto de la página web, pensando en que estas serían influenciadas por el valor de la fuerza de gravedad universal. Por lo que en el documento final, se realizaron cambios en las líneas del código para cumplir con el objetivo planteado. De igual manera, de acuerdo a las mejoras marcadas en la revisión, se hizo los cambios correspondientes en las gráficas

Práctica 9: interacciones entre partículas

Ricardo Rosas Macías

30 de marzo de 2019

1. Introducción

En la práctica se muestra la interacción de partículas que tienen un movimiento en función a las cargas, de manera que las partículas crean fuerzas de atracción y repulsión entre ellas al estar cerca de su campo.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [1], de tal forma que el movimiento de las partículas esté ligado a la masa que esta posee, de modo que emule a una fuerza de tipo gravitacional; resultando una manifestación de atracción entre ellas.

2.1. Descripción

La finalidad del experimento es [1]:

“Aregar a cada partícula una masa y hacer que la masa cause fuerzas gravitacionales (atracciones) además de las fuerzas causadas por las cargas. Asimismo, estudiar la distribución de velocidades de las partículas y verifica gráficamente que esté presente una relación entre los tres factores: la velocidad, la magnitud de la carga, y la masa de las partículas.”

3. Resultados y conclusiones

En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajó, además se creó una masa para todas partículas que posteriormente tendrá un efecto en la interacción con todas las que se encuentren en su alrededor, igualmente se estableció parámetros para que dicha masa sea una fuerza más en el experimento.

```
1 n <- 50
2 p <- data.frame(x = rnorm(n), y=rnorm(n), c=rnorm(n), m= (rexp(n, rate = 1.2) +1))
3
4 mi <- p[i,]$m
```

```
5  
6 return(c(fx , fy)/(mi)) ?
```

De acuerdo a lo anteriormente mencionado, esta masa efectuará cambios positivos en la carga de la partícula. Por lo tanto en las siguientes líneas de código se muestra el efecto que tendrá la masa de la partícula tanto a la fuerza gravitacional entre todas las partículas así como el cambio en la velocidad por dicho fenómeno durante los 100 pasos posteriormente declarados.

```
1 valores <- data.frame()  
2 for (iter in 1:tmax) {  
3   val <- c()  
4  
5   Fcreada <- delta*f  
6  
7   for(t in seq(1, (length(Fcreada) -1), by = 2)){  
8     val <- c(val , (sum((Fcreada[t])^2, (Fcreada[t +1])^2))^2)^{(1/2)}  
9   }  
10  valores <- rbind(valores , val)  
11  
12 tprom <- c()  
13 for(q in 1:50){  
14   tprom <- c(tprom , mean(valores[,q]))
```

2 fuerzas?

Con ayuda de la paquetería *Lattice* se obtuvo la visualización de la ejecución del código, como se muestra en la figura 1, con la cual se determina que el código creado realmente si cumple con el objetivo planteado. Se observa que al generar las partículas muchas empiezan a interactuar con sus vecinas de manera que ocasionan una convergencia para todas las partículas.

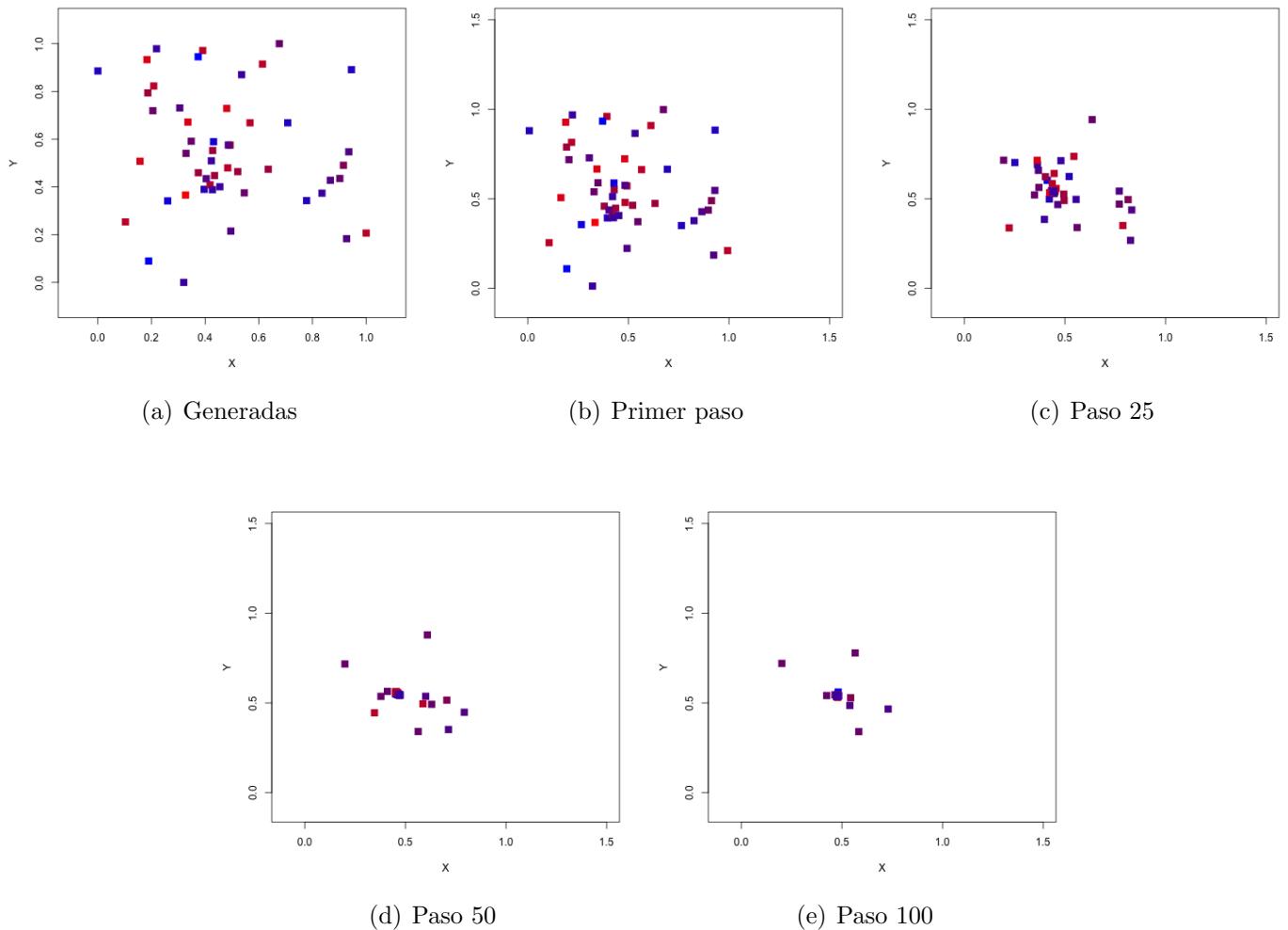


Figura 1: Atracción de partículas en la simulación

Para estudiar la distribución de velocidades las partículas se obtuvieron histogramas como se puede observar en la figura 2, que permiten ver una normalización de la velocidad respecto a la distancia recorrida por la interacción de las cargas de las partículas.

Por último se obtuvo la figura 3 para ver de manera más puntual la distribución mencionada en la sección anteriormente, En donde se evidencia que conforme avanza el número movimientos respecto a los pasos establecidos, estos van disminuyendo de forma que ~~no~~ indica que el experimento se mantiene estable, sin afectar drásticamente con un rechazo por las cargas iguales.

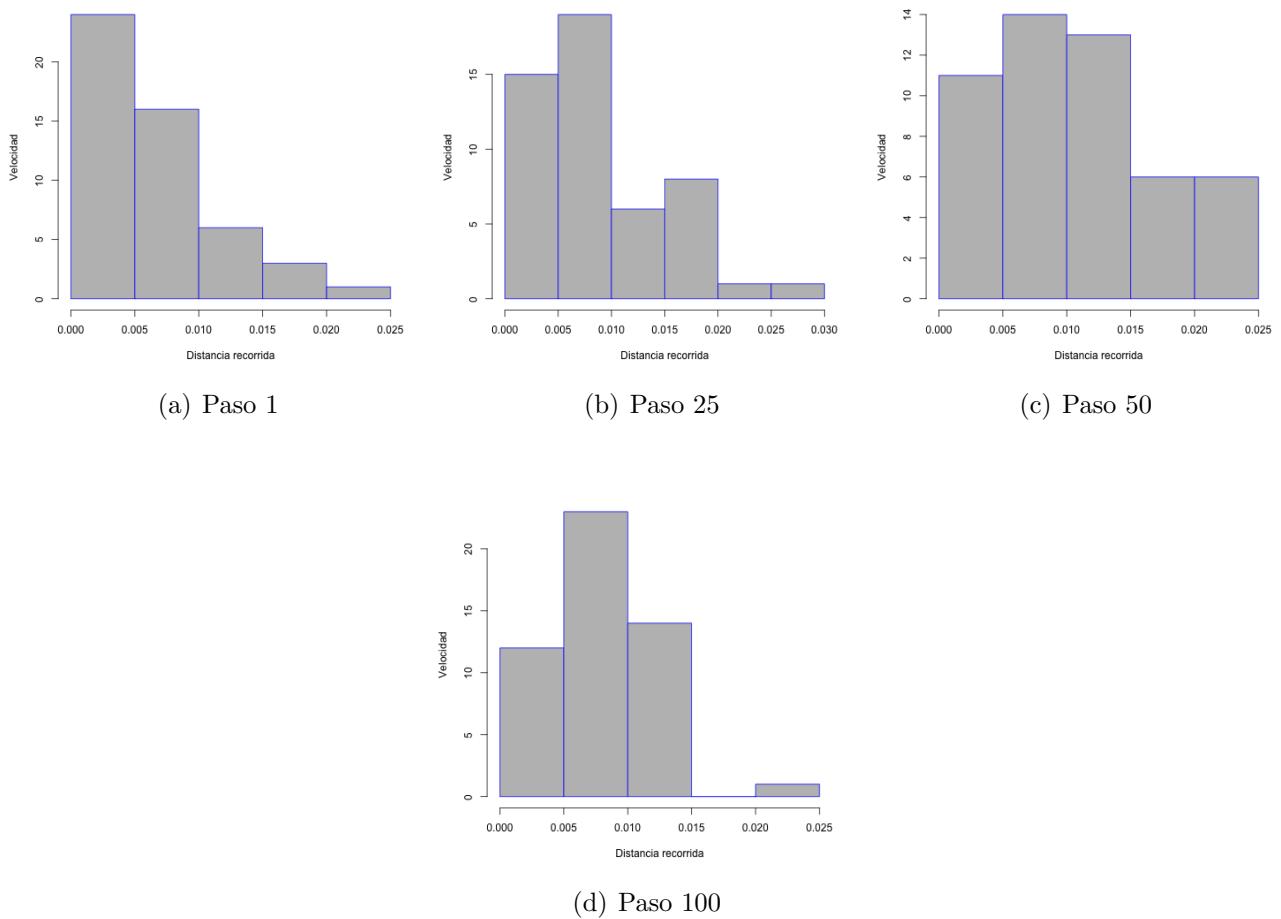


Figura 2: Distancia de partículas

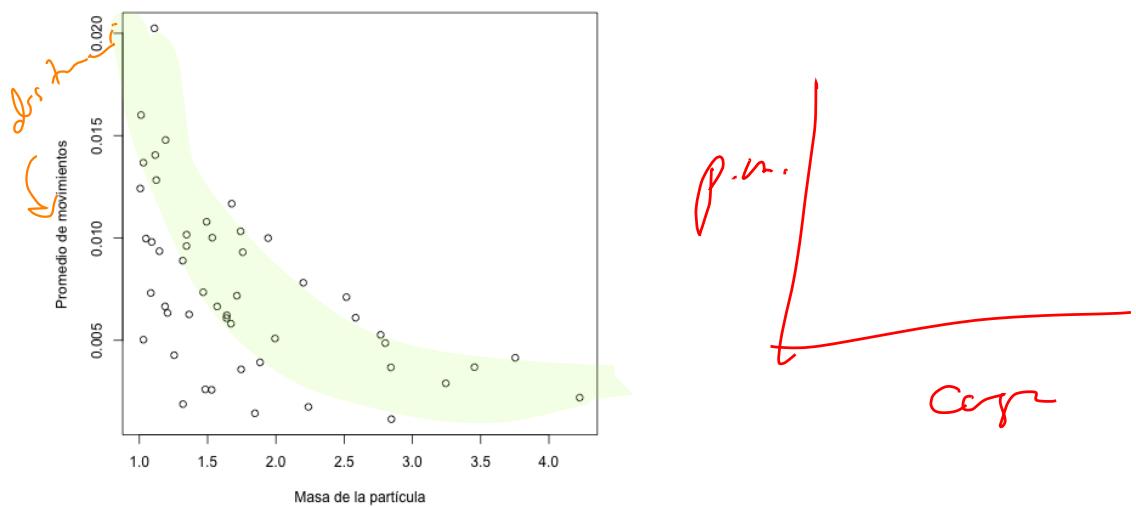


Figura 3: Movimientos respecto a la masa de las partículas

Referencias

- [1] Elisa Schaeffer. Práctica 9: interacciones entre partículas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.

Práctica 9: interacciones entre partículas

Ricardo Rosas Macías

20 de mayo de 2019

1. Introducción

En la práctica se muestra la interacción de partículas que tienen un movimiento en función a las cargas, de manera que las partículas crean fuerzas de atracción y repulsión entre ellas al estar cerca de su campo.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [2], de tal forma que el movimiento de las partículas esté ligado a la masa que esta posee, de modo que emule a una fuerza de tipo gravitacional; resultando una manifestación de atracción entre ellas.

2.1. Descripción

La finalidad del experimento es [2]:

“Aregar a cada partícula una masa y hacer que la masa cause fuerzas gravitacionales (atracciones) además de las fuerzas causadas por las cargas. Asimismo, estudiar la distribución de velocidades de las partículas y verifica gráficamente que esté presente una relación entre los tres factores: la velocidad, la magnitud de la carga, y la masa de las partículas.”

3. Resultados y conclusiones

En base al trabajo anteriormente reportado [3][4], se realizó el código que se muestra en la parte inferior. En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajó, además se creó una masa para todas partículas que posteriormente tendrá un efecto en la interacción con todas las que se encuentren en su alrededor, igualmente se estableció parámetros para que dicha masa sea una fuerza más en el experimento.

```
1 n <- 50
2 p <- data.frame(x = rnorm(n), y=rnorm(n), c=rnorm(n), m=rnorm(n))
3 p$m <- abs(p$m)
```

De acuerdo a lo anteriormente mencionado, esta masa efectuará cambios positivos en la carga de la partícula. Por lo tanto en las siguientes líneas de código se muestra el efecto que tendrá la masa de la partícula tanto a la fuerza gravitacional entre todas las partículas así como el cambio en la velocidad por dicho fenómeno durante los 100 pasos posteriormente declarados.

```

1 mi <- p[i,]$m
2
3 pi<-cbind(p$x,p$y)
4 pi<-data.frame(pi)
5 colnames(pi)<-c('x','y')
6
7 disx<-pi$x-p$x
8 disy<-pi$y-p$y
9 v<-sqrt(disx^2+disy^2)
10 res<-cbind(p$m,v)
11 datos<-rbind(datos,res)
12
13 ggplot() +
14   geom_point(data=p, aes(x = p$x, y= p$y, size=p$m, color=p$g))+
15   scale_x_continuous(name="x",limits = c(0, 1))+ 
16   scale_y_continuous(name="y", limits = c(0, 1))+ 
17   scale_colour_manual(values=colores)+ 
18   ggtile(paste("Paso",iter))+ 
19   theme(plot.title = element_text(hjust = 0.5))+ 
20   guides(size=FALSE, color=guide_legend(title="Carga"))+ 
21   theme(axis.text.x=element_text(size=12), 
22         axis.text.y=element_text(size=12), 
23         plot.title=element_text(size=14), 
24         axis.title.x = element_text(size=12), 
25         axis.title.y = element_text(size=12))
26   ggsave(paste("P9_t",iter,".png", sep=""))
27 }
28 stopImplicitCluster()
29
30 datos$n<-seq(1,n,1)
31 colnames(datos)<-c("Masa","Velocidad","n")
32 tabla<-data.frame()
33
34 for (i in 1:n){
35   res<-datos[datos$n==i,]
36   resultados<-cbind(res[i,]$Masa,mean(res$Velocidad))
37   tabla<-rbind(tabla,resultados)
38 }
39 colnames(tabla)<-c("Masa","Velocidad")
40
41 ggplot(tabla, aes(x=Masa, y=Velocidad))+ 
42   geom_point(shape = 17, color = "#FC4E07", size = 1.7)+ 
43   geom_smooth(method = "lm", formula = y ~ log(x))+ 
44   scale_x_continuous(name="Masa")+
45   scale_y_continuous(name="Velocidad")

```

```
47 ggsave("Veldep.png")
```

Con ayuda de la paquetería *Lattice* se obtuvo la visualización de la ejecución del código, como se muestra en la figura 1, con la cual se determina que el código creado realmente si cumple con el objetivo planteado. Se observa que al generar las partículas muchas empiezan a interactuar con sus vecinas de manera que ocasionan una convergencia para todas las partículas.

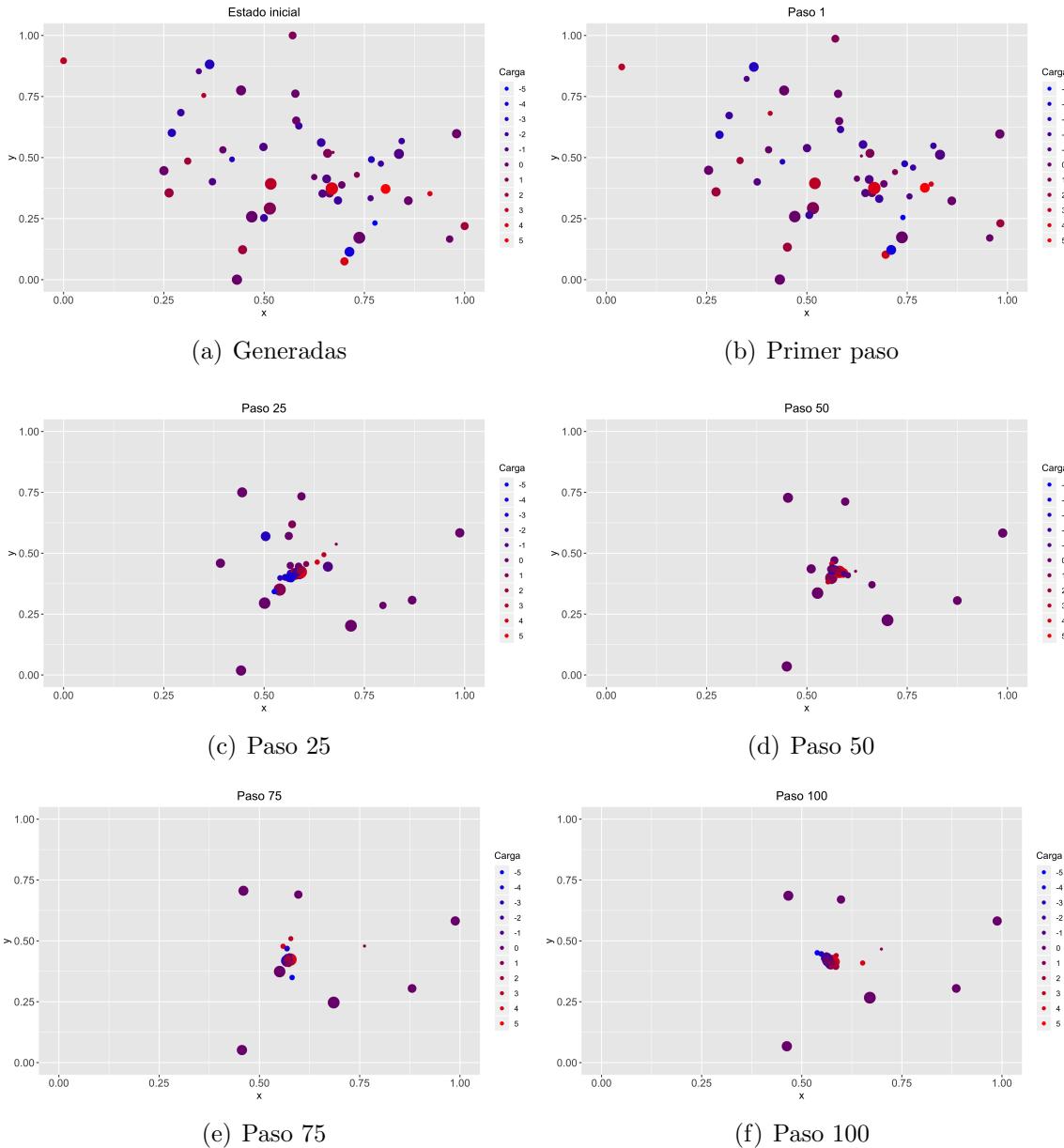


Figura 1: Atracción de partículas en la simulación

Para estudiar la distribución de velocidades las partículas se obtuvieron histogramas como se puede observar en la figura 2, que permiten ver una normalización de la velocidad respecto a la distancia recorrida por la interacción de las cargas de las partículas.

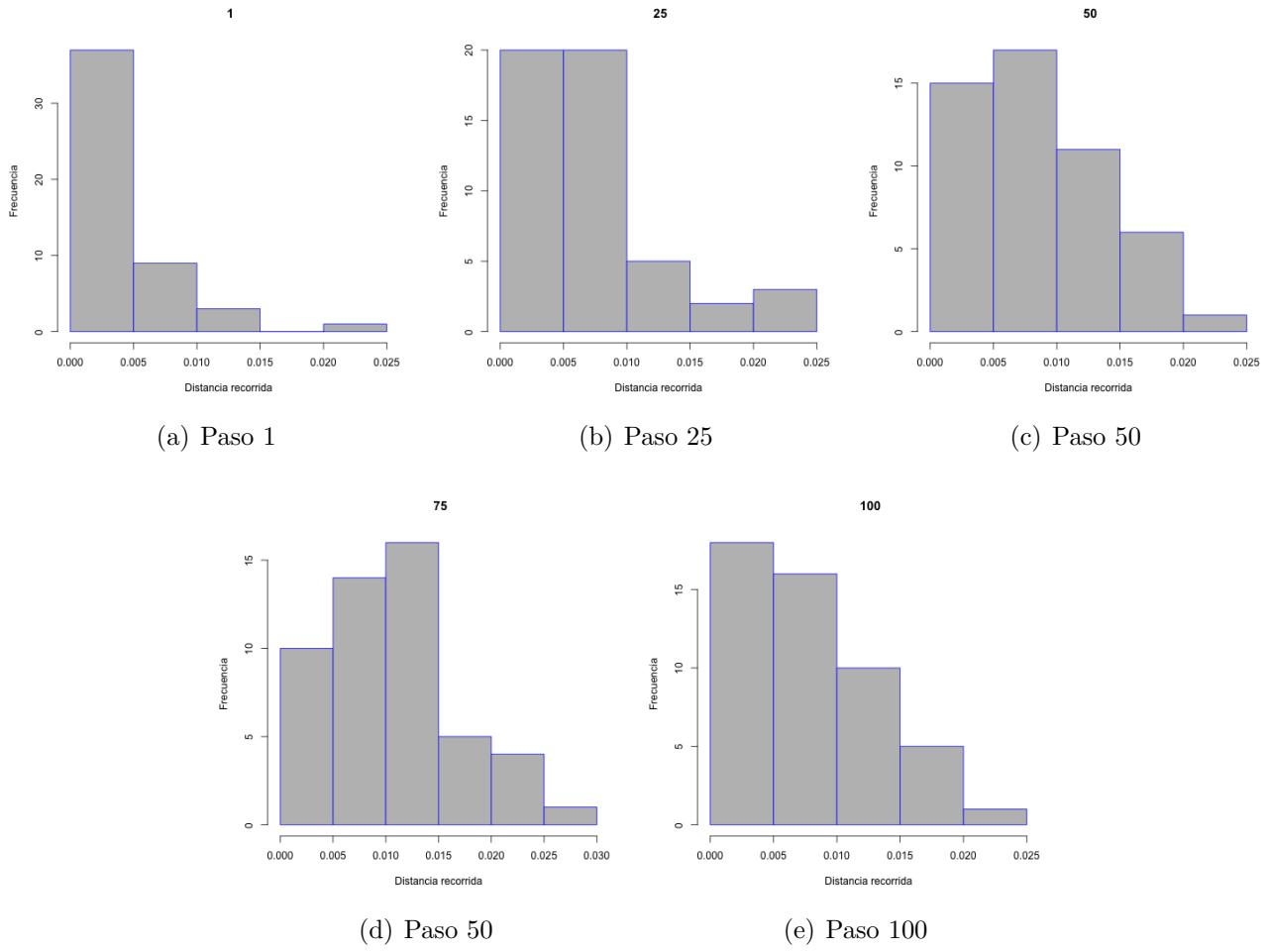


Figura 2: Distancia de partículas

Por último se obtuvo la figura 3 para ver de manera más puntual la distribución mencionada en la sección anteriormente, en donde se evidencia que conforme avanza el número movimientos respecto a la velocidad de los pasos establecidos; estos van disminuyendo, de forma que indica que el experimento se mantiene estable, sin afectar drásticamente con un rechazo por las cargas iguales.

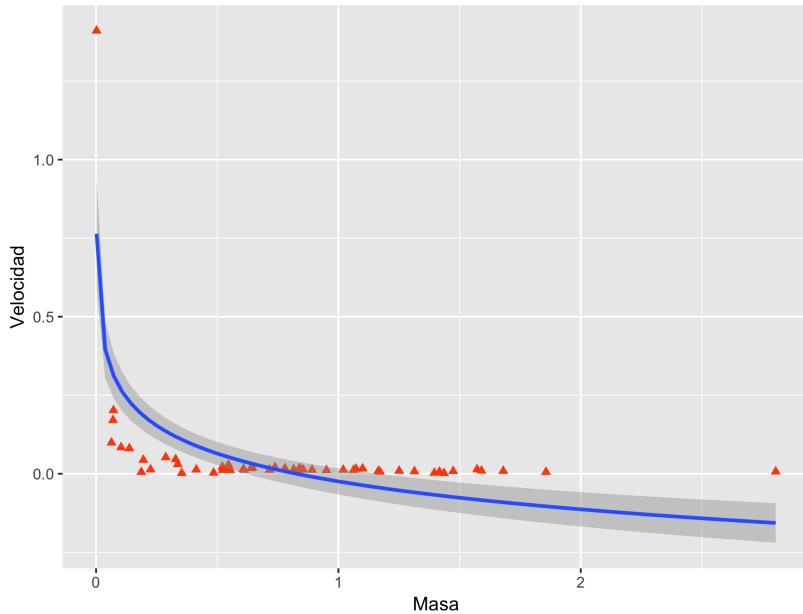


Figura 3: Vel respecto a la masa de las partículas

Referencias

- [1] Ricardo Rosas Macías. Práctica 9: interacciones entre partículas, 2019. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP9>.
- [2] Elisa Schaeffer. Práctica 9: interacciones entre partículas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.
- [3] José Ángel A. Contreras. Práctica 9, 2018. URL https://github.com/Xhangelx13x/SimNano_Xhan/tree/master/P9.

Práctica 10: algoritmo genético

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la décima práctica al principio tuve problemas para generar las instancias, me costo trabajo entender el objetivo planteado; para interpretar y llevar acabo los objetivos del código. Por otro lado, en el documento final se pueden observar cómo creé las instancias, además de una mejor representación del análisis estadístico; el documento tiene mejores gráficas representativas de la ejecución del código. Cabe resaltar que desconocía del problema de la mochila, pero la aplicación en esta práctica me ayudo a qué quedará clara su finalidad.

Práctica 10: algoritmo genético

Ricardo Rosas Macías

9 de abril de 2019



1. Introducción

En la práctica se aborda el problema de la mochila; mejor conocido como Knapsack, en el cual se busca la optimización de la combinación posible para la mezcla de artículos máximo que puede estar dentro de esta, en función al peso de cada uno.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [2], de tal forma que la programación dinámica con ayuda de los algoritmos permite crear de una manera más sencilla la mejor combinación posible.

2.1. Descripción

La finalidad del experimento es [2]:

“Paralelizar el algoritmo genético y estudia los efectos en su tiempo de ejecución con pruebas estadísticas y visualizaciones, variando el número de objetos en la instancia. Genere instancias con tres distintas reglas: el peso y el valor de cada objeto se generan independientemente con una distribución normal, 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, 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.

Asimismo, determinar para cada uno de los tres casos a partir de qué tamaño de instancia el algoritmo genético es mejor que el algoritmo exacto en términos de valor total obtenido por segundo de ejecución.”

3. Resultados y conclusiones

En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajó; se estableció una serie de instancias como se describen en el objetivo. El código permite exhibir la eficiencia de selección de muestra en el experimento; siendo repetido 5 veces para obtener un valor estadístico más exacto. Además se parallelizó y configuró para que proporcione el tiempo que toma en ejecutar el código, de manera que logre mostrar las diferencias en tiempos de ejecución normal y parallelizada.

```
1 cluster <- makeCluster(detectCores() - 1)
2 datos=data.frame()
3 for(init in c(25, 50, 75)){
4   print(init)
5   for(replica in 1:5){
6
7     startP=Sys.time()
8     print("parallel")
9     p <- as.data.frame(t(parSapply(cluster, 1:init, function(i){return(round(runif(n))))}))
10
11   clusterExport(cluster, "p")
12   mutan=sample(1:tam, round(pm*tam))
13   p <- rbind(p,(t(parSapply(cluster, mutan, function(i){return(mutacion(unlist(p[i,]), n)))))))
14   clusterExport(cluster, "tam")
15   clusterExport(cluster, "p")
16   padres <- parSapply(cluster, 1:rep, function(i){return(sample(1:tam, 2, replace=
17     FALSE)))})
18   hijos <- parSapply(cluster, 1:rep, function(i){return(as.matrix(unlist(reproduccion(
19     p[padres[1, i],], p[padres[2, i],], n)), ncol=n))})
20   p = rbind(p, hijos)
21
22 tam <- dim(p)[1]
23 clusterExport(cluster, "p")
24 obj=parSapply(cluster, 1:tam, function(i){return(objetivo(unlist(p[i,]), valores))})
25 fact=parSapply(cluster, 1:tam, function(i){return(factible(unlist(p[i,]), pesos,
26   capacidad))})
27
28 p <- cbind(p, obj)
29 p <- cbind(p, fact)
30 mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
31 p <- p[mantener,]
32 tam <- dim(p)[1]
33 assert(tam == init)
34 factibles <- p$p$fact == TRUE,]
35 mejor <- max(factibles$obj)
36 mejores <- c(mejores, mejor)
37 }
38 mejorP=mejor
39 endP=Sys.time()
40 startS=Sys.time()
41 print("normal")
```

Con ayuda de la paquetería *ggplot2* [1] se obtuvo la visualización de código anterior. En la figura 1 se puede observar en como la paralelización del experimento puede permitir minimizar los tiempos de ejecución. Asimismo, al incrementar la instancia se puede ver un comportamiento más abrupto en la ejecución normal, en comparación del paraleizado; en donde muestra cambios ligeros que pasan inapercibidos.

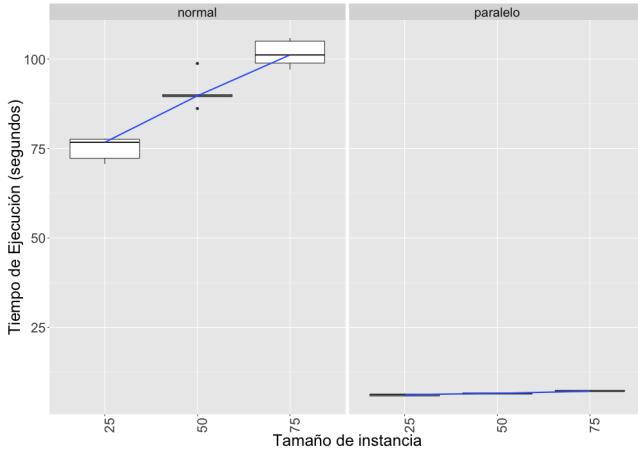


Figura 1: Comparación de ejecución de código

Referencias

- [1] Hadley Wickham et al. Package ‘ggplot2’, 2019. URL <https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf>.
- [2] Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

Práctica 10: algoritmo genético

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

En la práctica se aborda el problema de la mochila; mejor conocido como Knapsack, en el cual se busca la optimización de la combinación posible para la mezcla de artículos máximo que puede estar dentro de esta, en función al peso de cada uno.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [2], de tal forma que la programación dinámica con ayuda de los algoritmos permite crear de una manera más sencilla la mejor combinación posible.

2.1. Descripción

La finalidad del experimento es [2]:

“Paralelizar el algoritmo genético y estudia los efectos en su tiempo de ejecución con pruebas estadísticas y visualizaciones, variando el número de objetos en la instancia. Genere instancias con tres distintas reglas: el peso y el valor de cada objeto se generan independientemente con una distribución normal, 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, 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.

Asimismo, determinar para cada uno de los tres casos a partir de qué tamaño de instancia el algoritmo genético es mejor que el algoritmo exacto en términos de valor total obtenido por segundo de ejecución.”

3. Resultados y conclusiones

Se realizó el código en base a los antecedentes anteriormente reportados [3]. En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajó; se estableció una serie de instancias como se describen en el objetivo. El código permite exhibir la eficiencia de selección de muestra en el experimento; siendo repetido 5 veces para obtener un valor estadístico más exacto. Además se parallelizó y configuró para que proporcione el tiempo que toma en ejecutar el código, de manera que logre mostrar las diferencias en tiempos de ejecución normal y paralelizada.

```
1 startP=Sys.time()
2 n <- 50
3 generaciones <- c(200, 250, 300)
4 pmutaciones <- c(0.05, 0.1, 0.2)
5 reproducciones <- c(50, 75, 100)
6 iteraciones <- c(10, 15, 20)
7 replicas <- 5
8
9 datos <- data.frame(Replica = integer(), Optimo = integer(),
10                      Generaciones = numeric(), ProbMut = numeric(),
11                      Reproducciones = numeric(), Iteraciones = numeric(),
12                      Objetivo = double())
13 cluster <- makeCluster(detectCores())
14 opt <- knapsack(capacidad, pesos, valores)[2]
15 clusterExport(cluster, "n")
16 clusterExport(cluster, "pesos")
17 clusterExport(cluster, "valores")
18 clusterExport(cluster, "capacidad")
19
20 for (tmax in iteraciones) {
21   for (rep in reproducciones) {
22     for (pm in pmutaciones) {
23       for (init in generaciones) {
24         for (r in 1:replicas) {
25
26           #Inicio del genetico parsapply
27           pobl <- t(parSapply(cluster, 1:init, function(i) {
28             return(round(runif(n)))
29           }))
30
31           p <- as.data.frame(pobl)
32           tam <- dim(p)[1]
33
34           clusterExport(cluster, "p")
35           clusterExport(cluster, "tam")
36
37           probabilidades <- runif(tam) < pm
38           mutados <- which(probabilidades %in% TRUE)
39
40           mutaciones <- t(parSapply(cluster, mutados, function(i) {
41             pos <- sample(1:n, 1)
42             mut <- p[i,]
43             mut[pos] <- (!p[i,][pos]) * 1
44             return(as.numeric(mut))
45           }))
46
47           }
48
49           }
50
51           }
52
53           }
54
55           }
56
57           }
58
59           }
60
61           }
62
63           }
64
65           }
66
67           }
68
69           }
70
71           }
72
73           }
74
75           }
76
77           }
78
79           }
80
81           }
82
83           }
84
85           }
86
87           }
88
89           }
90
91           }
92
93           }
94
95           }
96
97           }
98
99           }
100
101           }
102
103           }
104
105           }
106
107           }
108
109           }
110
111           }
112
113           }
114
115           }
116
117           }
118
119           }
120
121           }
122
123           }
124
125           }
126
127           }
128
129           }
130
131           }
132
133           }
134
135           }
136
137           }
138
139           }
140
141           }
142
143           }
144
145           }
```

```

46
47 hijos <- matrix(parSapply(cluster , 1:rep , function(i) {
48   padres <- sample(1:tam, 2, replace = FALSE)
49   pos <- sample(2:(n-1), 1)
50   x <- p[padres[1],]
51   y <- p[padres[2],]
52   xy <- c(x[1:pos], y[(pos+1):n])
53   yx <- c(y[1:pos], x[(pos+1):n])
54   return(as.numeric(c(xy, yx)))
55 }), ncol = n, byrow = TRUE)
56
57 p <- rbind(p, mutaciones, hijos)
58 tam <- dim(p)[1]
59
60 clusterExport(cluster , "p")
61 clusterExport(cluster , "tam")
62
63 p$obj <- parSapply(cluster , 1:tam, function(i) {
64   return(sum(p[i,] * valores))
65 })
66
67 p$fact <- parSapply(cluster , 1:tam, function(i) {
68   return(sum(p[i,] * pesos) <= capacidad)
69 })
70
71 mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
72
73 p <- p[mantener,]
74 tam <- dim(p)[1]
75
76 assert(tam == init)
77
78 factibles <- p[p$fact == TRUE,]
79 mejor <- max(factibles$obj)
80 mejores <- c(mejores, mejor)
81
82 }
83
84 datos <- rbind(datos, data.frame(Replica = r, Optimo = optimo,
85                           Generaciones = init, ProbMut = pm,
86                           Reproducciones = rep, Iteraciones = tmax,
87                           Objetivo = max(mejores)))
88   }
89 }
90 }
91 }
92 }
93 stopCluster(cluster)
94 endP=Sys.time()

```

En la figura 1 se puede observar en como la paralelización del experimento puede permitir minimizar los tiempos de ejecución. Asimismo, al incrementar la instancia se puede ver un comportamiento más abrupto en la ejecución normal, en comparación del paralelizado; que muestra ligeros cambios.

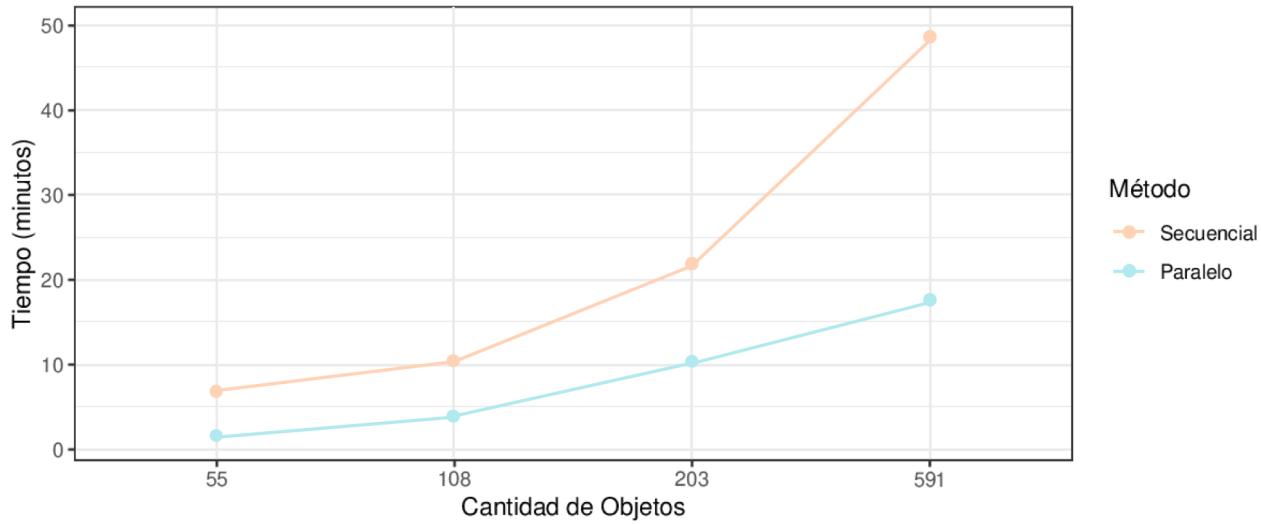


Figura 1: Comparación de ejecución de código

Con ayuda de la paquetería *ggplot2* [1] se obtuvo la visualización del error de la instancia creada, de modo que permite demostrar de manera cuantitativa el ruido del experimento. En la figura [2] se logra denotar que al incrementar las variables este tiene un ruido que se mantiene constante en comparación con el comportamiento de las otras cantidades de generaciones.

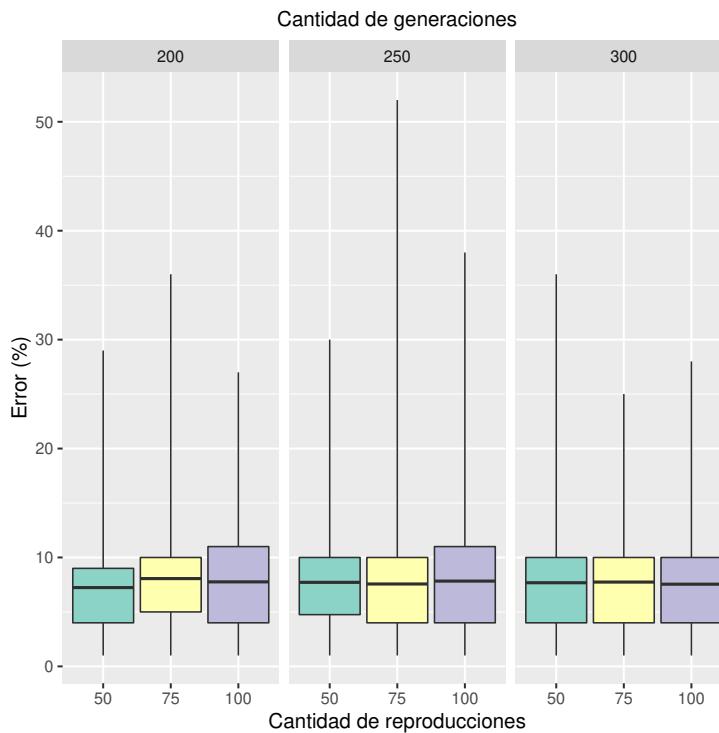


Figura 2: Porcentaje de error respecto a las instancias generadas

Referencias

- [1] Hadley Wickham et al. Package ‘ggplot2’, 2019. URL <https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf>.
- [2] Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.
- [3] Ángel Moreno. Homework 10, 2018. URL <https://github.com/angisabel44/Simulation/tree/master/Homework10>.

Práctica 11: frentes de Pareto

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la décimo primera práctica se puede observar que tuve mínimos errores. Asimismo, el primer documento se entregó con la tarea y dos retos terminados; por primera vez, por lo cual tuvo una de las mejores puntuaciones en comparación con todas las prácticas anteriores. Esto se puede atribuir debido a que tuve más tiempo para realizarla; contaba con el tiempo del periodo vacacional.

Por otra parte, antes desconocía de este tema, pero me pareció muy interesante, además que creó que en un futuro lo podría aplicar en la investigación e I+D, para la selección de materiales.

Práctica 11: frentes de Pareto

Ricardo Rosas Macías

18 de abril de 2019

1. Introducción

El análisis frentes de Pareto, permite obtener un resultado mediante la optimización de multi-objetivo, con ayuda de criterios de utilidad; lo cual permite discernir y proporcionar un solución en equilibrio de estas dos entidades, dentro de los márgenes de las variantes.

2. Objetivo

Se realizó cambios en el código del sitio web [3], de modo que proporcione una solución optima que mejore en un objetivo, descartando eficazmente las demás opciones; de manera que esta sea dejada en la denominada frontera de Pareto.

2.1. Descripción

La finalidad del experimento es [3]:

“Paralelizar el cálculo y graficar el porcentaje de soluciones de Pareto, como función del número de funciones objetivo con diagramas de violín combinados con diagramas de caja-bigote, verificando que las diferencias observadas sean estadísticamente significativas.

El primer reto es seleccionar un subconjunto (cuyo tamaño como un porcentaje del frente original se proporciona como un parámetro) 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.

El segundo reto es adaptar el algoritmo genético de la práctica anterior para que vaya buscando mejora a un frente; la población inicial es el frente generado en la tarea y se aplique la diversificación del primer reto a cada generación después de los cruzamientos y las mutaciones.”

3. Resultados y conclusiones

En base al trabajo anteriormente reportado [1], se realizó el código que se muestra en la parte inferior. En el cual las primeras líneas, exhiben los parámetros con lo que se ejecutó, asimismo en este se evidencia la paralelización de las treinta réplicas de ejecución para las funciones objetivo de 2-14.

```
1 library(parallel) (–)
2 dat <- data.frame()
3 verify <- function(i){
4   val <- c()
5   for (j in 1:k) {
6     val <- c(val, eval(obj[[j]], sol[i,], tc))
7   }
8   return(val)
9 }
10 prop <- function(i){
11   return(list(poli(vc, md, tc)))
12 }
13 cluster <- makeCluster(detectCores()-1)
14 clusterExport(cluster, c("domin.by", "sign", "eval", "poli", "pick.one"))
15
16 vc <- 4
17 md <- 3
18 tc <- 5
19 funciones <- seq(2, 14, by=1)
20 for (k in funciones) {
21   for (replicas in 1:30) {
22     obj <- list()
23     clusterExport(cluster, c("vc", "md", "tc"))
24     obj <- parSapply(cluster, 1:k, prop)
25     minim <- (runif(k) > 0.5)
26     sign <- (1 + -2 * minim)
27     n <- 200 # soluciones aleatorias
28     sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
29     val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
30     clusterExport(cluster, c("tc", "obj", "sol", "eval", "k", "n"))
31     val <- parSapply(cluster, 1:n, verify)
32     val <- t(val)
33
34 dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
35 vec <- seq(1, k, by = 1)
36 for (i in vec) {
37   if((i+1) == is.element(i+1, vec)*(i+1)){
```

Con ayuda de la paquetería *ggplot2* [2] se obtuvo la visualización de código anterior. En la figura 1 evidencia un comportamiento creciente en las soluciones no dominadas dado al incrementar las funciones objetivo.

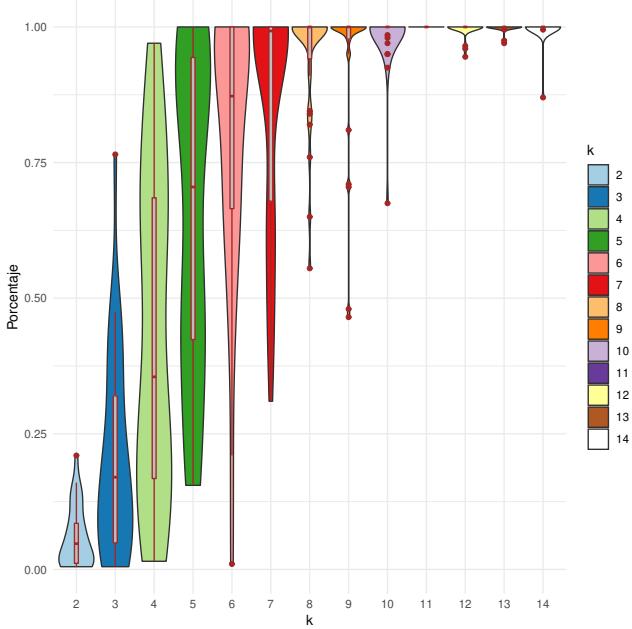


Figura 1: Porcentaje de soluciones

Asimismo, se realizó una prueba *Shapiro-Wilk* ~~test~~ para verificar la probabilidad de una distribución normal; es una manera eficiente para determinar la normalidad de las variables. Por otro lado, se hizo una comprobación con *kruskal.test* para evidenciar la existencia de alguna diferencia entre los niveles, además se ejecutó una prueba *Pairwise Wilcox test* para presentar que niveles son los causantes de estas diferencias.

```

1 a <- shapiro.test(dat$Porcentaje)
2 a$p.value
3 a$p.value < 0.05
4 b <- kruskal.test(dat$Porcentaje ~ dat$k)
5 b$p.value
6 b$p.value < 0.05
7 c <- pairwise.wilcox.test(dat$Porcentaje, dat$k, p.adjust.method = "none")
8 c$p.value
9 c$p.value < 0.05

```

Por consiguiente, se obtuvieron los resultados de *Shapiro-Wilk test* del código anterior, en donde se puede notar una diferencia en el valor $p < 0,05$, lo cual nos dice que hay diferencias relevantes en las cantidades de funciones objetivo. Por otra parte el valor $p < 0,05$ verificado con el *kruskal.test* nos muestra que solamente en algunos niveles las diferencias son despreciables. Por último con el *Pairwise Wilcox test* se expone con un *TRUE* a los niveles que presentan dichas diferencias.

```

1
2 a <- shapiro.test(dat$Porcentaje)
3 a$p.value
4 [1] 1.030554e-25
5 a$p.value < 0.05
6 [1] TRUE

```

```

7 b <- kruskal.test(dat$Porcentaje ~ dat$k)
8 b$p.value
9 [1] 7.036467e-50
10 b$p.value < 0.05
11 [1] TRUE
12
13 c <- pairwise.wilcox.test(dat$Porcentaje, dat$k, p.adjust.method = "none")
14 c$p.value
15
16 2 3 4 5 6 7
17 3 3.731573e-04 NA NA NA NA NA NA
18 4 2.731352e-06 4.354560e-02 NA NA NA NA NA
19 5 2.943146e-11 6.726009e-08 5.925208e-03 NA NA NA NA
20 6 2.181749e-10 7.452172e-08 6.305376e-04 9.600383e-02 NA NA NA
21 7 3.673572e-10 2.059118e-08 1.276700e-03 2.134350e-01 4.800559e-01 NA
22 8 1.601010e-10 2.393985e-09 1.529001e-05 1.819774e-03 2.996001e-01 3.185668e-02
23 9 1.078845e-11 1.095960e-11 3.606285e-08 1.653501e-07 4.931635e-03 2.820771e-05
24 10 9.200223e-12 1.281555e-11 2.630009e-08 2.818889e-07 3.853621e-03 1.984614e-05
25 11 1.679691e-12 1.710707e-12 1.042511e-10 9.844761e-11 6.089947e-06 4.130365e-09
26 12 3.089163e-12 3.143503e-12 7.853390e-10 2.089705e-09 7.063860e-05 1.041084e-07
27 13 3.089163e-12 3.143503e-12 8.249127e-10 2.305425e-09 2.211817e-06 1.943190e-09
28 14 1.182431e-12 1.204873e-12 5.772013e-11 5.742555e-11 2.211817e-06 1.943190e-09
29 8 9 10 11 12 13
30 3 NA NA NA NA NA NA NA
31 4 NA NA NA NA NA NA NA
32 5. NA NA NA NA NA NA NA
33 6 NA NA NA NA NA NA NA
34 7 NA NA NA NA NA NA NA
35 8 NA NA NA NA NA NA NA
36 9 5.275003e-02 NA NA NA NA NA NA
37 10 4.003725e-02 0.816158979 NA NA NA NA NA
38 11 4.066763e-05 0.004908162 0.01050717 NA NA NA NA
39 12 7.157985e-04 0.062558202 0.10263574 0.2892071 NA NA
40 13 6.473481e-04 0.056812262 0.10260428 0.2970199 1.000000000 NA
41 14 1.270509e-05 0.001369638 0.00278692 0.3337107 0.08152297 0.08152297
42 c$p.value < 0.05
43 2 3 4 5 6 7 8 9 10 11 12 13
44 3 TRUE NA NA
45 4 TRUE TRUE NA NA NA NA NA NA NA NA NA NA
46 5 TRUE TRUE TRUE NA NA NA NA NA NA NA NA NA
47 6 TRUE TRUE TRUE FALSE NA NA NA NA NA NA NA NA
48 7 TRUE TRUE TRUE FALSE FALSE NA NA NA NA NA NA NA
49 8 TRUE TRUE TRUE TRUE FALSE TRUE NA NA NA NA NA NA
50 9 TRUE TRUE TRUE TRUE TRUE TRUE FALSE NA NA NA NA NA
51 10 TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE NA NA NA NA
52 11 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE NA NA NA NA
53 12 TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
54 13 TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE NA
55 14 TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE

```

3.1. Reto 1

Para el primer reto se creó un código en el cual se seleccionó un subconjunto que permita observar el comportamiento del tamaño del porcentaje del frente de Pareto y este actuó de modo diverso, ocasionando una agrupación en una sola zona, cercana a las soluciones.

```

1 cluster <- makeCluster(detectCores()-1)
2 clusterExport(cluster, c("domin.by", "sign", "eval", "poli", "n", "pick.one"))
3
4 dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
5 Select <- frente[c(round(runif(round(dim(frente)[1]/2), min = 1, max = dim(frente)
6 [1]))),]
7 Select1 <- data.frame()
8 Select1 <- rbind(Select1, Select)
9
10 vec <- seq(1, k, by = 1)
11 for (i in vec) {
12   if ((i+1) == is.element(i+1, vec)*(i+1)){
13     png(paste("p11_frente", k, "_", replicas, "_", i, "-", i+1, ".png", sep=""))
14     xt = paste("Objetivo ", i, " (", cual[minim[i] + 1], ") ", sep = " ")
15     yt = paste("Objetivo ", i+1, " (", cual[minim[i+1] + 1], ") ", sep = " ")
16     plot(val[,i], val[,i+1], xlab=xt, ylab=yt)
17     points(frente[,i], frente[,i+1], col="green", pch=16, cex=1.5)
18     points(Select1[,i], Select1[,i+1], col="red", pch=16, cex=1.5)
19     graphics.off()
20   }
21 }
22 data <- data.frame(pos=rep(0, n), dom=dominadores)

```

En la figura 2 se manifiesta un frente diversificado en atención a lo cual los puntos rojos son los puntos del frente de Pareto y los verdes al frente original. Se puede notar que al aumentar los pasos en la ejecución del código los puntos rojos van disminuyendo, debido a la selección determinada en el código anterior.

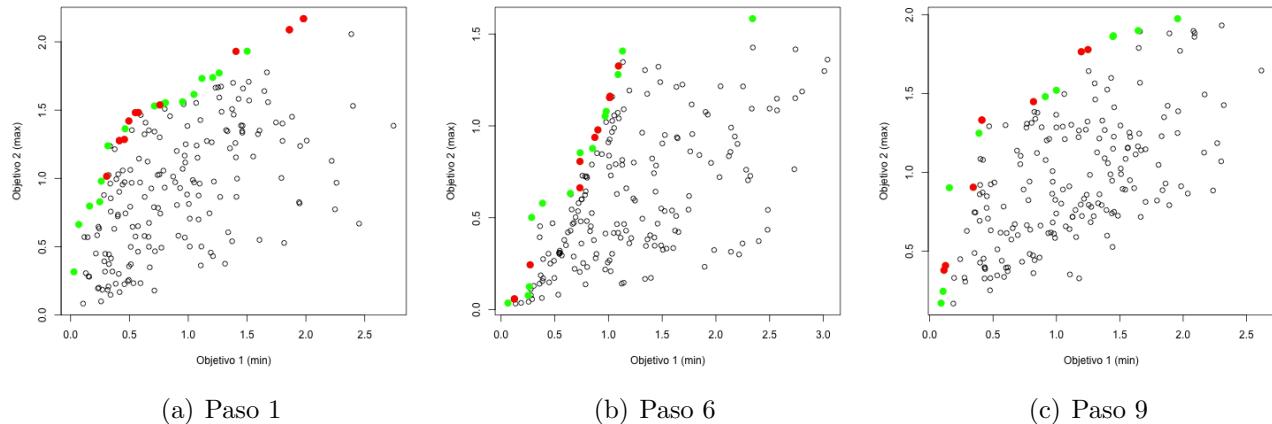


Figura 2: Divergencia de variables

3.2. Reto 2

En el segundo reto se realizó una combinación del código del algoritmo genético [4] con el proporcionado para esta práctica, para realizar una especie de selección natural que permita tener una mejor generación de la población inicial, como se muestra en las siguientes líneas de código.

```
1 conquered <- function(i){  
2   d <- logical()  
3   for (j in 1:n) {  
4     d <- c(d, domin.by(sign * val[i,] , sign * val[j,] , k))  
5   }  
6   return(d)  
7 }  
8  
9 cluster <- makeCluster(detectCores() - 1)  
10 clusterExport(cluster , "verify")  
11 clusterExport(cluster , "eval")  
12 clusterExport(cluster , "obj")  
13 clusterExport(cluster , "sol")  
14 clusterExport(cluster , "tc")  
15 clusterExport(cluster , "k")  
16 clusterExport(cluster , "n")  
17  
18 val <- parSapply(cluster , 1:n, verify)  
19 val <- t(val)  
20 stopCluster(cluster)  
21  
22 mejor1 <- which.max(sign[1] * val[,1])  
23 mejor2 <- which.max(sign[2] * val[,2])  
24 cual <- c("max" , "min")  
25 xl <- paste("Primer objetivo (", cual[minim[1] + 1], ")") , sep="")  
26 yl <- paste("Segundo objetivo (", cual[minim[2] + 1], ")") , sep="")  
27  
28 no.dom <- logical()  
29 conqueredres <- integer()  
30  
31 cluster <- makeCluster(detectCores() - 1)  
32 clusterExport(cluster , "conquered")  
33 clusterExport(cluster , "domin.by")  
34 clusterExport(cluster , "val")  
35 clusterExport(cluster , "sign")  
36 clusterExport(cluster , "k")  
37 clusterExport(cluster , "n")  
38  
39 d <- parSapply(cluster , 1:n, conquered)  
40 stopCluster(cluster)  
41  
42 for(x in 1:nrow(sol)){  
43   cuantos <- sum(d[,x])  
44   conqueredres <- c(conqueredres , cuantos)  
45   no.dom <- c(no.dom, cuantos == 0)  
46   dom <- c(dom, cuantos != 0)  
47 }  
48 frente <- subset(val , no.dom)  
49
```

```

50 mutacion <- function(sol , vc) {
51   pos <- sample(1:vc , 1)
52   mut <- sol
53   delta <- 0.1
54   mut[pos] <- (sol[pos]) * delta
55   return(mut)
56 }
57
58 muta <- function(i){
59   if (runif(1) < pm) {
60     return(mutacion(sol[i , ] , vc))
61   }
62   else{
63     return(sol[i , ])
64   }
65 }
66
67 reproduccion <- function(x, y, vc) {
68   pos <- sample(2:(vc-1) , 1)
69   xy <- c(x[1:pos] , y[(pos+1):vc])
70   yx <- c(y[1:pos] , x[(pos+1):vc])
71   return(c(xy , yx))
72 }
73
74
75 cluster <- makeCluster(detectCores() - 1)
76
77 pm <- 0.05
78 rep <- 50
79 tmax <- 100
80 for (iter in 1:tmax) {
81
82   clusterExport(cluster , "pm")
83   clusterExport(cluster , "vc")
84   clusterExport(cluster , "sol")
85   clusterExport(cluster , "mutacion")
86   clusterExport(cluster , "muta")
87   sol <- t(parSapply(cluster , 1:n, muta))
88
89   for (i in 1:rep) {
90     padres <- sample(1:n, 2, replace=FALSE)
91     hijos <- reproduccion(sol[padres[1],] , sol[padres[2],] , vc)
92     sol <- rbind(sol , hijos[1:vc]) # primer hijo
93     sol <- rbind(sol , hijos[(vc+1):(2*vc)]) # segundo hijo
94   }
95   val <- matrix(rep(NA, k * nrow(sol)) , nrow=nrow(sol) , ncol=k)
96   clusterExport(cluster , "verify")
97   clusterExport(cluster , "eval")
98   clusterExport(cluster , "obj")
99   clusterExport(cluster , "sol")
100  clusterExport(cluster , "tc")
101  clusterExport(cluster , "k")
102  clusterExport(cluster , "n")
103  val <- parSapply(cluster , 1:nrow(sol) , verify)
104  val <- t(val)

```

```

105 no.dom <- logical()
106 dom <- logical()
107 conqueredres <- integer()
108 clusterExport(cluster, "conquered")
109 clusterExport(cluster, "domin.by")
110 clusterExport(cluster, "val")
111 clusterExport(cluster, "sign")
112 clusterExport(cluster, "k")
113 clusterExport(cluster, "n")
114
115 d <- parSapply(cluster, 1:nrow(sol), conquered)
116 for(x in 1:nrow(sol)){
117   cuantos <- sum(d[,x])
118   conqueredres <- c(conqueredres, cuantos)
119   no.dom <- c(no.dom, cuantos == 0)
120   dom <- c(dom, cuantos != 0)
121 }
122 frente_sol <- subset(sol, no.dom)
123 dominadas <- subset(sol, dom)
124 frente <- subset(val, no.dom)
125 domi <- order(conqueredres)
126 domi <- domi[1:n]
127
128 sol <- sol[domi,]
129 val <- val[domi,]
130 digitos <- floor(log(tmax, 10)) + 1
131 t1 <- paste0(iter, "", sep="")
132 while (nchar(t1) < digitos) {
133   t1 <- paste("0", t1, sep="")
134 }
135
136 if(nrow(frente) == n)
137 {
138   break;
139 }
140
141 stopCluster(cluster)
142 system("convert -delay 50 -size 300x300 Genetico*.png -loop 0 Gen.gif")

```

Fueron seleccionadas cinco representaciones gráficas significativas del cambio, que conforman a la figura 3, en la cual se muestra la distribución ocasionada por el algoritmo genético, que en virtud de ello el frente tiene cambios relevantes en las primeras generaciones, después de la octava generación las soluciones se van homogeneizando ligeramente.

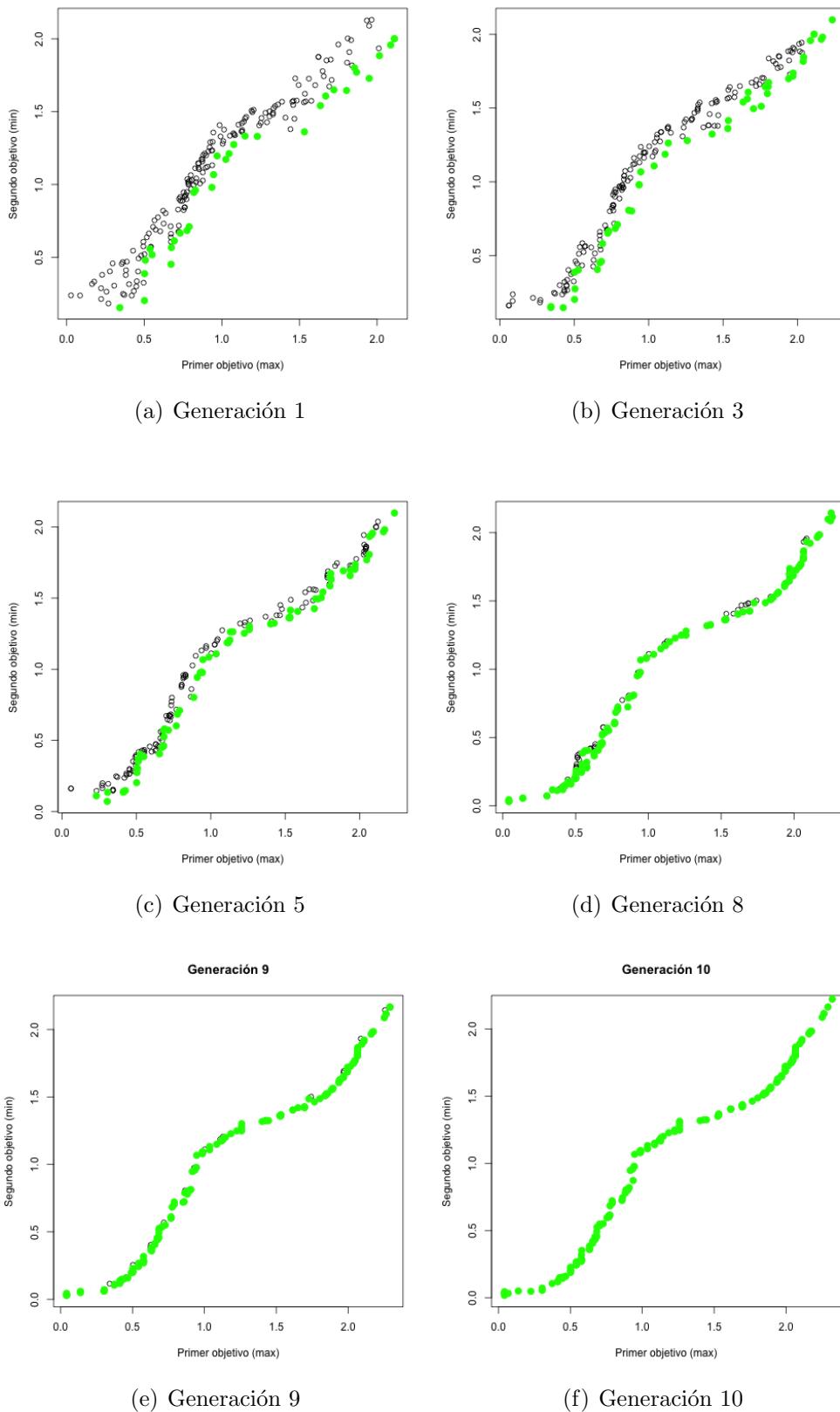


Figura 3: Distribución de algoritmo genético

Referencias

- [1] Yessica Reyna Fernández. Práctica 11, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/files/Practica%2011/>.
- [2] Alboukadel Kassambara. ggplot2 violin plot : Quick start guide - r software and data visualization, 2015. URL <http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-and-data-visualization>.
- [3] Elisa Schaeffer. Práctica 11: frentes de pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.
- [4] Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

Práctica 11: frentes de Pareto

Ricardo Rosas Macías

20 de mayo de 2019

1. Introducción

El análisis frentes de Pareto, permite obtener un resultado mediante la optimización de multi-objetivo, con ayuda de criterios de utilidad; lo cual permite discernir y proporcionar un solución en equilibrio de estas dos entidades, dentro de los márgenes de las variantes.

2. Objetivo

Se realizó cambios en el código del sitio web [3], de modo que proporcione una solución óptima que mejore en un objetivo, descartando eficazmente las demás opciones; de manera que esta sea dejada en la denominada frontera de Pareto.

2.1. Descripción

La finalidad del experimento es [3]:

“Paralelizar el cálculo y graficar el porcentaje de soluciones de Pareto, como función del número de funciones objetivo con diagramas de violín combinados con diagramas de caja-bigote, verificando que las diferencias observadas sean estadísticamente significativas.

El primer reto es seleccionar un subconjunto (cuyo tamaño como un porcentaje del frente original se proporciona como un parámetro) 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.

El segundo reto es adaptar el algoritmo genético de la práctica anterior para que vaya buscando mejora a un frente; la población inicial es el frente generado en la tarea y se aplique la diversificación del primer reto a cada generación después de los cruzamientos y las mutaciones.”

3. Resultados y conclusiones

En base al trabajo anteriormente reportado [1], se realizó el código que se muestra en la parte inferior. En el cual las primeras líneas, exhiben los parámetros con lo que se ejecutó, asimismo en este se evidencia la parallelización de las treinta réplicas de ejecución para las funciones objetivo de 2–14.

```
1 library(parallel)
2 dat <- data.frame()
3 verify <- function(i){
4   val <- c()
5   for (j in 1:k) {
6     val <- c(val, eval(obj[[j]], sol[i,], tc))
7   }
8   return(val)
9 }
10 prop <- function(i){
11   return(list(poli(vc, md, tc)))
12 }
13 cluster <- makeCluster(detectCores()-1)
14 clusterExport(cluster, c("domin.by", "sign", "eval", "poli", "pick.one"))
15
16 vc <- 4
17 md <- 3
18 tc <- 5
19 funciones <- seq(2, 14, by=1)
20 for (k in funciones) {
21   for (replicas in 1:30) {
22     obj <- list()
23     clusterExport(cluster, c("vc", "md", "tc"))
24     obj <- parSapply(cluster, 1:k, prop)
25     minim <- (runif(k) > 0.5)
26     sign <- (1 + -2 * minim)
27     n <- 200 # soluciones aleatorias
28     sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
29     val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
30     clusterExport(cluster, c("tc", "obj", "sol", "eval", "k", "n"))
31     val <- parSapply(cluster, 1:n, verify)
32     val <- t(val)
33
34 dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
35 vec <- seq(1, k, by = 1)
36 for (i in vec) {
37   if((i+1) == is.element(i+1, vec)*(i+1)){
```

Con ayuda de la paquetería *ggplot2* [2] se obtuvo la visualización de código anterior. En la figura [1] evidencia un comportamiento creciente en las soluciones no dominadas dado al incrementar las funciones objetivo.

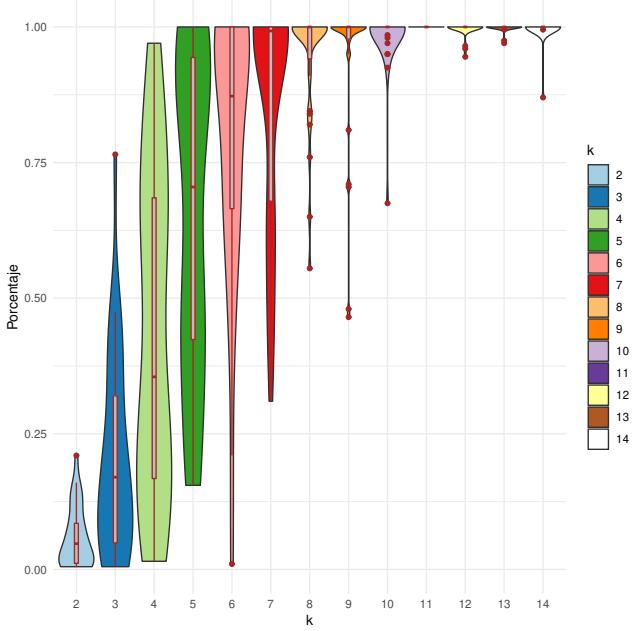


Figura 1: Porcentaje de soluciones

Asimismo, se realizó una prueba *Shapiro-Wilk* para verificar la probabilidad de una distribución normal; es una manera eficiente para determinar la normalidad de las variables. Por otro lado, se hizo una comprobación con *kruskal.test* para evidenciar la existencia de alguna diferencia entre los niveles, además se ejecutó una prueba *Pairwise Wilcox* para presentar que niveles son los causantes de estas diferencias.

```

1 a <- shapiro.test(dat$Porcentaje)
2 a$p.value
3 a$p.value < 0.05
4 b <- kruskal.test(dat$Porcentaje ~ dat$k)
5 b$p.value
6 b$p.value < 0.05
7 c <- pairwise.wilcox.test(dat$Porcentaje, dat$k, p.adjust.method = "none")
8 c$p.value
9 c$p.value < 0.05

```

Por consiguiente, se obtuvieron los resultados de *Shapiro-Wilk test* del código anterior, en donde se puede notar una diferencia en el valor $p < 0,05$, lo cual nos dice que hay diferencias relevantes en las cantidades de funciones objetivo. Por otra parte el valor $p < 0,05$ verificado con el *kruskal.test* nos muestra que solamente en algunos niveles las diferencias son despreciables. Por último con el *Pairwise Wilcox test* se expone con un *TRUE* a los niveles que presentan dichas diferencias.

```

1
2 a <- shapiro.test(dat$Porcentaje)
3 a$p.value
4 [1] 1.030554e-25
5 a$p.value < 0.05

```

```

6 [1] TRUE
7 b <- kruskal.test(dat$Porcentaje ~ dat$k)
8 b$p.value
9 [1] 7.036467e-50
10 b$p.value < 0.05
11 [1] TRUE
12
13 c <- pairwise.wilcox.test(dat$Porcentaje, dat$k, p.adjust.method = "none")
14 c$p.value
15
16      2     3     4     5     6     7     8     9    10    11    12    13
17 3  TRUE   NA   NA
18 4  TRUE  TRUE   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
19 5  TRUE  TRUE  TRUE   NA   NA   NA   NA   NA   NA   NA   NA   NA
20 6  TRUE  TRUE  TRUE  FALSE   NA   NA   NA   NA   NA   NA   NA   NA
21 7  TRUE  TRUE  TRUE  FALSE  FALSE   NA   NA   NA   NA   NA   NA   NA
22 8  TRUE  TRUE  TRUE  TRUE  FALSE  TRUE   NA   NA   NA   NA   NA   NA
23 9  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  FALSE   NA   NA   NA   NA   NA
24 10  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  FALSE   NA   NA   NA   NA
25 11  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE   NA   NA   NA   NA
26 12  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  FALSE
27 13  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  FALSE  FALSE  FALSE   NA
28 14  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  FALSE  FALSE  FALSE
```

3.1. Reto 1

Para el primer reto se creó un código, de acuerdo al trabajo anteriormente reportado [1]. En el cual se seleccionó un subconjunto que permita observar el comportamiento del tamaño del porcentaje del frente de Pareto y este actuó de modo diverso, ocasionando una agrupación en una sola zona, cercana a las soluciones.

```

1 cluster <- makeCluster(detectCores() - 1)
2 clusterExport(cluster, c("domin.by", "sign", "eval", "poli", "n", "pick.one"))
3
4 dat <- rbind(dat, c(k, replicas, sum(no.dom)/n))
5 Select <- frente[c(round(runif(round(dim(frente)[1]/2), min = 1, max = dim(frente)
6 [1]))),]
7 Select1 <- data.frame()
8 Select1 <- rbind(Select1, Select)
9
10 vec <- seq(1, k, by = 1)
11 for (i in vec) {
12   if ((i+1) == is.element(i+1, vec)*(i+1)) {
13     png(paste("p11_frente", k, "_", replicas, "_", i, "_", i+1, ".png", sep=""))
14     xt = paste("Objetivo ", i, " (", cual[minim[i] + 1], ")", sep = " ")
15     yt = paste("Objetivo ", i+1, " (", cual[minim[i+1] + 1], ")", sep = " ")
16     plot(val[,i], val[,i+1], xlab=xt, ylab=yt)
17     points(frente[,i], frente[,i+1], col="green", pch=16, cex=1.5)
18     points(Select1[,i], Select1[,i+1], col="red", pch=16, cex=1.5)
19   }
20   graphics.off()
```

```

19 }
20 }
21 data <- data.frame( pos=rep(0, n), dom=dominadores )

```

En la figura 2 se manifiesta un frente diversificado en atención a lo cual los puntos rojos son los puntos del frente de Pareto y los verdes al frente original. Se puede notar que al aumentar los pasos en la ejecución del código los puntos rojos van disminuyendo, debido a la selección determinada en el código anterior.

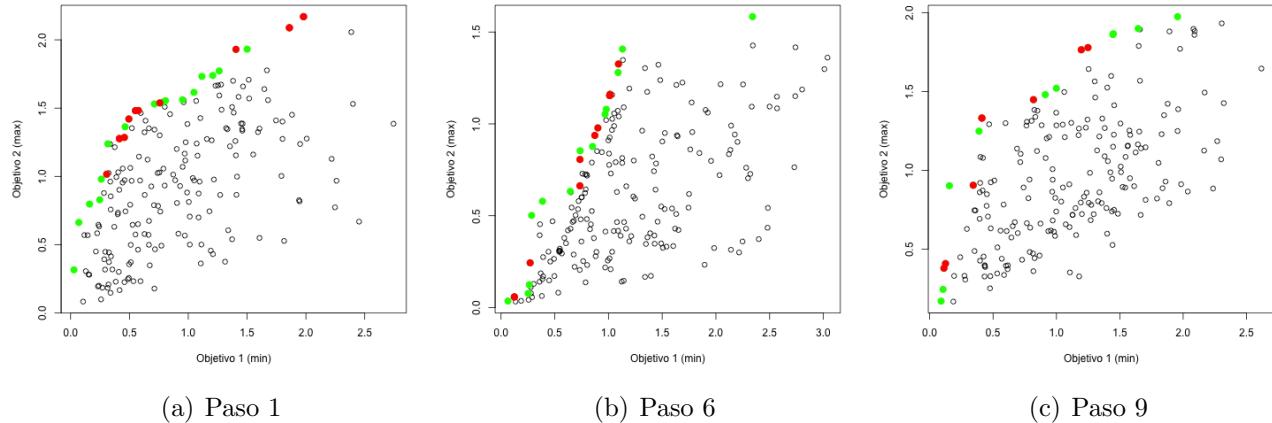


Figura 2: Divergencia de variables

3.2. Reto 2

En el segundo reto se realizó una combinación del código del algoritmo genético [4] con el proporcionado para esta práctica, para realizar una especie de selección natural que permita tener una mejor generación de la población inicial, como se muestra en las siguientes líneas de código.

```

1 conquered <- function(i){
2   d <- logical()
3   for (j in 1:n) {
4     d <- c(d, domin.by(sign * val[i,], sign * val[j,], k))
5   }
6   return(d)
7 }

8
9 cluster <- makeCluster(detectCores() - 1)
10 clusterExport(cluster, "verify")
11 clusterExport(cluster, "eval")
12 clusterExport(cluster, "obj")
13 clusterExport(cluster, "sol")
14 clusterExport(cluster, "tc")
15 clusterExport(cluster, "k")
16 clusterExport(cluster, "n")

```

```

17
18 val <- parSapply(cluster , 1:n, verify)
19 val <- t(val)
20 stopCluster(cluster)
21
22 mejor1 <- which.max(sign[1] * val[,1])
23 mejor2 <- which.max(sign[2] * val[,2])
24 cual <- c("max" , "min")
25 xl <- paste("Primer objetivo (" , cual[minim[1] + 1], ")" , sep="")
26 yl <- paste("Segundo objetivo (" , cual[minim[2] + 1], ")" , sep="")
27
28 no.dom <- logical()
29 conqueredres <- integer()
30
31 cluster <- makeCluster(detectCores() - 1)
32 clusterExport(cluster , "conquered")
33 clusterExport(cluster , "domin.by")
34 clusterExport(cluster , "val")
35 clusterExport(cluster , "sign")
36 clusterExport(cluster , "k")
37 clusterExport(cluster , "n")
38
39 d <- parSapply(cluster , 1:n, conquered)
40 stopCluster(cluster)
41
42 for(x in 1:nrow(sol)){
43   cuantos <- sum(d[,x])
44   conqueredres <- c(conqueredres , cuantos)
45   no.dom <- c(no.dom, cuantos == 0)
46   dom <- c(dom, cuantos != 0)
47 }
48 frente <- subset(val , no.dom)
49
50 mutacion <- function(sol , vc) {
51   pos <- sample(1:vc , 1)
52   mut <- sol
53   delta <- 0.1
54   mut[pos] <- (sol[pos]) * delta
55   return(mut)
56 }
57
58 muta <- function(i){
59   if (runif(1) < pm) {
60     return(mutacion(sol[i , ] , vc))
61   }
62   else{
63     return(sol[i , ])
64   }
65 }
66
67 reproduccion <- function(x, y, vc) {
68   pos <- sample(2:(vc-1) , 1)
69   xy <- c(x[1:pos] , y[(pos+1):vc])
70   yx <- c(y[1:pos] , x[(pos+1):vc])
71   return(c(xy , yx))

```

```

72 }
73
74
75 cluster <- makeCluster(detectCores() - 1)
76
77 pm <- 0.05
78 rep <- 50
79 tmax <- 100
80 for (iter in 1:tmax) {
81
82 clusterExport(cluster, "pm")
83 clusterExport(cluster, "vc")
84 clusterExport(cluster, "sol")
85 clusterExport(cluster, "mutacion")
86 clusterExport(cluster, "muta")
87 sol <- t(parSapply(cluster, 1:n, muta))
88
89 for (i in 1:rep) {
90   padres <- sample(1:n, 2, replace=FALSE)
91   hijos <- reproduccion(sol[padres[1],], sol[padres[2],], vc)
92   sol <- rbind(sol, hijos[1:vc]) # primer hijo
93   sol <- rbind(sol, hijos[(vc+1):(2*vc)]) # segundo hijo
94 }
95 val <- matrix(rep(NA, k * nrow(sol)), nrow=nrow(sol), ncol=k)
96 clusterExport(cluster, "verify")
97 clusterExport(cluster, "eval")
98 clusterExport(cluster, "obj")
99 clusterExport(cluster, "sol")
100 clusterExport(cluster, "tc")
101 clusterExport(cluster, "k")
102 clusterExport(cluster, "n")
103 val <- parSapply(cluster, 1:nrow(sol), verify)
104 val <- t(val)
105 no.dom <- logical()
106 dom <- logical()
107 conqueredres <- integer()
108 clusterExport(cluster, "conquered")
109 clusterExport(cluster, "domin.by")
110 clusterExport(cluster, "val")
111 clusterExport(cluster, "sign")
112 clusterExport(cluster, "k")
113 clusterExport(cluster, "n")
114
115 d <- parSapply(cluster, 1:nrow(sol), conquered)
116 for(x in 1:nrow(sol)){
117   cuantos <- sum(d[,x])
118   conqueredres <- c(conqueredres, cuantos)
119   no.dom <- c(no.dom, cuantos == 0)
120   dom <- c(dom, cuantos != 0)
121 }
122 frente_sol <- subset(sol, no.dom)
123 dominadas <- subset(sol, dom)
124 frente <- subset(val, no.dom)
125 domi <- order(conqueredres)
126 domi <- domi[1:n]

```

```

127
128 sol <- sol[domi,]
129 val <- val[domi,]
130 digitos <- floor(log(tmax, 10)) + 1
131 t1 <- paste0(iter, "", sep="")
132 while (nchar(t1) < digitos) {
133   t1 <- paste("0", t1, sep="")
134 }
135
136 if (nrow(frente) == n)
137 {
138   break;
139 }
140
141 stopCluster(cluster)
142 system("convert -delay 50 -size 300x300 Genetico*.png -loop 0 Gen.gif")

```

Fueron seleccionadas cinco representaciones gráficas significativas del cambio, que conforman a la figura 3, en la cual se muestra la distribución ocasionada por el algoritmo genético, que en virtud de ello el frente tiene cambios relevantes en las primeras generaciones, después de la octava generación las soluciones se van homogeneizando ligeramente.

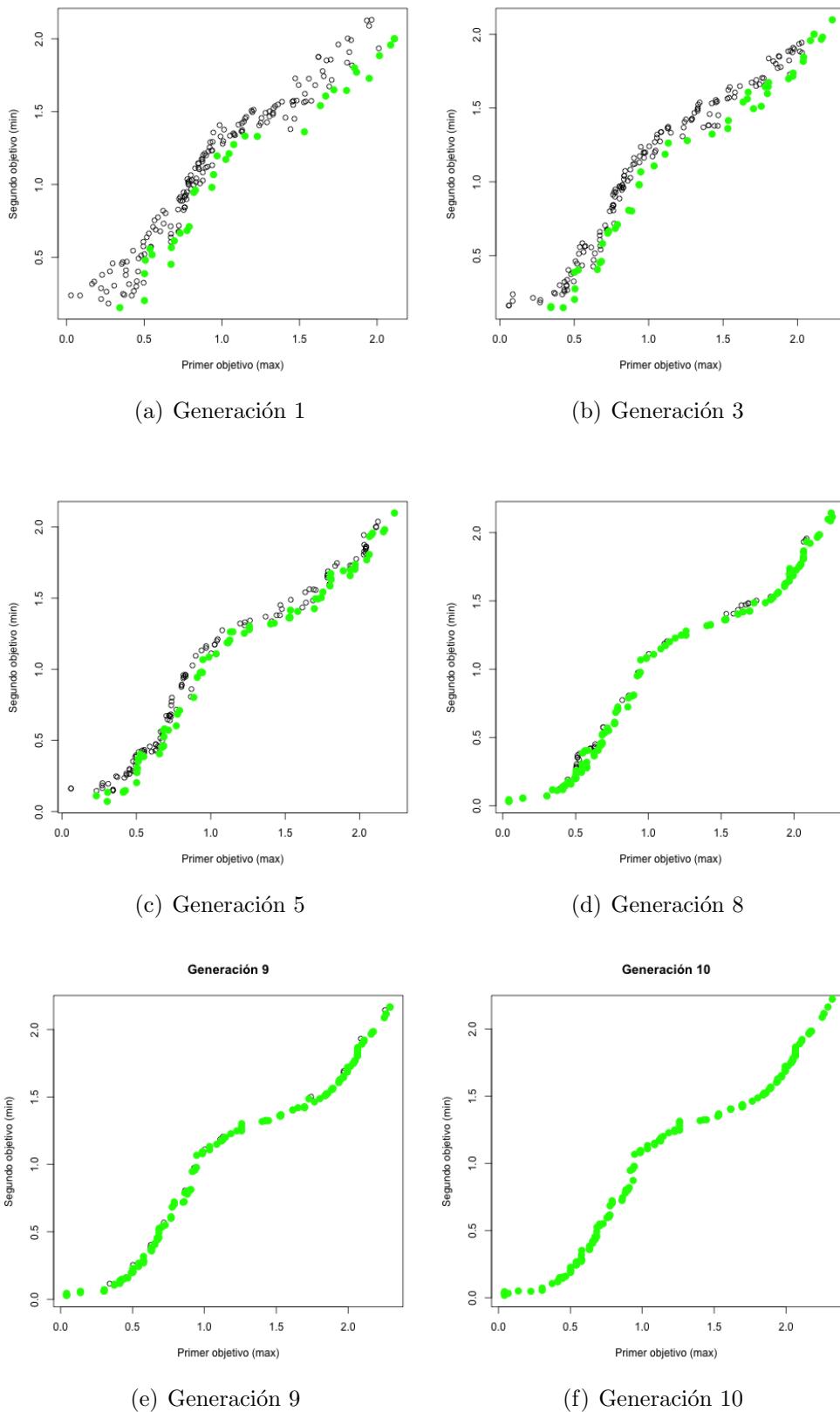


Figura 3: Distribución de algoritmo genético

Referencias

- [1] Yessica Reyna Fernández. Práctica 11, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/files/Practica%2011/>.
- [2] Alboukadel Kassambara. ggplot2 violin plot : Quick start guide - r software and data visualization, 2015. URL <http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-and-data-visualization>.
- [3] Elisa Schaeffer. Práctica 11: frentes de pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.
- [4] Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

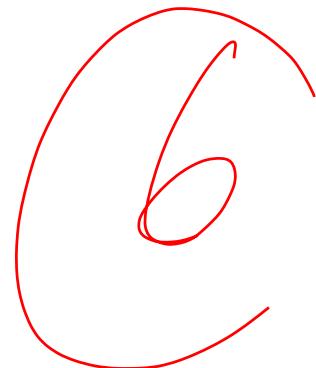
Práctica 12: red neuronal

Ricardo Rosas Macías

21 de mayo de 2019

1. Reporte de aprendizaje

En la décimo segunda práctica tuve problemas para reportar los resultados obtenidos de la simulación, así como un mal uso de recursos visuales; a raíz de un deficiente acomodo en las letras, así como una variación de tamaño. También cabe señalar que el primer documento entregado, carecía de un buen reporte del análisis estadístico, por lo cual en el documento final hice énfasis en los resultados del Anova. De igual manera se realizaron los cambios respectivos que se marcaron en la revisión. En contraste, no pude realizar un código para obtener una plantilla previa del aprendizaje del código, esto debido a la falta de comprensión del tema. Por lo cual en un futuro me gustaría seguir practicando y aprendiendo, para realizar código que ejecute este tipo de fenómenos.



Práctica 12: red neuronal

Ricardo Rosas Macías

5 de mayo de 2019

1. Introducción

La red neuronal es un modelo que funciona de una forma similar y simplificada a la de un cerebro humano, en un ordenador. Este tipo de inteligencia artificial permite obtener un resultado óptimo con la ayuda de criterios de utilidad, a través de la combinación de parámetros y ponderaciones de entrenamiento de selección multi-objetivo; asistidos de perceptrones, que favorecen la identificación de la posición, de modo que favorecen discernir la predicción del hiperplano.

2. Objetivo

Para llevar acabo el objetivo de la práctica, se hizo un proceso de aprendizaje de patrones, con los cuales la red neuronal fuera capaz de crear ponderaciones sinápticas para la elección correcta del resultado. Asimismo, se paralelizó el experimento para obtener un mejor desempeño en los tiempos de ejecución del código.

2.1. Descripción

La finalidad del experimento es [3]:

“Paralelizar un segmento del código y estudiar de manera sistemática el desempeño de la red neuronal para los diez dígitos en función de las tres probabilidades asignadas a la generación de los dígitos (ngb), variando a las tres en un experimento factorial adecuado.

Para el primer reto, se extiende y entrena la red neuronal para que reconozca además por al menos doce símbolos ASCII adicionales, aumentando la resolución de las imágenes a 5×7 de lo original de 3×5 (modificando las plantillas de los dígitos acorde a este cambio).”

3. Resultados y conclusiones

De acuerdo a la literatura anteriormente reportada [1], se hicieron líneas de código que permitieran una ejecución paralela. De igual manera, se cuantificó el tiempo que toma en realizar la ejecución del código y se comparó con la ejecución normal, como se muestra en la parte inferior.

```

1 repetitions <- 30
2 trainNP <- microbenchmark(
3   for (t in 1:5000) {
4     d <- sample(0:tope, 1)
5     pixeles <- runif(dim) < modelos[d + 1,]
6     correcto <- binario(d, n)
7     for (i in 1:n) {
8       w <- neuronas[i,]
9       deseada <- correcto[i]
10      resultado <- sum(w * pixeles) >= 0
11      if (deseada != resultado) {
12        ajuste <- tasa * (deseada - resultado)
13        tasa <- tranqui * tasa
14        neuronas[i,] <- w + ajuste * pixeles
15      }
16    }
17  }
18 , times = repetitions, unit = "s")
19
20 testNP <- microbenchmark({
21   contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
22   rownames(contadores) <- 0:tope
23   colnames(contadores) <- c(0:tope, NA)
24
25   for (t in 1:300) {
26     d <- sample(0:tope, 1)
27     pixeles <- runif(dim) < modelos[d + 1,]
28     correcto <- binario(d, n)
29     salida <- rep(FALSE, n)
30     for (i in 1:n) {
31       w <- neuronas[i,]
32       deseada <- correcto[i]
33       resultado <- sum(w * pixeles) >= 0
34       salida[i] <- resultado
35     }
36     r <- min(decimal(salida, n), k)
37     contadores[d+1, r+1] <- contadores[d+1, r+1] + 1
38   }
39 }
40 , times = repetitions, unit = "s")
41
42 newrons <- function(i, neuronas, pixeles){
43   w <- neuronas[i,]
44   return(sum(w * pixeles) >= 0)
45 }
46 newr <- function(){
47   d <- sample(0:tope, 1)
48   pixeles <- runif(dim) < modelos[d + 1,]
49   correcto <- binario(d, n)
50
51   salida <- sapply(1:n, newrons, neuronas, pixeles)
52   r <- min(decimal(salida, n), k)
53   return(c(d+1, r+1))
54 }

```

```

55 suppressMessages(library(doParallel))
56 clust <- makeCluster(detectCores()-1)
57 registerDoParallel(clust)
58
59 testP <- microbenchmark({
60   contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
61   rownames(contadores) <- 0:tope
62   colnames(contadores) <- c(0:tope, NA)
63   conta <- foreach(t = 1:300, .combine = "rbind", .export = c("tope", "dim", "modelos",
64     "neuronas", "pixeles")) %dopar% newr()
65   for (i in 1:300){
66     contadores[conta[i, 1], conta[i, 2]] <- contadores[conta[i, 1], conta[i, 2]] + 1
67   }
68 }, times = repetitions, unit = "s")
69
70 stopCluster(clust)
71 trains <- cbind("Normal" = trainNP$time)
72 tests <- cbind("Paralelizada" = testP$time, "Normal" = testNP$time)

```

Se realizó una visualización gráfica con ayuda de la paquetería *ggplot2*, como se muestra en la figura 1, en la cual se observa claramente un aumento de tiempos al realizar la ejecución normal, de modo que permite definir que la ejecución paralela es mejor para el experimento.

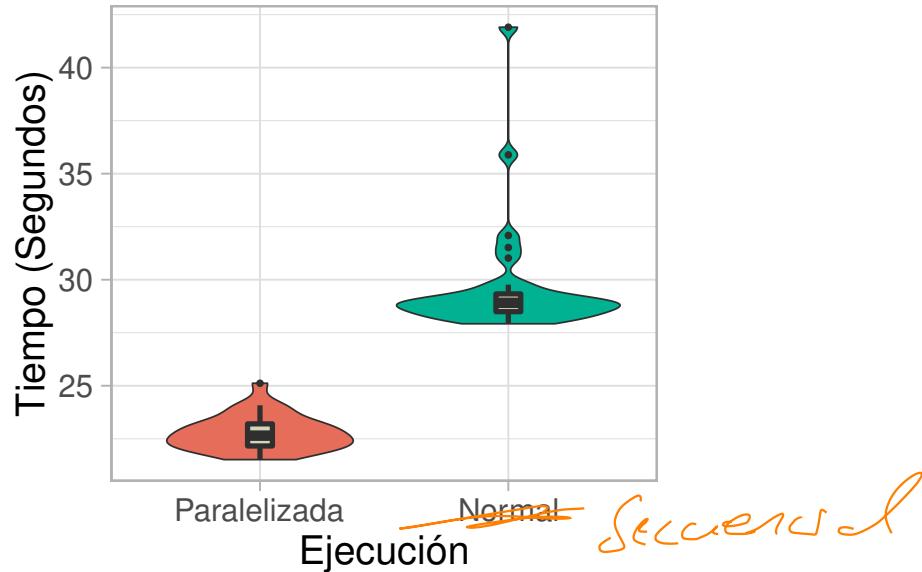


Figura 1: Tiempo de tarda la ejecución del código

Acorde al precedente [1], se hicieron líneas de código que proporcionen el desempeño de la red neuronal para los diez dígitos. Por ende, en el código inferior, se determinó una variación de las probabilidades de color los pixeles; de manera que permita observar el desempeño de la red neuronal a estos cambios.

```

1 newrons <- function(i, neuronas, pixeles){
2   w <- neuronas[i,]

```

```

3   return (sum(w * pixeles) >= 0)
4 }
5 newr <- function(){
6   d <- sample(0:tope, 1)
7   pixeles <- runif(dim) < modelos[d + 1,]
8   correcto <- binario(d, n)
9   salida <- sapply(1:n, newrons, neuronas, pixeles)
10  r <- min(decimal(salida, n), k)
11  return (c(d+1, r+1))
12 }
13
14 Negros <- c(0.90, 0.95, 0.995)
15 Grises <- c(0.80, 0.90, 0.992)
16 Blancos <- c(0.002, 0.1, 0.5)
17
18 data <- data.frame("Porcen"=numeric(), "N" = numeric(), "G" = numeric(), "B"= numeric())
19 for (negro in Negros){
20   for (gris in Grises) {
21     for (blanco in Blancos){
22
23       modelos <- read.csv("digitos.modelo.csv", sep=" ", header=FALSE, stringsAsFactors=F)
24       modelos[modelos=='n'] <- negro
25       modelos[modelos=='g'] <- gris
26       modelos[modelos=='b'] <- blanco
27
28       suppressMessages(library(doParallel))
29       clust <- makeCluster(detectCores()-1)
30       registerDoParallel(clust)
31
32       contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
33       rownames(contadores) <- 0:tope
34       colnames(contadores) <- c(0:tope, NA)
35       conta <- foreach(t = 1:300, .combine = "rbind", .export = c("tope", "dim", "
36         modelos", "neuronas", "pixeles")) %dopar% newr()
37       for (i in 1:300){
38         contadores[conta[i, 1], conta[i, 2]] <- contadores[conta[i, 1], conta[i, 2]] + 1
39       }
40       stopCluster(clust)
41
42       atino <- numeric()
43       noatino <- numeric()
44       for (i in 1:(dim(contadores)[1])){
45         noatino[i] <- 0
46         for (j in 1:(dim(contadores)[2])){
47           if(i == j){
48             atino[i] <- contadores[i, j]
49           } else {
50             noatino[i] <- sum(noatino[i], contadores[i, j])
51           }
52         }
53       }
54       porcen <- numeric()
55       for (i in 1:length(atino)){
56         porcen[i] <- noatino[i]/(atino[i]+noatino[i])
57         por <- c(porcen[i], negro, gris, blanco)
58       }
59     }
60   }
61 }

```

```

57 } data <- rbind(data , por)
58 }
59 }
60 }
61 }

```

Por consiguiente, se obtuvieron los resultados del código anterior gráficamente, en la figura 2, en donde se puede observar como un gran porcentaje de error de cada dígito se encuentra muy cerca de la normalidad. Por lo cual, se procedió a realizar un análisis de varianza; para obtener información más detallada, cómo se muestra en el cuadro 1, en el cual se aprecia un mayor efecto en las probabilidades de los píxeles negros y blancos, en atención a lo cual los pixeles blancos cuentan con mayor relevancia, debido a la mayor influencia de clasificación.

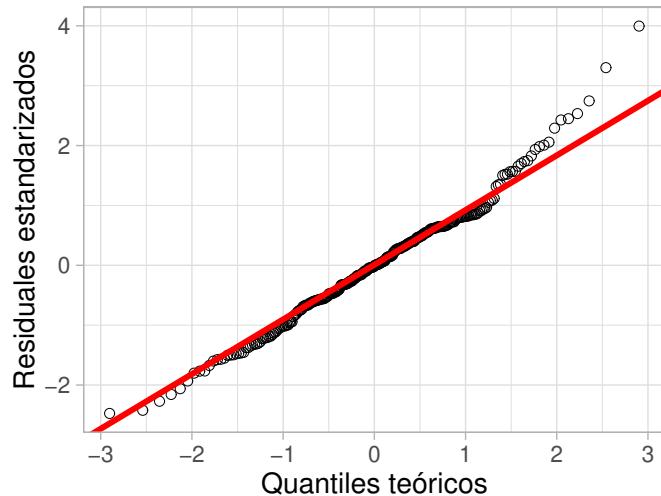


Figura 2: Evaluación de normalidad

Cuadro 1: Resultados de análisis de varianza

	Df	Sum Sq	Mean Sq	F Value	Pr(>F)
Negro	2	1.27	0.63	16.05	0.0000
Gris	2	0.63	0.31	7.96	0.0004
Blanco	2	12.18	6.09	154.15	0.0000
Negro:Gris	4	0.19	0.05	1.22	0.3047
Negro:Blanco	4	0.31	0.08	1.95	0.1036
Gris:Blanco	4	0.11	0.03	0.70	0.5960
Negro:Gris:Blanco	8	0.15	0.02	0.48	0.8729
Residuals	243	9.60	0.04		

B

3.1. Reto 1

Con ayuda de la literatura [2] se realizaron las líneas de código que se muestra en la parte inferior, en las cuales con la ayuda de la codificación de color para cada pixel que se observa en el cuadro 2 permitió realizar una resolución de imagen de 5×7 y reconocimiento de símbolos ASCII.

Cuadro 2: Codificación de colores de la plantilla (digitosext.modelo.csv)

```
n g n g n g b b b g n b b b b n g b b b g n b b b n g b b b g n g n g n  
n n g b b b b n b b b b n b b b b g b b b b n b b b n b b n n g n n  
n g g n g b b b b g b b b b n g g n g g n b b b b g b b b b g n g g n  
g n g n g b b b b n b b b b g g n g n n b b b b g b b b b n g n g n  
g g b g g n n b n n g g b g g n n g n n b b b b g g b b b n n b b b g  
g n n g g b b b b n b b b b n g g n n b b b b g b b b b g n g g n n  
g n g n g n b b b b g b b b b n g n g n g b b b b g n b b b b n g n g n  
n g n g n b b b b g b b b b n b b g n g b b b b n b b b b g b b b b n  
n g n g n g b b b b n b b b b n g n g n g b b b b n g b b b b g n g n g n  
g n g n g n b b b b n g b b b g n g n g n b b b b g b b b b n b b b g  
n n b b b g g b b b b n n b b b g g b b b b n n b b b g g g g g n n n n  
n g n g n g n g n g b b b b n b b b b g b b b b n b b b g n g n g n g n  
n g n g n g n g n g n g b b b g n b b b b n g b b b g n g n g n g n g n  
g g b g g n n b n n g g b g g n n n n g g b g g n n b n n g g b g g  
g n g n g n g n g n b b g b b b b n b b b g b b b b n b b b b g b b b  
n g n g n g n g n g n g b b b g n g n b b b g b b b g n g n g n g n g n  
n n n n n n n n n n b b b b n n n b b b b n n b b b b n n b b b b  
n n b n n g g b g g n n b n n g g b g g n n b n n g g g g g n n n n  
g n g n g n b b b b n g b b b g n g n g n g b b b b n b b b b g b b b  
n g n g n n b b b b n g b b b g n n g n g b b b b g n b b b b n n b b  
n b b b n g b b b g n b g b n g b g b n b g b n b g b n g g n g g  
g n g n g n b n b n g b g b g n b n b n g b g b g n b b b b n g b b b g
```

```
1 modelos <- read.csv("digitosext.modelo.csv", sep=" ", header=FALSE, stringsAsFactors=F)  
2  
3 r <- 7  
4 c <- 5  
5 dim <- r * c  
6  
7 n <- 49  
8 w <- ceiling(sqrt(n))  
9 h <- ceiling(n / w)  
10  
11 SASCHIAD <- c(0:9, "U", "L", "I", "C", "H", "M", "T", "P", "A", "W", "E", "F", )  
12  
13 png("plantilla.png", width=1600, height=2000)  
14 par(mfrow=c(w, h), mar = c(0,0,7,0))  
15 suppressMessages(library("sna"))  
16  
17 for (j in 1:n) {  
18   d <- sample(0:21, 1)
```

```

19  pixeles <- runif(dim) < modelos[d + 1,]
20  imagen <- matrix(pixeles, nrow=r, ncol=c, byrow=TRUE)
21  plot.sociomatrix(imagen, drawlab=FALSE, diaglab=FALSE,
22                    main=paste(SASCIAD[d+1], " "), cex.main=5)
23 }
24 graphics.off()

```

En la figura 3 se observa la plantilla con los números 0–9 y con los símbolos ASCII. De tal forma que se puede concluir que el aprendizaje de la red neuronal es exitoso con el código anterior.

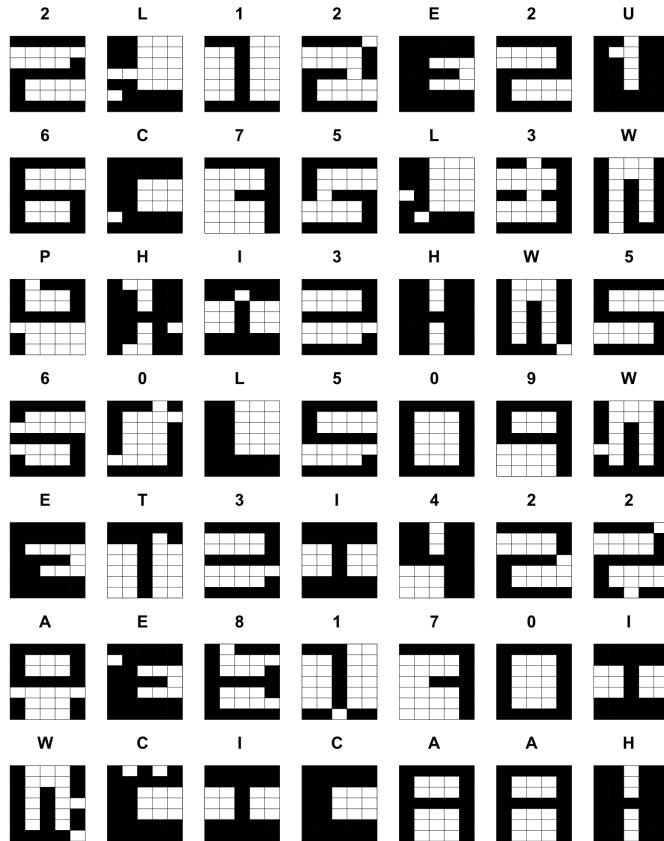


Figura 3: Plantilla con símbolos ASCII adicionales de 5×7

Referencias

- [1] Astrid González. Red neuronal, 2018. URL https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_12/.
- [2] Liliana Saus. Red neuronal, 2018. URL <https://github.com/pejli/simulacion/tree/master/P12>.
- [3] Elisa Schaeffer. Práctica 12: red neuronal, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p12.html>.

Práctica 12: red neuronal

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

La red neuronal es un modelo que funciona de una forma similar y simplificada a la de un cerebro humano, en un ordenador. Este tipo de inteligencia artificial permite obtener una solución con la ayuda de criterios de utilidad, a través de la combinación de parámetros y ponderaciones de entrenamiento de selección multi-objetivo; asistidos de perceptrones, que favorecen la identificación de la posición, de modo que favorecen discernir la predicción del hiperplano.

2. Objetivo

Para llevar acabo el objetivo de la práctica, se hizo un proceso de aprendizaje de patrones, con los cuales la red neuronal fuera capaz de crear ponderaciones sinápticas para la elección correcta del resultado. Asimismo, se paralelizó el experimento para obtener un mejor desempeño en los tiempos de ejecución del código.

2.1. Descripción

La finalidad del experimento es [3]:

“Paralelizar un segmento del código y estudiar de manera sistemática el desempeño de la red neuronal para los diez dígitos en función de las tres probabilidades asignadas a la generación de los dígitos (ngb), variando a las tres en un experimento factorial adecuado.

Para el primer reto, se extiende y entrena la red neuronal para que reconozca además por al menos doce símbolos ASCII adicionales, aumentando la resolución de las imágenes a 5×7 de lo original de 3×5 (modificando las plantillas de los dígitos acorde a este cambio).”

3. Resultados y conclusiones

De acuerdo a la literatura anteriormente reportada [1], se hicieron líneas de código que permitieran una ejecución paralela. De igual manera, se cuantificó el tiempo que toma en realizar la ejecución del código y se comparó con la ejecución normal, como se muestra en la parte inferior.

```

1 repetitions <- 30
2 trainNP <- microbenchmark(
3   for (t in 1:5000) {
4     d <- sample(0:tope, 1)
5     pixeles <- runif(dim) < modelos[d + 1,]
6     correcto <- binario(d, n)
7     for (i in 1:n) {
8       w <- neuronas[i,]
9       deseada <- correcto[i]
10      resultado <- sum(w * pixeles) >= 0
11      if (deseada != resultado) {
12        ajuste <- tasa * (deseada - resultado)
13        tasa <- tranqui * tasa
14        neuronas[i,] <- w + ajuste * pixeles
15      }
16    }
17  }
18 , times = repetitions, unit = "s")
19
20 testNP <- microbenchmark({
21   contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
22   rownames(contadores) <- 0:tope
23   colnames(contadores) <- c(0:tope, NA)
24
25   for (t in 1:300) {
26     d <- sample(0:tope, 1)
27     pixeles <- runif(dim) < modelos[d + 1,]
28     correcto <- binario(d, n)
29     salida <- rep(FALSE, n)
30     for (i in 1:n) {
31       w <- neuronas[i,]
32       deseada <- correcto[i]
33       resultado <- sum(w * pixeles) >= 0
34       salida[i] <- resultado
35     }
36     r <- min(decimal(salida, n), k)
37     contadores[d+1, r+1] <- contadores[d+1, r+1] + 1
38   }
39 }
40 , times = repetitions, unit = "s")
41
42 newrons <- function(i, neuronas, pixeles)
43 newr <- function(){
44   d <- sample(0:tope, 1)
45   pixeles <- runif(dim) < modelos[d + 1,]
46
47   salida <- sapply(1:n, neuronas[i,])
48   r <- min(decimal(salida, n), k)
49   return(c(d+1, r+1))
50 }
51 suppressMessages(library(doParallel))
52 clust <- makeCluster(detectCores()-1)
53 registerDoParallel(clust)
54

```

```

55 testP <- microbenchmark({
56   contadores <- matrix( rep(0, k*(k+1)) , nrow=k, ncol=(k+1))
57   rownames(contadores) <- 0:tope
58   colnames(contadores) <- c(0:tope, NA)
59   conta <- foreach(t = 1:300, .combine = "rbind", .export = c("tope", "dim", "modelos",
60     "neuronas", "pixeles")) %dopar% newr()
61   for (i in 1:300){
62     contadores[conta[i, 1], conta[i, 2]] <- contadores[conta[i, 1], conta[i, 2]] + 1
63   }
64   , times = repetitions, unit = "s")
65
66 stopCluster(clust)
67 trains <- cbind("Normal" = trainNP$time)
68 tests <- cbind("Paralelizada" = testP$time, "Normal" = testNP$time)

```

Se realizó una visualización gráfica con ayuda de la paquetería *ggplot2*, como se muestra en la figura 1, en la cual se observa claramente un aumento de tiempos al realizar la ejecución normal, de modo que permite definir que la ejecución paralela es mejor para el experimento.

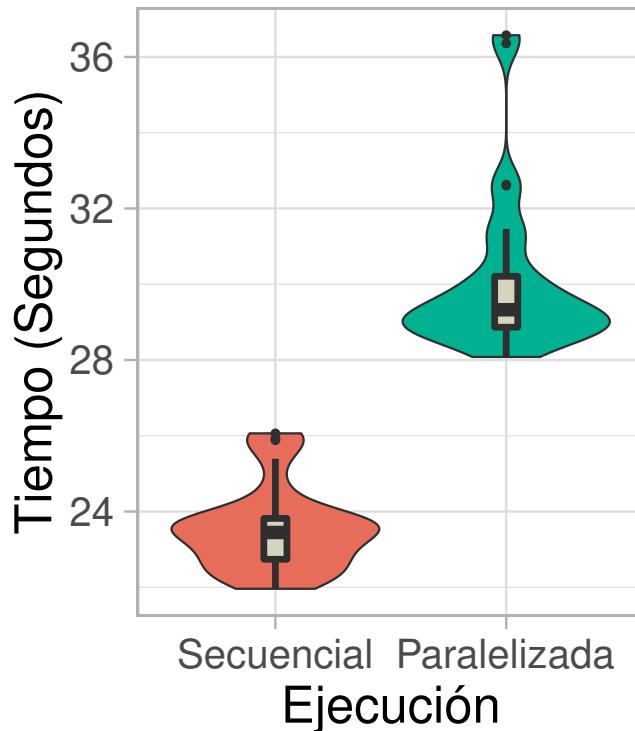


Figura 1: Tiempo de tarda la ejecución del código

Acorde al precedente [1], se hicieron líneas de código que proporcionen el desempeño de la red neuronal para los diez dígitos. Por ende, en el código inferior, se determinó una variación de las probabilidades de color los pixeles; de manera que permita observar el desempeño de la red neuronal a estos cambios.

```

1 newrons <- function(i, neuronas, pixeles){
2   w <- neuronas[i,]
3   return(sum(w * pixeles) >= 0)
4 }
5 newr <- function(){
6   d <- sample(0:tope, 1)
7   pixeles <- runif(dim) <- modelos[d + 1,]
8   correcto <- binario(d, n)
9   salida <- sapply(1:n, newrons, neuronas, pixeles)
10  r <- min(decimal(salida, n), k)
11  return(c(d+1, r+1))
12 }
13
14 Negros <- c(0.90, 0.95, 0.995)
15 Grises <- c(0.80, 0.90, 0.992)
16 Blancos <- c(0.002, 0.02, 0.1)
17
18 data <- data.frame("Porcen"=numeric(), "N" = numeric(), "G" = numeric(), "B"= numeric())
19 for (negro in Negros){
20   for (gris in Grises) {
21     for (blanco in Blancos){
22
23     modelos <- read.csv("digitos.modelo.csv", sep=" ", header=FALSE, stringsAsFactors=F)
24     modelos[modelos=="n"] <- negro
25     modelos[modelos=="g"] <- gris
26     modelos[modelos=="b"] <- blanco
27
28     suppressMessages(library(doParallel))
29     clust <- makeCluster(detectCores()-1)
30     registerDoParallel(clust)
31
32     contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
33     rownames(contadores) <- 0:tope
34     colnames(contadores) <- c(0:tope, NA)
35     conta <- foreach(t = 1:300, .combine = "rbind", .export = c("tope", "dim", "
36       modelos", "neuronas", "pixeles")) %dopar% newr()
37     for (i in 1:300){
38       contadores[conta[i, 1], conta[i, 2]] <- contadores[conta[i, 1], conta[i, 2]] + 1
39     }
40     stopCluster(clust)
41
42     atino <- numeric()
43     noatino <- numeric()
44     for (i in 1:(dim(contadores)[1])){
45       noatino[i] <- 0
46       for (j in 1:(dim(contadores)[2])){
47         if(i == j){
48           atino[i] <- contadores[i, j]
49         } else {
50           noatino[i] <- sum(noatino[i], contadores[i, j])
51         }
52       }
53     }
54     porcen <- numeric()
55     for (i in 1:length(atino)){

```

```

55 porcen [ i ] <- noatino [ i ] / ( atino [ i ]+noatino [ i ] )
56 por <- c( porcen [ i ] , negro , gris , blanco )
57 data <- rbind( data , por )
58 }
59 }
60 }
61 }

```

Por consiguiente, se obtuvieron los resultados del código anterior gráficamente, en la figura 2, en donde se puede observar como un gran porcentaje de error de cada dígito se encuentra muy cerca de la normalidad. Por lo cual, se procedió a realizar un análisis de varianza; para obtener información más detallada, cómo se muestra en el cuadro 1, en el cual se aprecia un mayor efecto en las probabilidades de los píxeles negros y blancos, en atención a lo cual los píxeles blancos cuentan con mayor relevancia, debido a la mayor influencia de clasificación.

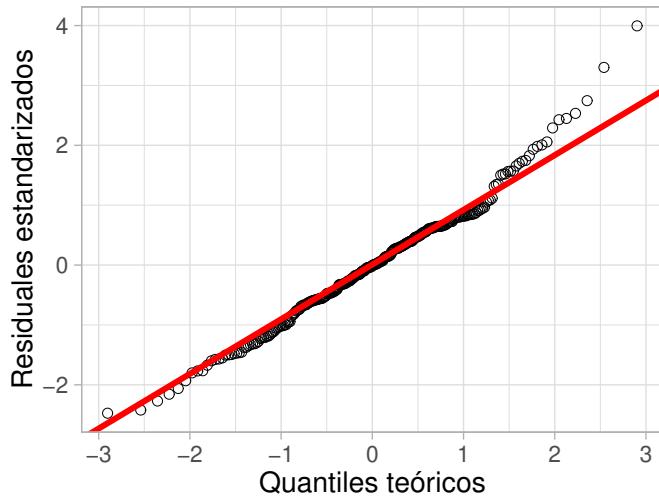


Figura 2: Evaluación de normalidad

Por otro lado, en el cuadro 1 se observa una diferencia estadísticamente significativa en el “Valor F” de 19.408 y 172.176 los píxeles Negro y blanco respectivamente; en comparación al gris, de manera que cuentan con mayores grados de libertad como se muestra en la columna “Df”, así como con un valor “Pr” por debajo de 0.05, que indica una menor desviación estándar. De modo que se puede concluir que hay un incremento en la probabilidad de presencia en los píxeles Blancos.

Cuadro 1: Resultados de análisis de varianza

	Df	Sum Sq	Media Sq	valor F	Pr(>F)
Negro	2	1.435	0.718	19.408	1.52e-08
Gris	2	0.502	0.251	6.792	0.00135
Blanco	2	12.732	6.366	172.176	<2e-16
Negro:Gris	4	0.020	0.005	0.138	0.96800
Negro:Blanco	4	0.245	0.061	1.657	0.16067
Gris:Blanco	4	0.678	0.170	4.586	0.00138
Negro:Gris:Blanco	8	0.183	0.023	0.617	0.76287
Residuals	243	8.984	0.037		

3.1. Reto 1

Con ayuda de la literatura [2] se realizó un documento CSV que se muestra en la parte inferior, en las cuales con la ayuda de la codificación de color para cada pixel, permitió realizar una resolución de imagen de 5×7 y reconocimiento de símbolos ASCII.

```

1 n g n g n g b b b g n b b b n g b b b g n b b b n g b b b g n g n
2 n n g b b b b n b b b b n b b g b b b b n b b b b n b b n n g n n
3 n g g n g b b b b g b b b b n g g n g g n b b b b g b b b b g n g g n
4 g n g n g b b b b n b b b b g g n g n n b b b b g b b b b n g n g n g
5 g g b g g n n b n n g g b g g n n g n n b b b b g g b b b b n n b b g g
6 g n n g g g b b b b n b b b b n g g n n b b b b g g b b b b g n g g n n
7 g n g n g n b b b b g b b b b n g n g n g b b b g n b b b b n g n g n g
8 n g n g n b b b b g b b b b n b b g n g b b b b n b b b b g b b b b n
9 n g n g n g b b b g n b b b b n g n g n g n b b b b n g b b b g n g n g n
10 g n g n g n b b b b n g b b b g n g n g n b b b b g g b b b b n b b b b g
11 n n b b b g g b b b b n n b b g g b b b n n b b g g g g g n n n n n n
12 n g n g n g n g n g b b n b b b b g b b b b n b b g g n g n g n g n g n
13 n g n g n g n g n g n g b b b b g n b b b b n g b b b g g n g n g n g n g
14 g g b g g n n b n n n g g b g g n n n n g g b g g g n n b n n n g g b g g
15 g n g n g n g n g n b b g b b b b n b b b b g b b b b n b b b b g b b
16 n g n g n g n g n g b b b g n g n b n g b b b b g g n g n g n g n g n
17 n n n n n n n n n n b b b n n n b b n n b b n n b b n n b b n n b b b
18 n n b n n g g b g g n n b n n g g b g g n n b n n g g g g g n n n n n n
19 g n g n g n b b b b n g b b b g n g n g n g b b b b n b b b g b b b b
20 n g n g n n b b b b n g b b b g n n g n n g b b b g n b b b b n n b b b n
21 n b b b n g b b b g n b g b n g b n b g g b n b g n b g b n g g n g g
22 g n g n g n b n b n g b g b g n b n b n g b g b g n b b n g b b
```

```

1 modelos <- read.csv("digitosext.modelo.csv", sep=" ", header=FALSE, stringsAsFactors=F)
2
3 r <- 7
4 c <- 5
5 dim <- r * c
6
7 n <- 49
8 w <- ceiling(sqrt(n))
9 h <- ceiling(n / w)
```

```

10 SASCIAD <- c(0:9 , "U" , "L" , "I" , "C" , "H" , "M" , "T" , "P" , "A" , "W" , "E" , "F" , )
11
12 png("plantilla.png" , width=1600, height=2000)
13 par(mfrow=c(w, h) , mar = c(0,0,7,0))
14 suppressMessages(library("sna"))
15
16
17 for (j in 1:n) {
18   d <- sample(0:21, 1)
19   pixeles <- runif(dim) < modelos[d + 1,]
20   imagen <- matrix(pixeles, nrow=r, ncol=c, byrow=TRUE)
21   plot.sociomatrix(imagen, drawlab=FALSE, diaglab=FALSE,
22                     main=paste(SASCIAD[d+1], ""))
23 }
24 graphics.off()

```

En la figura 3 se observa la plantilla con los números 0–9 y con los símbolos ASCII, generada con la ejecución del código.

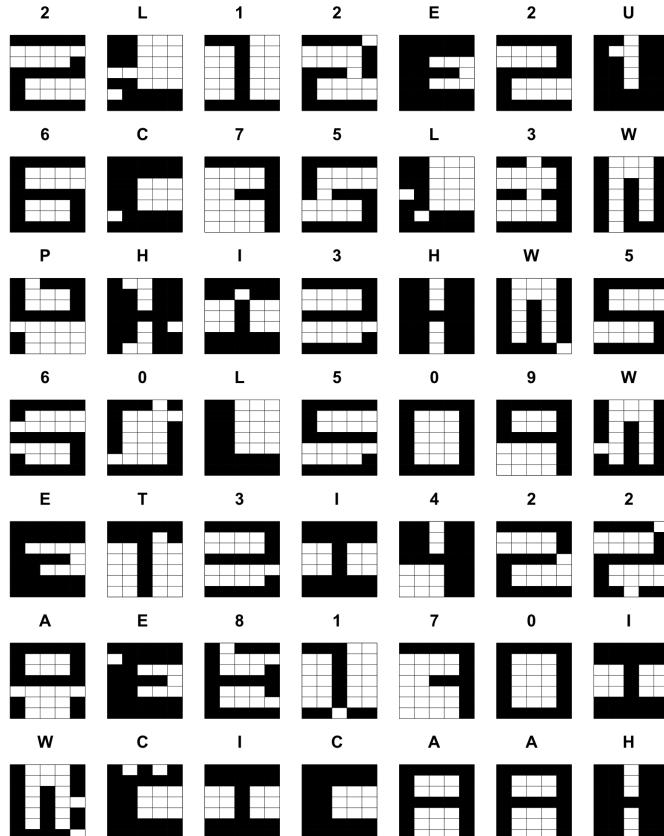


Figura 3: Plantilla con símbolos ASCII adicionales de 5×7

Referencias

- [1] Astrid González. Red neuronal, 2018. URL https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_12/.
- [2] Liliana Saus. Red neuronal, 2018. URL <https://github.com/pejli/simulacion/tree/master/P12>.
- [3] Elisa Schaeffer. Práctica 12: red neuronal, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p12.html>.