



UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME

MAESTRÍA EN CIENCIAS DE LA INGENIERÍA CON
ORIENTACIÓN EN NANOTECNOLOGÍA

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Portafolio tareas y proyecto final

ALUMNO:

Abraham Azael Morales Juárez

1422745

INVESTIGADORA:

Dra. Satu Elisa Schaeffer

Práctica 1: Movimiento Browniano

Abraham Azael Morales Juárez

REPORTE:

En el primer reporte que se realizó también fue la primera vez que usaba estos programas o que veía algo parecido a esto, tuve muchos problemas porque no sabía cómo usar o noción de cómo se usarlos. Al principio intentaba imaginar lo que hacía el código de la práctica en mi mente para comprenderla mejor pero aun así no sabía cómo esas líneas de código harían el trabajo que nos decían que hacían, tarde en asimilar esta nueva forma de ver las cosas, para mí fue algo completamente nuevo.

Conocí el programa de LATEX y R studio, me gustó mucho el LATEX por su diversidad de letra y la habilidad que se tiene para acomodar los datos de la manera que se guste.

El programa R studio es de mucha utilidad para realizar análisis de datos y llevar acabo simulaciones para poder observar comportamientos, además, conforme se iba avanzando con las prácticas se obtuvo más experiencia para poder realizarlas.

3

Movimiento Browniano

Abraham Azael Morales Juárez 1422745

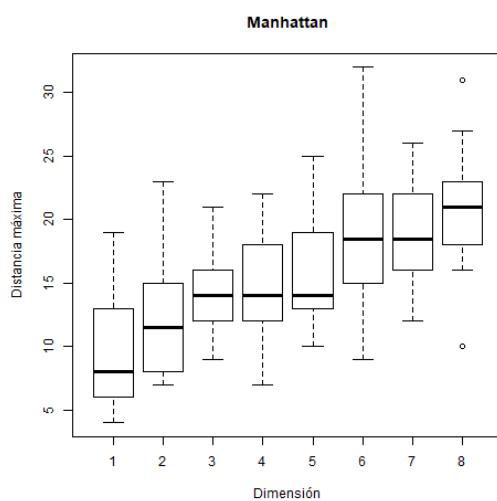
29 de enero de 2019

1. Objetivos *(a) free cs [1]:*

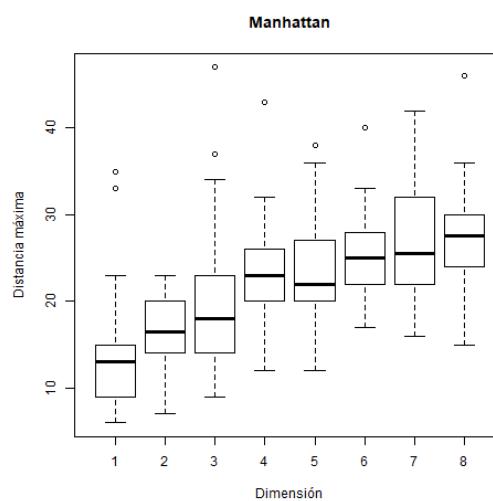
(1) Examinar de manera sistemática los efectos de la dimensión en la probabilidad de regreso al origen del movimiento Browniano para dimensiones 1 a 8 en incrementos lineales de uno, variando el número de pasos de la caminata como potencias de dos con exponente de 6 a 12 en incrementos lineales de uno, con 30 repeticiones del experimento para cada combinación. Grafica los resultados en una sola figura con diagramas de caja-bigote o de violin de tal forma que facilite comparar entre combinaciones de parámetros para poder evaluar si afectan los dos o si uno domina el comportamiento. //
[1]

2. Resultados

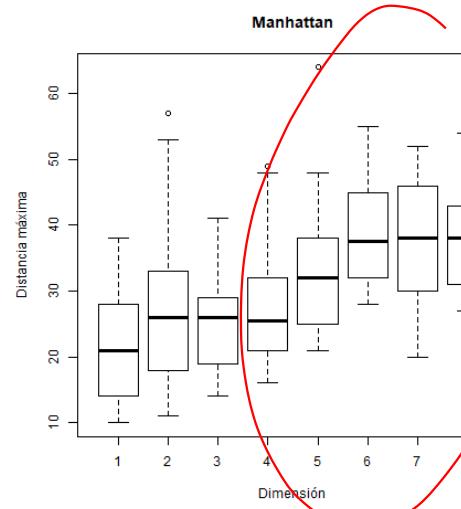
Los datos obtenidos a partir de las corridas virtuales reflejaron, como se puede ver en la Figura 1, distintos escenarios que se pudieron graficar para una mejor comparación y entendimiento; cada uno de estos con determinadas especificaciones nos mostró que entre mayor sea la cantidad de dimensiones existentes para el movimiento aleatorio de una partícula más lejana de su origen quedara al final de su movimiento. Cada gráfica se corrió 30 veces para poder determinar si por estadística pudiera volver al origen. Pero los resultados mostrarán que (al menos con las variables con las que se realizó el estudio) se aleja cada vez más del origen. Sin embargo, no se descarta la posibilidad de que esto pueda ocurrir, se necesitaría realizar otra corrida con diferentes parámetros para poder obtener un resultado positivo.



(c)



(b)



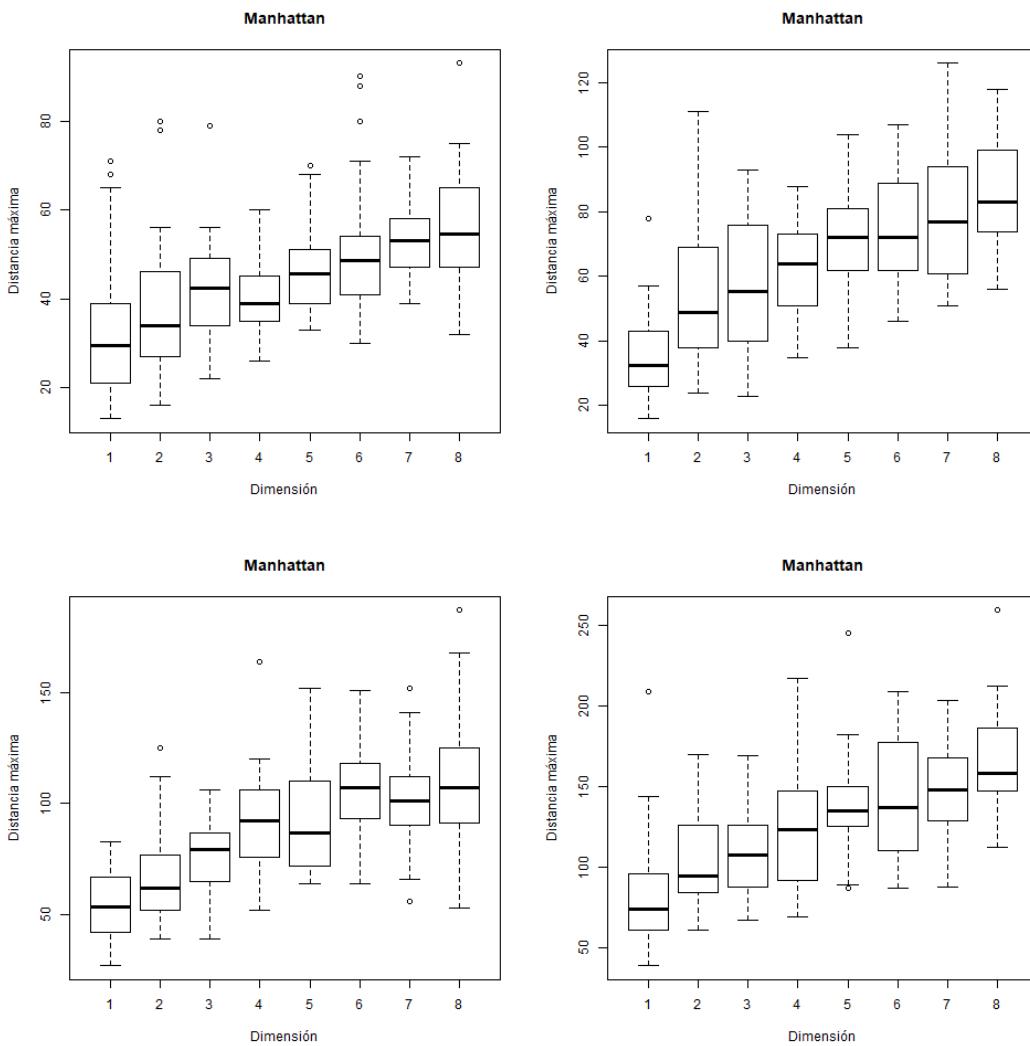


Fig. 1 ~~Graficas de comparación Manhattan, en orden de 2*6 hasta 2*12~~

3. Conclusiones

Se pudo concluir que entre mayor cantidad de dimensiones y menor número de pasos, la probabilidad de que una partícula en movimiento aleatorio vuelva al estado inicial es prácticamente nula.

Referencias

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

Búsqueda Local

Abraham Azael Morales Juárez 1422745

6 de junio de 2019

1. Introducción

El movimiento Browniano es el que realiza una partícula de manera aleatoria en un medio determinado. En esta práctica se realizará un experimento donde se genera una partícula y se le proporciona desplazamientos (pasos) aleatorios del mismo tamaño y en distintos planos (dimensiones)[\[3\]](#).

2. Objetivos

Examinar que tanta probabilidad tiene una partícula desplazándose con movimiento browniano de regresar a su punto de origen y observar que parámetros afectan en este resultado.

3. Resultados

Los datos obtenidos a partir de las corridas virtuales reflejaron (figura 1), distintos escenarios que se pudieron graficar para una mejor entendimiento; cada uno de estos con determinadas especificaciones mostró que entre mayor sea la cantidad de dimensiones existentes para el movimiento aleatorio de una partícula más lejana de su origen quedara al final de su movimiento. Cada gráfica se corrió 30 veces para poder determinar si por estadística pudiera volver al origen.

El código base que fue proporcionado al comienzo de la práctica [\[3\]](#), se modificó en base a otro previamente reportado [\[2\]](#) para la obtención de la probabilidad de que la partícula regrese al origen, la cantidad de pasos con la cual se realizarán los experimentos.

```
1 for (dimension in 1:8) {  
2   origen <- 0  
3   clusterExport(cluster , "dimension")  
4   resultadoP <- parSapply(cluster , 1:repetir ,  
5     function(r) {  
6       pos <- rep(0, dimension)  
7       mayor <- 0  
8       origen <- 0  
9       for (t in 1:Pasos) { #mod  
10         cambiar <- sample(1:dimension , 1)  
11         cambio <- 1  
12         if (runif(1) < 0.5) {  
13           cambio <- -1  
14         }  
15         pos[cambiar] <- pos[cambiar] + cambio  
16         if(all(pos==0)) #Comprobar que haya regresado a los puntos de origen  
17         {  
18           origen <- origen + 1  
19         }  
20       }  
21       return((origen/Pasos)*100)  
22     })  
23   datafin <- rbind(datafin , resultadoP )  
24 }  
25 }
```

Los resultados mostrarán que (al menos con las variables con las que se realizó el estudio) se aleja cada vez más del origen. Sin embargo, no se descarta la posibilidad de que esto pueda ocurrir, se necesitaría realizar otro análisis con diferentes parámetros para poder obtener un resultado con mayor grado de confianza.

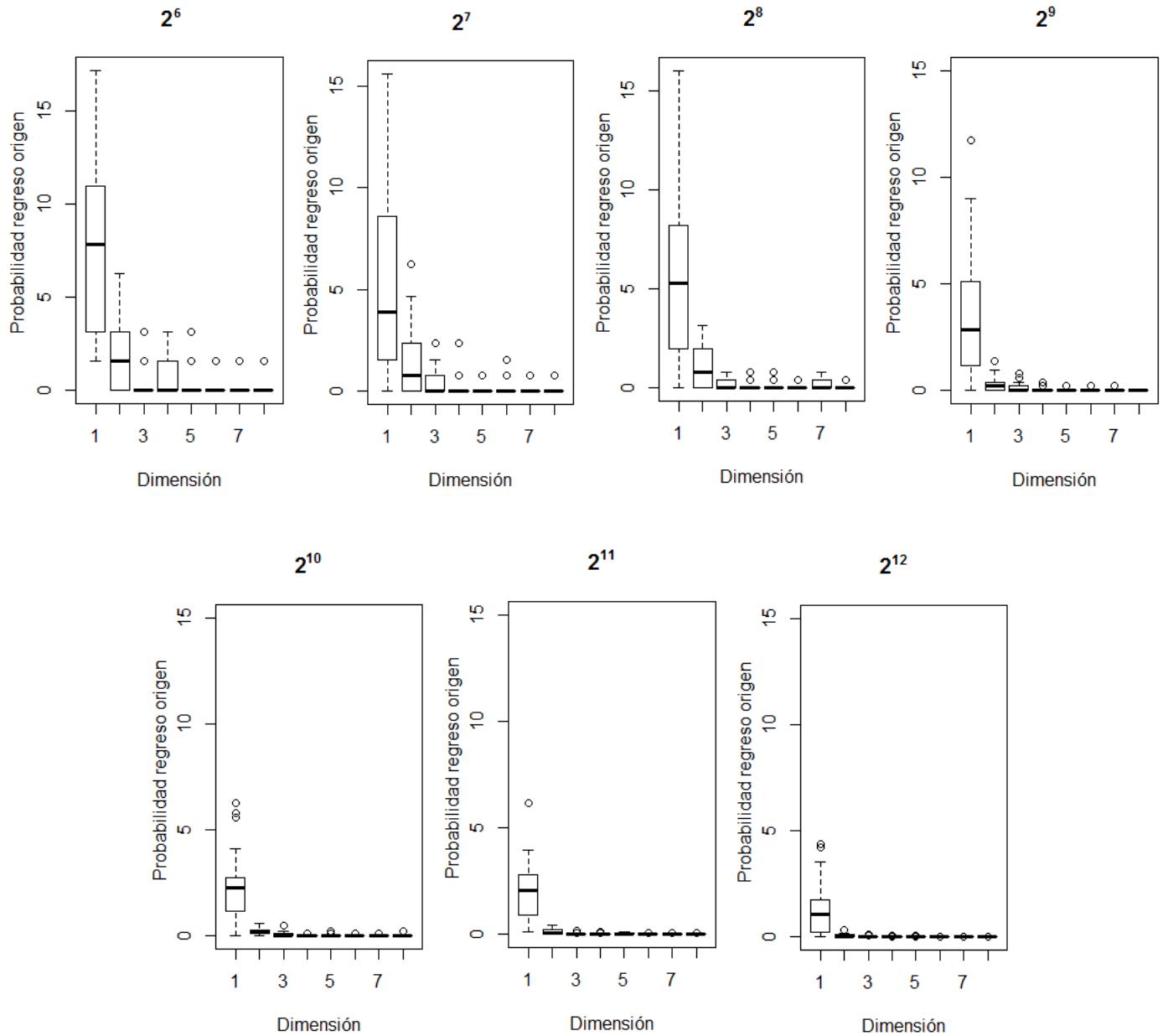


Figura 1: Probabilidad que las partículas regresen al origen con incrementos en sus pasos de 2^6 hasta 2^{12}

Se puede observar que conforme aumenta la cantidad de dimensiones disminuye la probabilidad de regresar al punto de origen, de la misma manera disminuye esta probabilidad conforme aumentan la cantidad de pasos realizados, observándose esto desde el comienzo con una dimensión.

4. Conclusiones

Se pudo concluir que entre mayor sea la cantidad de dimensiones y mayor el número de pasos, la probabilidad de que una partícula en movimiento aleatorio vuelva al estado inicial es demasiado baja. Sin embargo, aún así esto no se descarta.

5. Reto 2

El segundo reto, es realizar una comparación en los tiempos que se tardan en realizar los análisis comparando el método paralelo y secuencial. Para este reto se le agregó el parámetro System.time() en el código. y al final se realizarán las comparaciones entre estos métodos.

```
1 library(parallel)
2
3 repetir <- 30 #Repeticiones
4 Pasos <- 300
5 eucl <- FALSE
6
7 Paral.inicio <- Sys.time()
8
9 cluster <- makeCluster(detectCores() - 1)
10 clusterExport(cluster, "Pasos")
11 clusterExport(cluster, "repetir")
12
13
14 for (dimension in 1:8) {
15   origen <- 0
16   clusterExport(cluster, "dimension")
17   resultadoP <- parSapply(cluster, 1:repetir,
18                           function(r) {
19     pos <- rep(0, dimension)
20   })
21 stopCluster(cluster)
22
23 paral.final <- Sys.time()
24 Tiempo.paral <- paral.final - Paral.inicio
25 print(Tiempo.paral)
26
27 #Secuencial
28
29 secu.inicio <- Sys.time()
30
31 cluster <- makeCluster(detectCores() - 1)
32 clusterExport(cluster, "Pasos")
33 clusterExport(cluster, "repetir")
34 clusterExport(cluster, "eucl")
35 datafin1 <- data.frame()
36
37 for (dimension in 1:8) {
38   origen <- 0
39   clusterExport(cluster, "dimension")
40   resultadoP <- Sapply(cluster, 1:repetir,
41                         function(r) {
42   stopCluster(cluster)
43
44 secu.final <- Sys.time()
45 tiempo.sec <- secu.final - secu.inicio
46 print(tiempo.sec)
47
48 Tiempos <- data.frame(Tiempo.paral, tiempo.sec)
49 print(Tiempos)
50 prueba.T <- kruskal.test(Tiempos)
51 prueba.T$p.value
52 prueba.T$p.value < 0.05
```

En el cuál arrojó como resultado los siguientes datos:

- Tiempo paralelo: 1,39008 segundos.
- Tiempo secuencial: 1,509087 segundos.

Además, se realizó una prueba estadística tomando como punto de apoyo el reporte de un compañero , la prueba de **kruskal.test** para observar su hubiese alguna diferencia significativa, donde se obtuvo un valor de 0,3173105, tomándose en cuenta que el valor de $p < 0,05$. Lo que indica que las diferencias son muy pocas pero aun así se puede observar que si hubo un ahorro en los tiempos de ejecución.

Referencias

- [1] Ricardo Rosas Macías. Práctica 1: Movimiento browniano. 2019. URL [https://github.com/RicardoRosMac/
NanoSimulation/tree/master/HWP1](https://github.com/RicardoRosMac/NanoSimulation/tree/master/HWP1).
- [2] Edson Raúl Cepeda Márquez. Práctica 1: Movimiento browniano. 2019. URL [https://sourceforge.net/p/
systemssimulation/activity/?page=0&limit=100#5c4fff57ee24ca5fb0a844de](https://sourceforge.net/p/systemssimulation/activity/?page=0&limit=100#5c4fff57ee24ca5fb0a844de).
- [3] Satu Elisa Schaeffer. Práctica 1: Movimiento browniano, 2019. URL [https://elisa.dyndns-web.com/teaching/
comp/par/p1.html](https://elisa.dyndns-web.com/teaching/comp/par/p1.html).

Práctica 2: Autómata Celular

Abraham Azael Morales Juárez

REPORTE:

En este reporte aún no se lograba comprender como usar el programa, a pesar de que se estuvo intentando hacerla por varios días. Con esta práctica comprendí que tiene un gran rango de aplicaciones el uso de programas de simulación computacional y sobre todo en áreas que son complicadas y costosas al intentar desarrollar un material, para esta manera ahorrar tiempo y dinero.

Algo muy importante que cabe mencionar es que en esta y en otras prácticas adelante se tuvieron que modificar los parámetros para poder rectificarlas debido a que ya no se tiene acceso al mismo equipo de cómputo. Por esa razón las gráficas no son las mismas.

*Apunt de
l'automa*

Autómata celular

Abraham Azael Morales Juárez 1422745

5 de febrero de 2019

1. Introducción

En esta práctica de autómatas celulares, que ~~sólo~~ representadas por una matriz booleana (que es una matriz que contiene ceros y unos los cuales representan la vida y la muerte, respectivamente), determinaremos la supervivencia de cada celda con base de los valores de sus ocho vecinos [1].

2. Objetivo

Determinar el número de iteraciones que procede de 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 [1].

3. Resultados

Falta todo

Referencias

- [1] Elisa. Autómata celular, 2018. URL <https://elisa.dyndns-web.com/teaching/comp/par/p2.html>.

Apellidos

Autómata celular

Abraham Azael Morales Juárez 1422745

6 de junio de 2019

1. Introducción

En esta práctica de autómatas celulares, que serán representadas por una matriz booleana (que es una matriz que contiene 0 y 1 los cuales representan la vida y la muerte, respectivamente), determinaremos la supervivencia de cada celda con base de los valores de sus ocho vecinos [3]. La regla de supervivencia es sencilla: si una celda tiene tres vecinos vivos alrededor suyo (vecinos) estará viva. Se usarán números pseudoaleatorios para crear el estado inicial.

2. Objetivo

El objetivo principal es diseñar y experimentar la determinación de iteraciones que se necesitan para que las celdas mueran dentro de la simulación.

3. Resultados

Primeramente, se tuvo que modificar el código base proporcionado [3], el primer parámetro que se modificó fue el tamaño de la matriz en lugar de 30×30 se modificó a 15×15 debido a que la computadora usada no cuenta con buenas características. De igual manera se modificó la probabilidad de que las celdas vivas se generaran en la matriz teniendo como base el reporte de un compañero [2], un fragmento del código es colocado para una mejor ilustración.

```
1 library(sna)
2 library(parallel)
3
4 dim <- 15
5 num <- dim^2
6 data <- data.frame()
7
8 for(p in seq(0.1, 0.9, 0.1)) { #Probabilidad
9   vector <- c()
10  for(rep in 1:10) { #Repeticiones del experimento
11    actual <- matrix(1 * (runif(num) < p), nrow=dim, ncol=dim)
12    suppressMessages(library("sna"))
13    paso <- function(pos) {
14      fila <- floor((pos - 1) / dim) + 1
15      columna <- ((pos - 1) %% dim) + 1
16      vecindad <- actual[max(fila - 1, 1) : min(fila + 1, dim),
17                            max(columna - 1, 1) : min(columna + 1, dim)]
18      return(1 * ((sum(vecindad) - actual[fila, columna]) == 3))
19    }
```

Seguido del número de repeticiones que se colocaron para poder realizar el experimento y obtener datos confiables. También se establece el número de iteraciones los cuales permitirán identificar cuando ya todas las celdas murieron y se guarda el número de iteración donde lo hicieron.

```

1 for (iteracion in 1:15) { #Rango de iteraciones
2   clusterExport(cluster , "actual")
3   siguiente <- parSapply(cluster , 1:num, paso)
4   if (sum(siguiente) == 0) { # todos murieron
5     print("Ya no queda nadie vivo.")
6     itmuer <- iteracion #iteracion donde ninguna celda a quedado viva
7     break;
8   }
9   actual <- matrix(siguiente , nrow=dim , ncol=dim , byrow=TRUE)
10 }

```

Posteriormente ya es posible agrupar los datos y graficar los resultados, (figura 1).

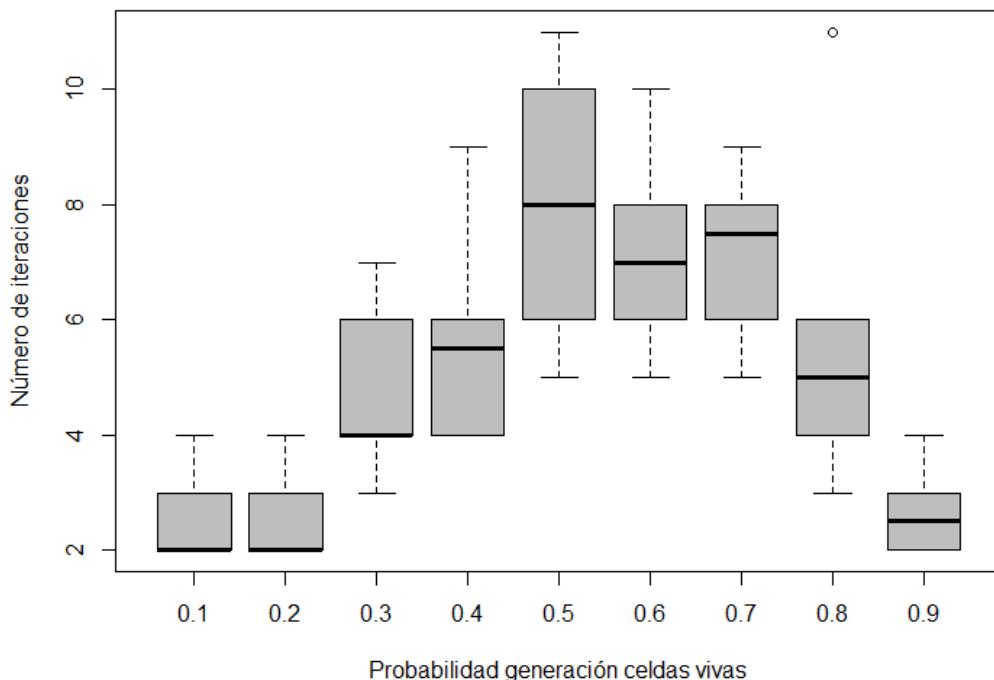


Figura 1: Número de iteraciones que necesitaron para que las celdas murieran con respecto a la probabilidad.

4. Conclusiones

Se observa que cuando aumenta la probabilidad de generación de celdas el número de iteraciones aumenta, por lo tanto, entre mayor sea el número de celdas generadas mayor será su probabilidad de mantenerse con vida. Además, que la probabilidad de 0,5 hasta el 0,7 se mantuvo en condiciones de equilibrio en las iteraciones realizadas entre sí. A partir de la probabilidad 0,8 en adelante decaen las iteraciones por lo que las celdas requieren menor cantidad para desaparecer.

5. Reto 1

Para este reto se hicieron modificaciones del código usado, para poder crear un crecimiento de núcleos en lugar de un decaimiento.

De la misma manera de agrega el crecimiento de cada una de las semillas presentes en la matriz y como se expanden hasta que ocupan toda la matriz.

En esta ocasión se pudo aplicar con parámetros mayores porque se tuvo acceso a una computadora con características

superiores. También se colocó una matriz de 50×50 , se colocó de manera aleatoria la posición de las semillas que formaran los núcleos. Además, se realizaron cambios en la función `paso`. Solo se puede generar un núcleo donde no se haya asignado uno con anterioridad, es decir, que no se sobre escribe una celda con un núcleo ya asignado, debido a que solamente busca sobre las que sean cero para realizar la expansión.

```

1 library(parallel)
2 dim <- 40
3 num <- dim^2
4 for (rep in 1:20) {
5   actual <- matrix(rep(0, num), nrow=dim, ncol=dim)
6   actual<-replace(actual, sample(num, 20), seq(1:20))
7   suppressMessages(library("sna"))
8   inicio=paste("p2_t0", rep, ".png", sep="")
9   png(inicio)
10  plot.sociomatrix(actual, diaglab=FALSE, main="Inicio")
11  graphics.off()
12
13 paso <- function(pos) {
14   a<-0
15   fila <- floor((pos - 1) / dim) + 1
16   columna <- ((pos - 1) %%dim) + 1
17   vecindad <- actual[max(fila - 1, 1) : min(fila + 1, dim),
18                      max(columna - 1, 1): min(columna + 1, dim)]
19   if(actual[fila ,columna]==0){
20     for (i in 1:length(vecindad)) {
21       if(vecindad[i]!=0){
22         a<-vecindad[i]
23         break;
24       }
25     }
}

```

Se generó un gif que se puede observar en el repositorio, pero se colocaron algunas imágenes del crecimiento de los núcleos, (figura 2 y 3).

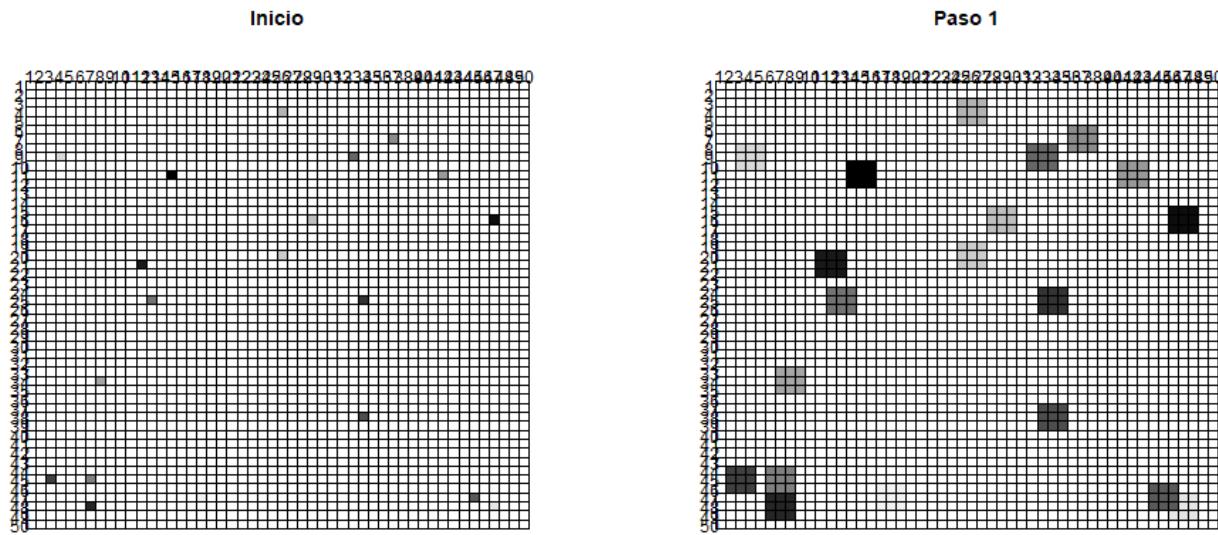


Figura 2: Generación de las celdas y crecimiento de los mismos hasta alcanzar los bordes.

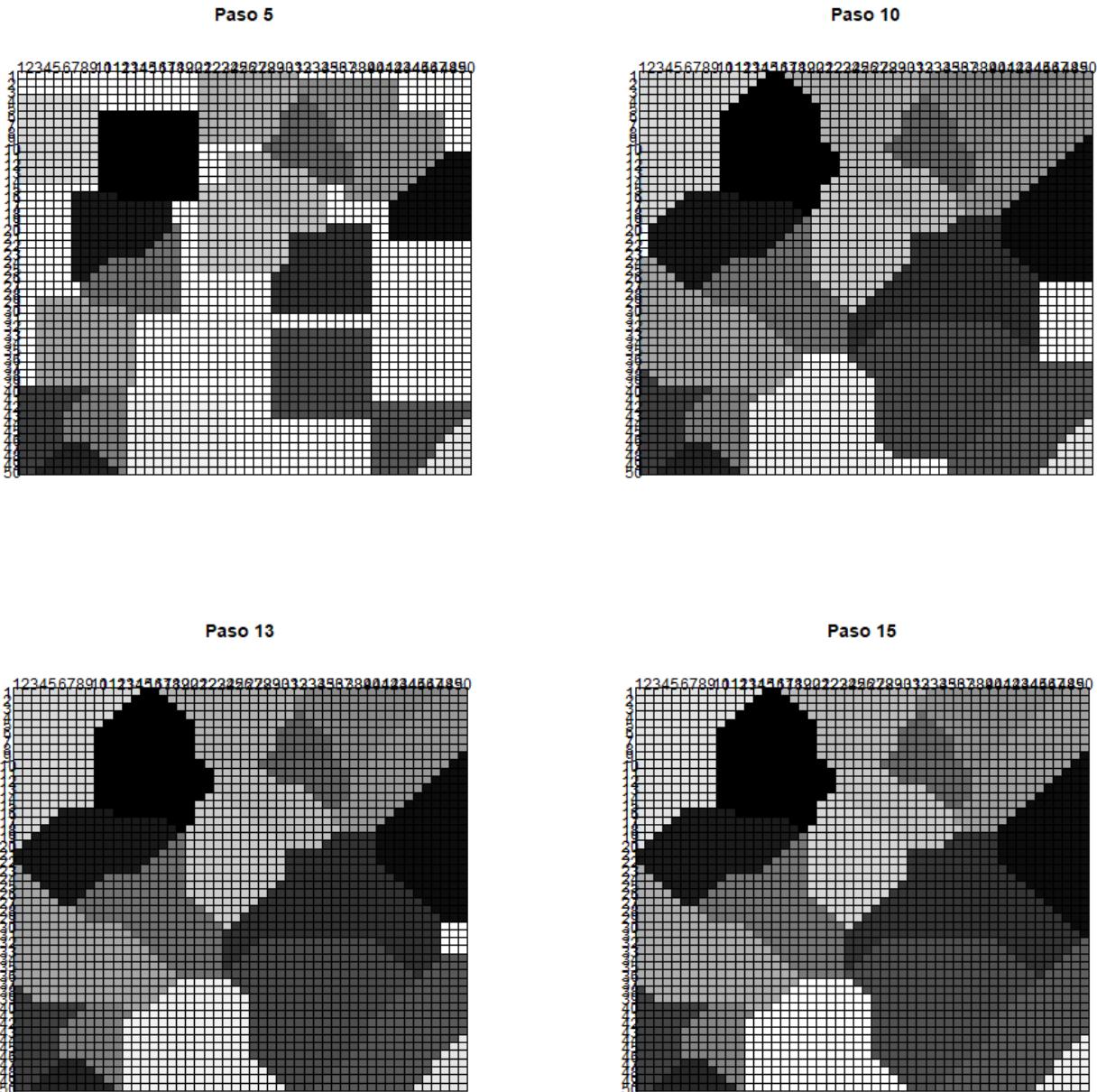


Figura 3: Generación de las celdas y crecimiento de los mismos hasta alcanzar los bordes.

Se puede concluir que los tamaños de los núcleos creados y que se expandieron en el centro y sin tocar los bordes, no presentan una proporcionalidad en sus tamaños debido, a que como se puede observar, (figuras 2 y 3), que el crecimiento de los núcleos internos está definido por el crecimiento de los núcleos exteriores, como se observa en el paso cinco, donde se estas comienzan a definir sus límites. Solo basta con un total de quince pasos para poder ocupar toda la matriz.

Referencias

- [1] Yessica Reyna Fernandez. Práctica 2: Autómata celular. 2018. URL <https://sourceforge.net/p/simulacion-de-sistemas/activity/?page=0&limit=100#5c065ccbee24ca4106c98602>.
- [2] Edson Raúl Cepeda Marquez. Práctica 2: Autómata celular. 2019. URL <https://sourceforge.net/projects/systemssimulation/>

- [3] Satu Elisa Schaeffer. Práctica 2: Autómata celular, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

2205

Teoría de Colas

4

Abraham Azael Morales Juárez 1422745

12 de febrero de 2019

1. Introducción

La teoría de colas, es el estudio matemático de las características de sistemas de líneas o colas. Estas se presentan diariamente en la vida cotidiana. Esto ocurre inclusive, por ejemplo, cuando realizamos una llamada, en ocasiones tenemos que esperar porque el usuario está ocupado. Esto también ocurre en el supermercado, almacenes, aeropuertos, solo por mencionar algunos. Por esta razón es ~~de~~ importante realizar estudios en el comportamiento de estos tipos de sistemas [1].

le

2. Objetivos

Examinar cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al cluster. Usar datos de entrada de un vector que contiene primos grandes y no-primos con un mismo número de dígitos. Investiga también el efecto de la proporción de primos y no primos en el vector. Además el efecto de la magnitud de los números incluidos en el vector, con pruebas estadísticas adecuadas [3].

3. Resultados y Discusión

En los datos obtenidos de la simulación se puede observar claramente la gran diferencia de procesamiento de datos, debido a la cantidad de núcleos destinados para trabajar con la información. También como era de esperarse, la cantidad de información influye en el tiempo que tardó en realizarse el experimento. Se realizó un análisis en paralelo y las especificaciones utilizados para la realización de esta práctica son las siguientes: 1 núcleo, 6 núcleos, y 7 núcleos, para los cuales los rangos de números fueron: 10,000 – 50,000, 50,000 – 250,000 y 250,000 – 500,000. Cabe destacar que se usaron 2 procesadores distintos core i7 y core i5, 8 y 2 núcleos respectivamente. Se usaron distintos códigos los cuales definían funciones, tanto para números primos como para compuestos, que son los siguientes códigos: EL código para obtener la información y almacenarla en vectores es el siguiente:

```
compuesto <- function(n) {
  if (n < 4) {
    return(FALSE)
  }
  for (i in 2:(n-1)) {
    if (n %% i == 0) {
      return(TRUE)
    }
  }
  return(FALSE)
}

primo <- function(n) {
  if (n == 1 || n == 2) {
    return(TRUE)
  }
  if (n %% 2 == 0) {
    return(FALSE)
  }
  for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
    if ((n %% i) == 0) {
      return(FALSE)
    }
  }
  return(TRUE)
}
```

Figura 1: Código para designar la función primo y compuesto

En la figura 3, que son gráficas caja-bigote, se presentan los resultados obtenidos del análisis con 7 núcleos, en los cuales las mejores gráficas se obtienen con los rangos menores, es decir, 10,000 – 50,000, en esta gráfica se observa que se obtuvo los primeros

```

suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectores(7) - 1))
otp <- numeric()
itp <- numeric()
atp <- numeric()
for (r in 1:replicas) {
  otp <- c(otp, system.time(foreach(n = original, .combine=c) %dopar% primo(n))[3]) # de menor a mayor
  itp <- c(itp, system.time(foreach(n = invertido, .combine=c) %dopar% primo(n))[3]) # de mayor a menor
  atp <- c(atp, system.time(foreach(n = sample(original), .combine=c) %dopar% primo(n))[3]) # orden aleatorio
}
stopImplicitcluster()
summary(otp) # primos desde:hasta
summary(itp) # primos hasta:desde
summary(atp) # primos aleatorio

suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectores(7) - 1))
otc <- numeric()
itc <- numeric()
atc <- numeric()
for (r in 1:replicas) {
  otc <- c(otc, system.time(foreach(n = original, .combine=c) %dopar% compuesto(n))[3]) # de menor a mayor
  itc <- c(itc, system.time(foreach(n = invertido, .combine=c) %dopar% compuesto(n))[3]) # de mayor a menor
  atc <- c(atc, system.time(foreach(n = sample(original), .combine=c) %dopar% compuesto(n))[3]) # orden aleatorio
}
stopImplicitcluster()
summary(otc) # compuestos desde:hasta
summary(itc) # compuestos hasta:desde
summary(atc) # compuestos aleatorio
# graficas boxplot
par (mfrow=c(1,1))
nombres = c("Original", "Invertido", "Aleatorio")
boxplot(summary(otp), summary(itp), summary(atp), main="Números primos 6 núcleos", xlab="vectores", ylab="Tiempo", boxwex=0.15, names = nombres, range = 0)
boxplot(summary(otc), summary(itc), summary(atc), main="Números compuestos 6 núcleos", xlab="vectores", ylab="Tiempo", boxwex=0.15, names = nombres, range = 0)

```

Figura 2: Código vector, boxplot y paralelo

resultados casi al mismo tiempo, pero teniendo datos muy irregulares con bigotes ~~que~~ largos y en donde particularmente con los números compuestos, se observa ~~que~~ más claramente la priorización del análisis debido a que, realizar todos las acciones que se necesitan partiendo de datos grandes es más complicada que partir de datos más sencillos. Aunque, se puede observar la diferencia que hay entre el análisis de 7 núcleos y de 1 núcleo. Sin embargo, el análisis con el de 6 núcleos, no hubo mucha variación con el de 1, a diferencia de lo que se ha reportado [4], lo más probable fue por el hecho que las computadoras usadas tienen designaciones prioritarias que pueden requerir el uso de núcleos para actualizarse o simplemente ejecutarse en segundo plano [2].

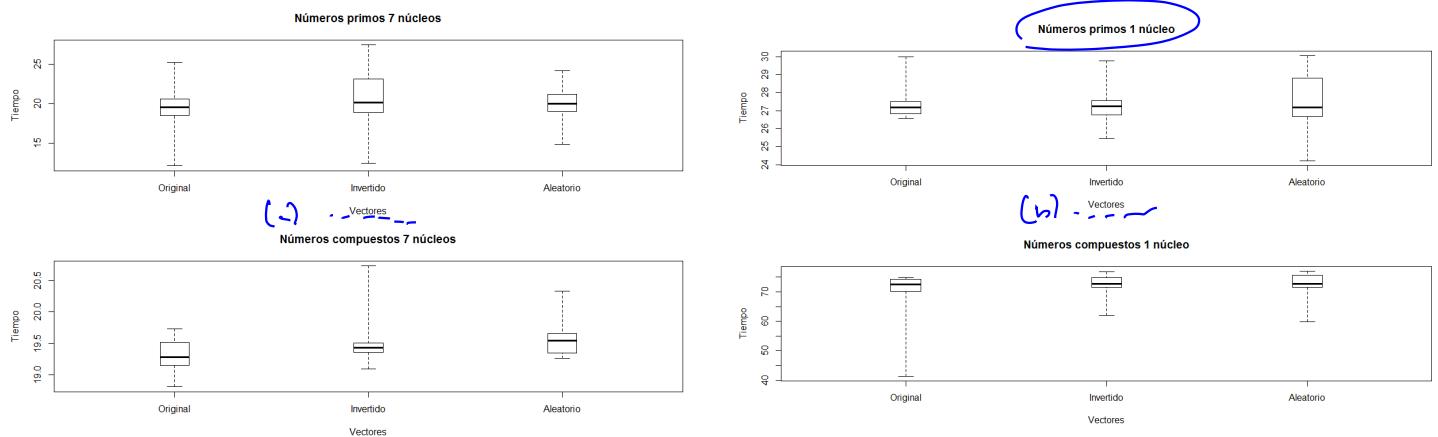
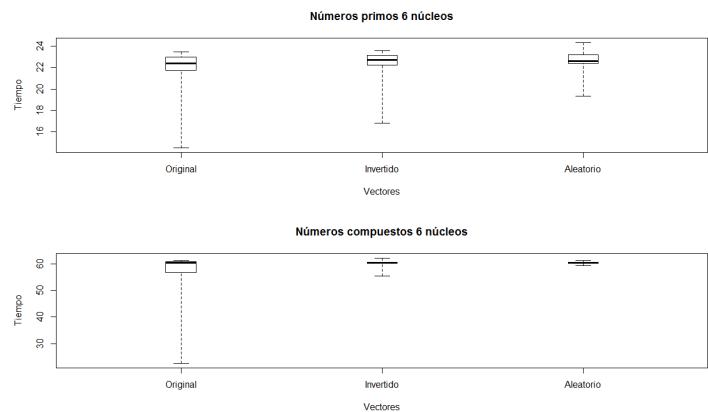


Figura 3: 7, 6 y 1 núcleo, 10,000 - 0,000



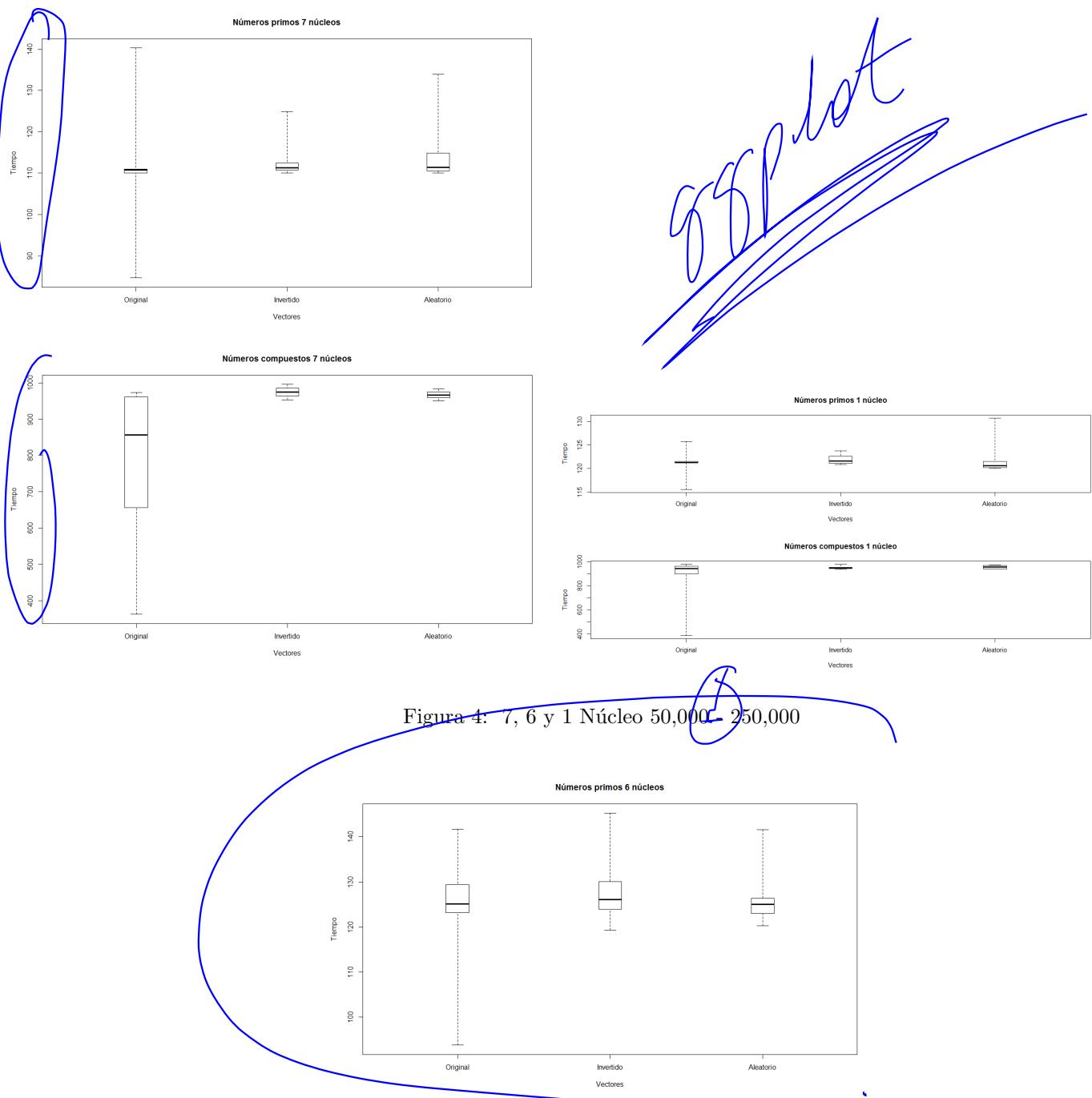


Figura 4: 7, 6 y 1 Núcleo 50,000 – 250,000

Con los resultados de 50000 – 250000, igual que los anteriores son muy distintos a los esperados y reportados [4], los tiempos de análisis son más cortos, (al menos con los números primos), con la computadora que tiene el procesador core i5. Mientras que con los 6 y 7 núcleos fue un poco mayor el tiempo esperado, pero similar entre ellos.

En el análisis más grande que se realizó, no se pudo obtener con 6 núcleos, por el tiempo de procesamiento que cada corrida requería. Aquí si se observó una diferencia muy grande entre los procesadores core i7 y el core i5, específicamente con los números compuestos, como se esperaba., y un poco menos con los números primos. Aunque, los resultados obtenidos siguen viéndose afectados, principalmente por los sistemas operativos instalados, que priorizan funciones de Windows ocasionando que los núcleos hagan otras tareas además de las asignadas. Afectando el proceso en paralelo que se está ejecutando [2].

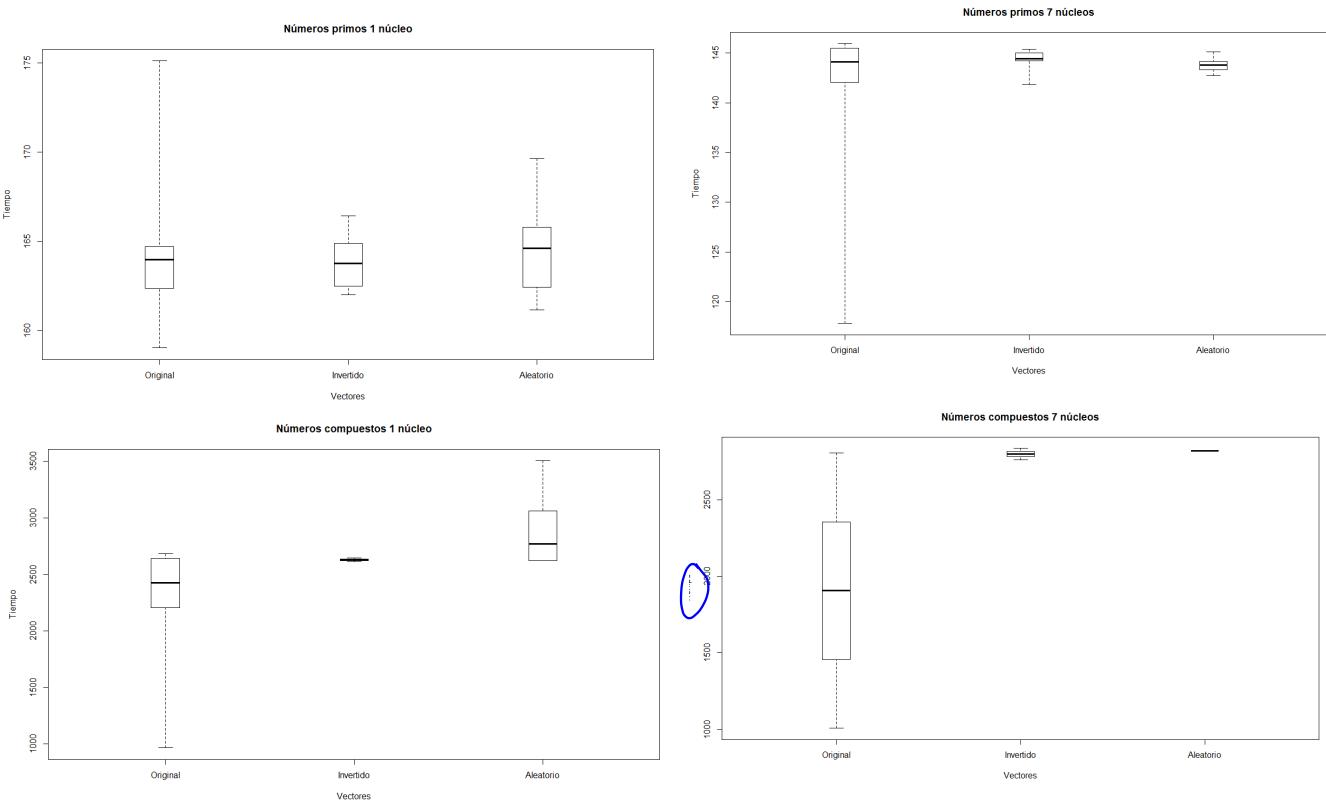


Figura 5: 7 y 1 núcleo, 250,000 - 500,000

4. Conclusiones

Estos análisis son de gran importancia, debido a que nos explican el comportamiento de las líneas de espera y el como poder tratar con ellas, con las herramientas disponibles. El uso de la herramienta paralelo es de gran importancia, debido a que con estos se puede llegar a predecir el comportamiento de distintos fenómenos o sistemas. Los equipos que se utilicen para realizar este tipo de simulaciones necesitan tener un control total sobre el software, para poder manipular y priorizar de manera voluntaria los núcleos.

Referencias

- [1] Pedro García. Aplicando la teoría de colas en dirección de operaciones. *Universidad Politécnica de Valencia*, 2015.
- [2] Rodríguez D. Joaquín López. Análisis de rendimiento de algoritmos paralelos. *Tecnológico de Costa Rica*, 2014.
- [3] Satu Elisa Schaeffer. Teoría de colas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.
- [4] Javier Serrano. Memoria, gestión de procesos en los sistemas operativos. *TFC: Arquitectura de computadoras y sistemas operativos*, 2011.

Domingo Rodríguez and Joaquin Lopez.

Teoría de colas

Abraham Azael Morales Juárez 1422745

6 de junio de 2019

1. Introducción

La teoría de colas es el estudio matemático de las características de sistemas de líneas o colas. Estas se presentan diariamente en la vida cotidiana. Esto ocurre inclusive, por ejemplo, cuando realizamos una llamada, en ocasiones tenemos que esperar porque el usuario está ocupado. Esto también ocurre en el supermercado, almacenes, aeropuertos, solo por mencionar algunos. Por esta razón es importante realizar estudios en el comportamiento de estos tipos de sistemas [1].

2. Objetivos

Examinar cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al cluster. Usar datos de entrada de un vector que contiene primos grandes y no-primos con un mismo número de dígitos. Investiga también el efecto de la proporción de primos y no primos en el vector. Además el efecto de la magnitud de los números incluidos en el vector, con pruebas estadísticas adecuadas [3].

3. Resultados y Discusión

En los datos obtenidos de la simulación se puede observar claramente la gran diferencia de procesamiento de datos, debido a la cantidad de núcleos destinados para trabajar con la información. También como era de esperarse, la cantidad de información influye en el tiempo que tardó en realizarse el experimento. Se realizó un análisis en paralelo y las especificaciones utilizados para la realización de esta práctica son las siguientes:

cinco y siete núcleos, para los cuales los rangos de números fueron: 5,000 – 10,000, 10,000 – 15,000 y 20,000 – 30,000. Se usaron distintos códigos los cuales definían funciones, tanto para números primos como para compuestos.

```
1 compuesto <- function(n) {
2   if (n < 4) {
3     return(FALSE)
4   }
5   for (i in 2:(n-1)) {
6     if (n %%i == 0) {
7       return(TRUE)
8     }
9   }
10  return(FALSE)
11 }
12
13 primo <- function(n) {
14   if (n == 1 || n == 2) {
15     return(TRUE)
16   }
17   if (n %%2 == 0) {
18     return(FALSE)
19   }
20   for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
21     if ((n %%i) == 0) {
22       return(FALSE)
23     }
```

En la figura 1 se muestran los resultados de la ejecución con un rango menor de 5,000 – 10,000, mostrando diferencias menores, sin embargo, estas diferencias concuerdan con lo ya reportado [4] además que se van haciendo mayores conforme el rango y cantidad de dígitos va en aumentando.

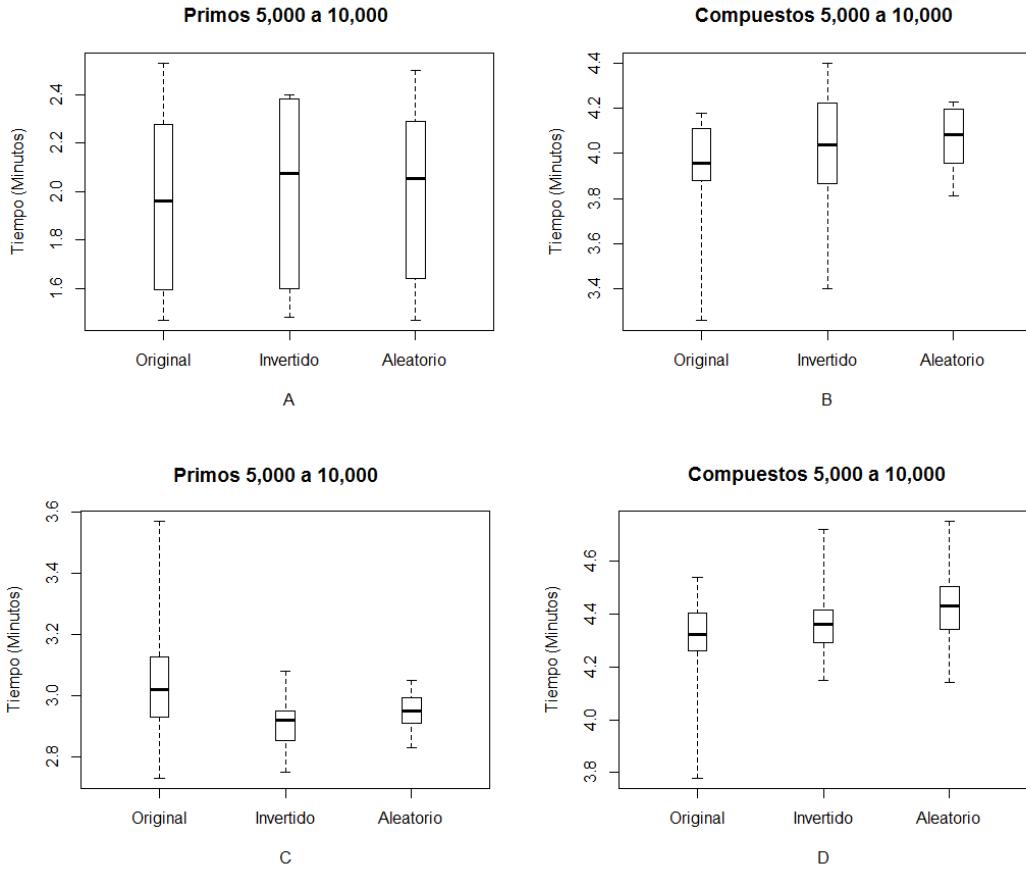


Figura 1: Comparación de tiempos de análisis, variando la cantidad de núcleos aplicados para realizarlo. Se muestran los resultados entre un rango de 5,000 – 10,000, A y B corresponden a siete núcleos, C y D a cinco núcleos.

Con los resultados de 10,000 – 15,000 (figura 2), se observa de misma manera como varían los tiempos de ejecución y como a medida que aumentan el rango a analizar junto con la cantidad de dígitos que se están colocando, aumenta el tiempo de ejecución y concordando con lo que se ha reportado [4] la aplicación de más núcleos conlleva a un análisis con mayor eficiencia en el tiempo.

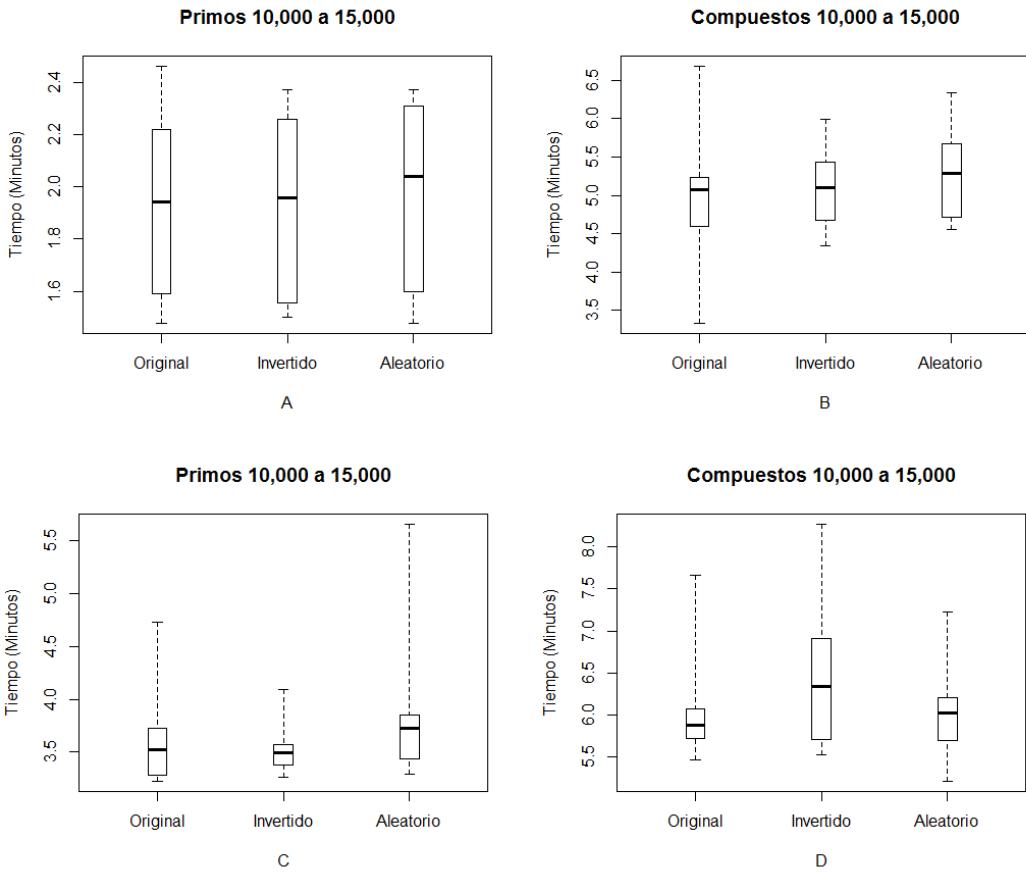


Figura 2: Comparación de tiempos de análisis, variando la cantidad de núcleos aplicados para realizarlo. Se muestran los resultados entre un rango de 10,000 – 15,000, A y B corresponden a siete núcleos, C y D a cinco núcleos.

En el análisis de mayor rango (20,000 – 30,000) (figura 3), es donde ocurrió el alza en los tiempos de ejecución de hasta de veintidos minutos para los números compuestos. Aunque es bien conocido el hecho que la velocidad de procesamiento disponible está muy a la mano con el tipo de sistema operativo, debido a que estos por determinación automática pueden destinar cierta potencia a otros procesos [2]. Se obtuvieron los mismos resultados, entre mayor cantidad de núcleos destinados obtienen mejor tiempo de análisis.

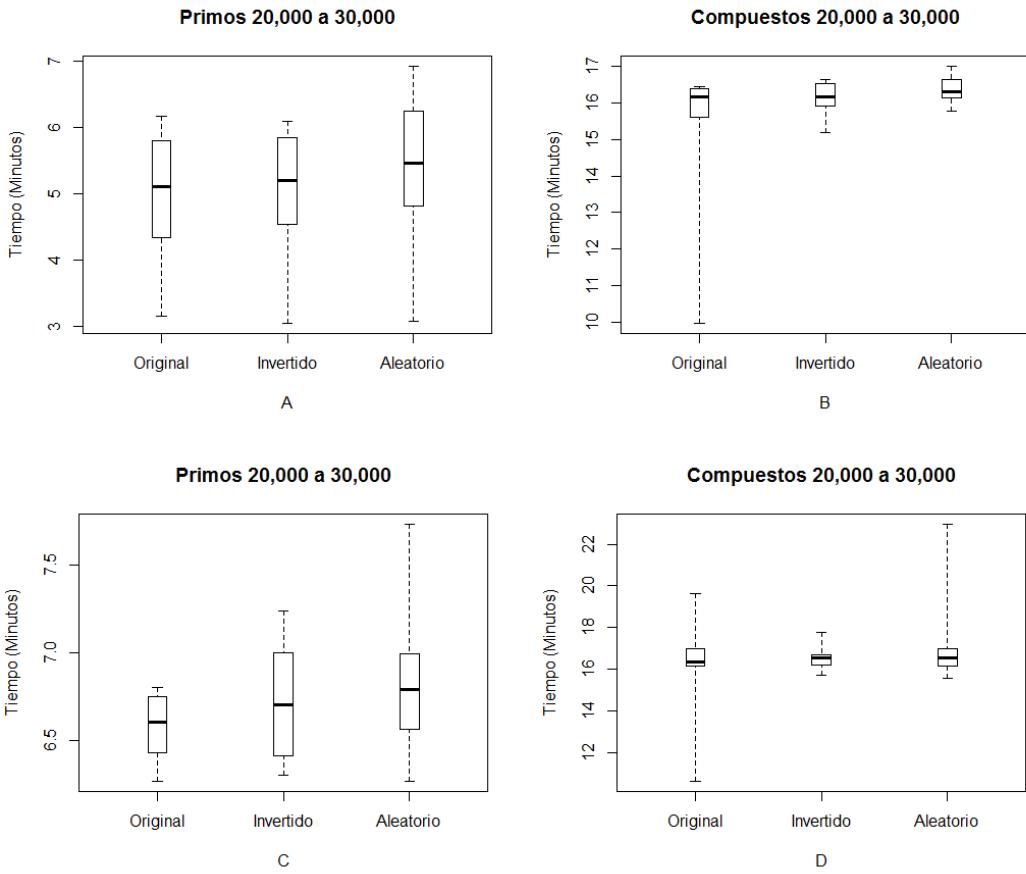


Figura 3: Comparación de tiempos de análisis, variando la cantidad de núcleos aplicados para realizarlo. Se muestran los resultados entre un rango de 20,000 – 30,000, A y B corresponden a siete núcleos, C y D a cinco núcleos.

4. Conclusiones

Estos análisis son de gran importancia, debido a que explican el comportamiento de las líneas de espera y el como poder tratar con ellas, con las herramientas disponibles. El uso de la herramienta paralelo es de gran importancia, debido a que con estos se puede llegar a predecir el comportamiento de distintos fenómenos o sistemas. Los equipos que se utilicen para realizar este tipo de simulaciones necesitan tener un control total sobre el software, para poder manipular y priorizar de manera voluntaria los núcleos.

Referencias

- [1] Pedro García. Aplicando la teoría de colas en dirección de operaciones. *Universidad Politécnica de Valencia*, 2015.
- [2] Joaquín Andrés López Molina and Daniel Mauricio Rodríguez Alpizar. Análisis de rendimiento de algoritmos paralelos. *Tecnológico de Costa Rica*, 2014.
- [3] Satu Elisa Schaeffer. Teoría de colas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.
- [4] Javier Serrano. Memoria, gestión de procesos en los sistemas operativos. *TFC: Arquitectura de computadoras y sistemas operativos*, 2011.

Diagramas de Voronoi

Abraham Azael Morales Juárez 1422745

19 de febrero de 2019



1. Objetivos

Analizar la relación entre el número de semillas y del tamaño de la matriz, tomando en cuenta los largos de las grietas. Observar cuantas celdas (diagramas voronoi) existen alrededor de la grieta. Además de medir la mayor distancia de Manhattan entre el extremo que esta a una distancia mayor del borde [2].

2. Metodología y resultados

Se colocó una cantidad k de semillas al azar en una matriz que representaba un material, posteriormente se produjo en ellas la presencia de diagramas voronoi desarrolladas a partir de la ubicación de las semillas [1]. La cantidad de semillas fue uniforme en intervalos enteros ($1:k$), ver figura 1.

$\text{Vect} + \Sigma \dots ?$

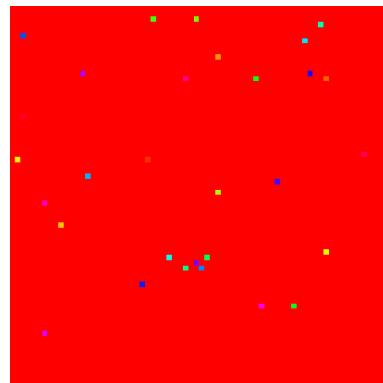
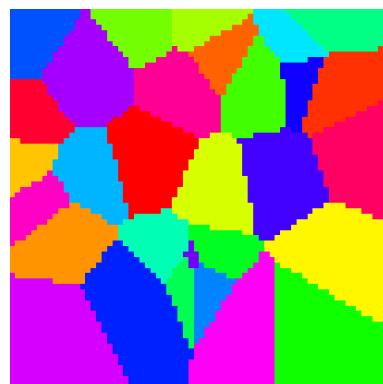


Figura 1: Semillas

Después se determinó que espacios vacíos corresponden al espacio de cada semilla, en otras palabras, para cada posición que vale cero, se calculó la distancia euclídea entre las semillas y la posición de interés, ver figura 2.



y a ver

Figura 2: Celdas Voronoi

Estas celdas representan núcleos en algún proceso de cristalización en un material y se provoca una grieta en ese material, ver figura 3, la cual se propaga con mayor velocidad entre las fronteras de los núcleos y viaja dificultosamente por el interior de la celda. Además, se toma en cuenta que no hay propagación preferencial por parte de la grieta.

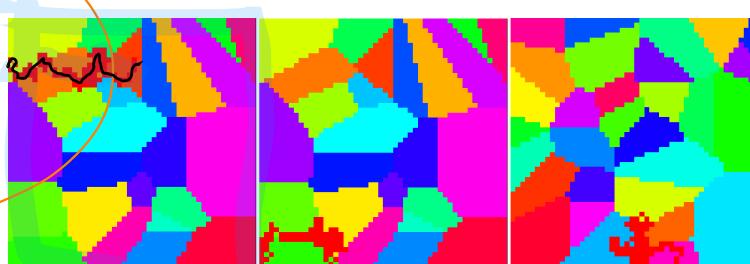


Figura 3: Grietas

\times $\$ \backslash \text{times} \$$

Para el desarrollo de la práctica se consideraron diferentes parámetros: tamaño de matriz, 50×50 , 60×60 y 70×70 , cantidad de semillas 20 , 25 y 30 . En la figura 4, se tienen las distancias Manhattan de cada una de las matrices, además se observa influencia de la cantidad de semillas aplicada en la generación de grietas, también que a pesar de la variación que se aplicó no hubo diferencia significante en las distancias manhattan, sin embargo, sí hubo en el tamaño de las grietas generadas.

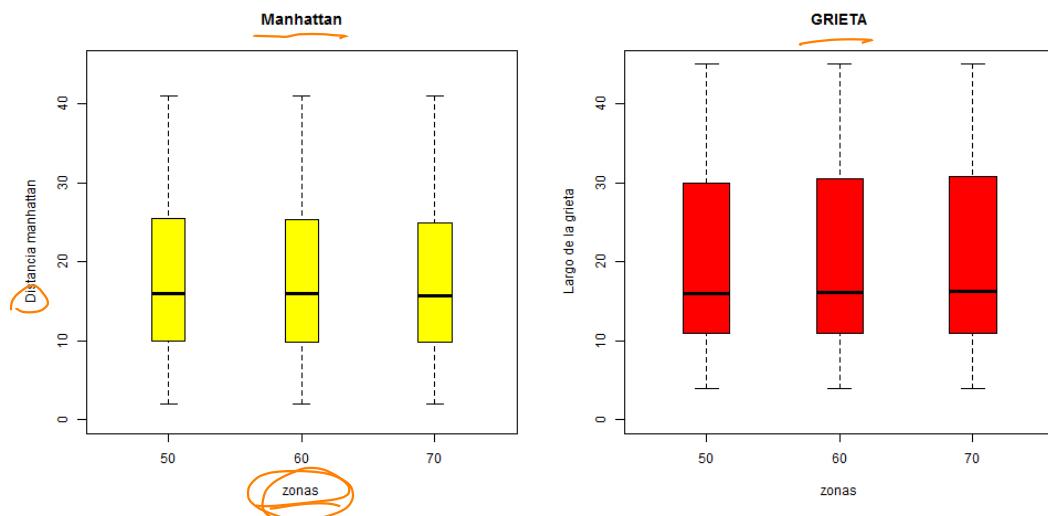


Figura 4: Distancias de grietas y distancia Manhattan

Debido a la poca variación de la cantidad de semillas en la matriz, no se puede observar claramente la disminución de la distancia manhattan y el largo de las grietas generadas.

De la misma manera se observó la influencia de la cantidad de semillas y la generación de grietas en las distintas matrices, ver figura 5 y 6.

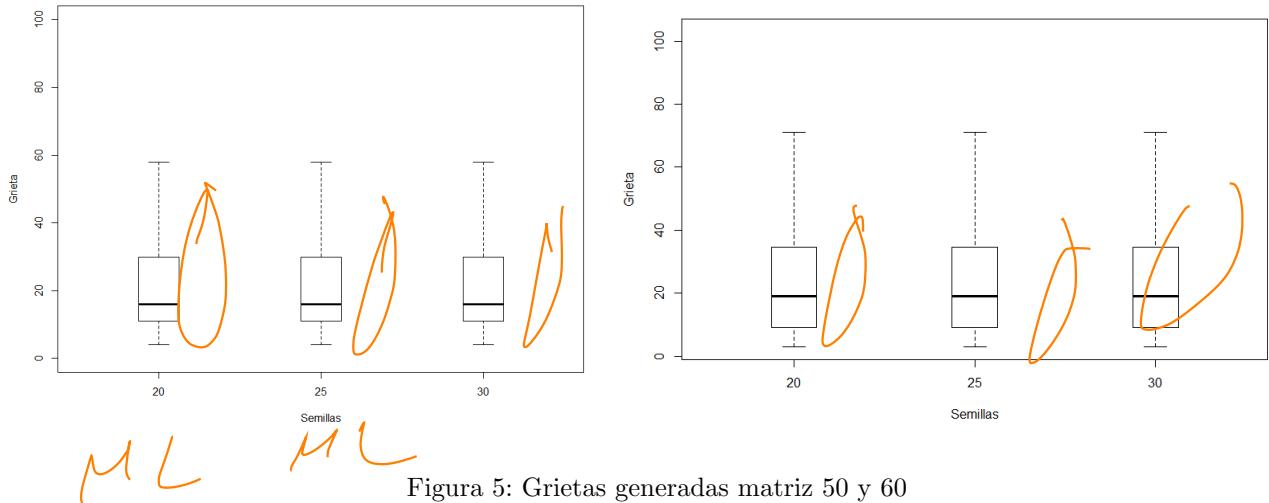


Figura 5: Grietas generadas matriz 50 y 60

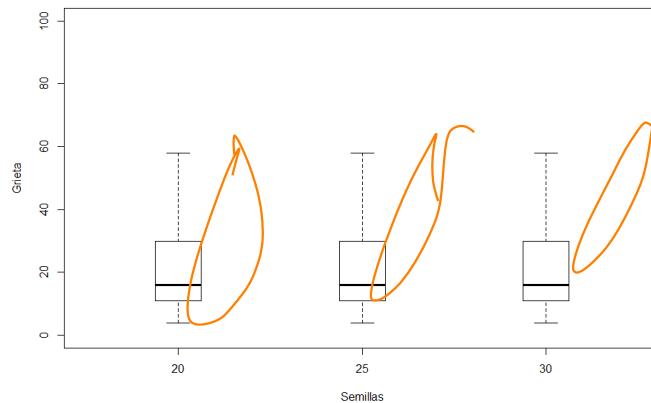


Figura 6: Grietas matriz 70

Debido a la baja variación en los parámetros, las diferencias son mínimas en las figuras. Lo cuál abre paso a una examinación extra con parámetros distintos.

Al final se realizó un análisis de varianza (ANOVA):

```
anova<-data.frame(datos)
stacked_groups<-stack(anova)
stacked_groups
resultadosa<-aov(values ~ ind, data = stacked_groups)
summary(resultadosa)
```

Arrojandonos los siguientes resultados

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
ind	4	14407296	3601824	3813	<2e-16 ***
Residuals	15995	15109818	945		

Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 . 1

discontin

3. Conclusiones

Conforme aumenta el número de semillas las grietas tienden a aumentar de tamaño, la presencia de un ambiente con mayor número de fronteras propiciara la propagación de las grietas con mayor facilidad. Con un número bajo en variables las diferencias son muy pocas, en comparación con lo que ya se ha reportado con parámetros mayores, tanto en cantidad de semillas como en largo de matrices. Entre mayor sea la matriz y menor cantidad de semillas se coloquen, la grieta avanzará con mayor dificultad.

Referencias

- [1] Jorge Mendoza. Diagramas de voronoi, 2018. URL <https://sourceforge.net/projects/simulacionenr/>.
- [2] SATU ELISA SCHAEFFER. Diagramas voronoi, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.

Diagramas de Voronoi

Abraham Azael Morales Juárez 1422745

6 de junio de 2019

1. Objetivos

Examinar el efecto del número de semillas y el tamaño de la zona en la distribución de los largos de las grietas que se forman, tomando en cuenta el número de celdas que contiene la grieta y la segunda es la mayor distancia Manhattan entre la grieta y el borde del cuadro [2].

2. Metodología y resultados

Se colocó una cantidad k de semillas al azar en una matriz que representaba un material, posteriormente se produjo en ellas la presencia de diagramas Voronoi desarrolladas a partir de la ubicación de las semillas. Para poder realizar esta práctica se hace referencia a un código previamente reportado [1].

```
1 tamano <- c(40, 50, 60)
2 semillas <- c(15,20,25)
3
4 for (n in tamano){
5   print(paste("matriz ", n))
6   for(k in semillas){
7     print(paste("total semillas ", k))
8     limite <- n
9     zona <- matrix(rep(0, n * n), nrow = n, ncol = n)
10    x <- rep(0, k) # almacenar las coordenadas "x" de las semillas
11    y <- rep(0, k) # coordenadas "y" de las semillas
12
13    for (semilla in 1:k) {
14      while (TRUE) { # hasta que hallamos una posicion vacia para la semilla
15        fila <- sample(1:n, 1)
16        columna <- sample(1:n, 1)
17        if (zona[fila , columna] == 0) {
18          zona[fila , columna] = semilla
19          x[semilla] <- columna
20          y[semilla] <- fila
21          break
22        }
23      }
24    }
25    suppressMessages(library(doParallel))
26    registerDoParallel(makeCluster(detectCores() - 1))
27    celdas <- foreach(p = 1:(n * n), .combine=c) %dopar% celda(p)
28    stopImplicitCluster()
29    voronoi <- matrix(celdas, nrow = n, ncol = n, byrow=TRUE)
30    rotate <- function(x) t(apply(x, 2, rev))
31    png("p4s.png")
32    par(mar = c(0,0,0,0))
33    image(rotate(zona), col=rainbow(k+1), xaxt='n', yaxt='n')
34    graphics.off()
35    png("p4c.png")
36    par(mar = c(0,0,0,0))
37    image(rotate(voronoi), col=rainbow(k+1), xaxt='n', yaxt='n')
38    graphics.off()
39
40  suppressMessages(library(doParallel))
```

```

41 registerDoParallel(makeCluster(detectCores() - 1))
42 largos <- foreach(r = 1:200, .combine=c) %elopar% propaga(r)
43 stopImplicitCluster()
44 summary(largos)
45 }
46 }
47 tab <- matrix(resultados, nrow = 3200, ncol=2, byrow = TRUE)
48 tab <- as.data.frame(tab)
49 names(tab) <- c("grieta", "manhattan")
50 replicas <- (rep(1:200, 16))
51 mat <- c(rep(50, 1066), rep(60, 1067), rep(70, 1067))
52 p <- c(rep(20, 213), rep(25, 213), rep(30, 214))
53 semillas <- rep(p, 5)
54
55 datos<-data.frame()
56 datos<-cbind(mat, semillas, replicas, tab)
57 dm<-largos(datos, method="manhattan")
58 cluster1<-hclust(dm)
59 boxplot(dm, col="blue", main="caliz", outline=F, boxwex=0.35)
60 manhattan<-dm
61 resultados <- c(manhattan, largos)

```

La cantidad de semillas se colocó de forma uniforme en intervalos enteros ($1:k$), (figura 1).

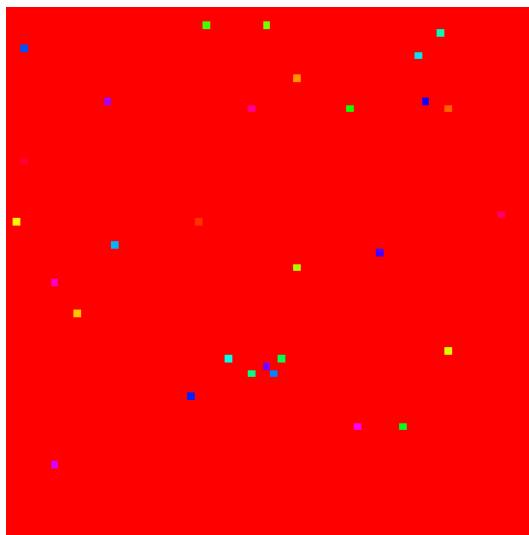


Figura 1: Semillas

Después se determinó que espacios vacíos corresponden al espacio de cada semilla, en otras palabras, para cada posición que vale cero, se calculó la distancia Manhattan entre las semillas, (figura 2).

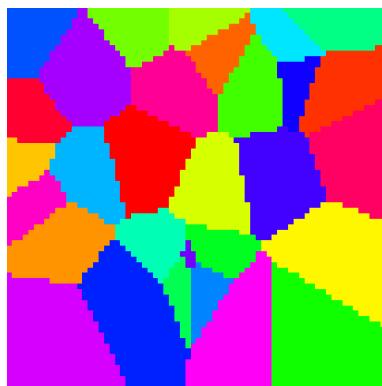


Figura 2: Celdas Voronoi

Estas celdas representan núcleos que posteriormente pasará a un proceso de cristalización en un material y se provoca una grieta en ese material, (figura 3), la cual se propaga con mayor velocidad entre las fronteras de los núcleos y viaja dificultosamente por el interior de la celda. Además, se toma en cuenta que no hay propagación preferencial por parte de la grieta.

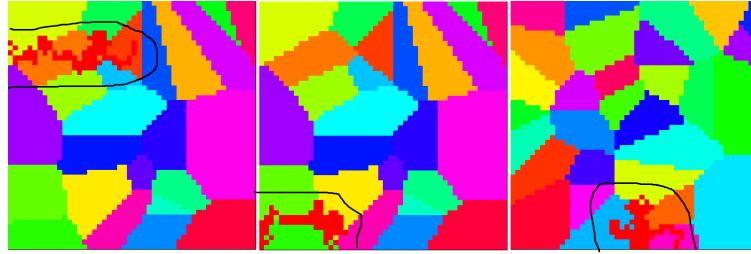


Figura 3: Grietas formadas en el material debido al crecimiento de las semillas

Para el desarrollo de la práctica se consideraron diferentes parámetros: tamaño de matriz, 50×50 , 60×60 y 70×70 , cantidad de semillas k , 20, 25, 30. En la figura 4, se tienen las distancias Manhattan de cada una de las matrices, además se observa influencia de la cantidad de semillas aplicada en la generación de grietas, también que a pesar de la variación que se aplicó no hubo diferencia significante en el largo de las grietas.

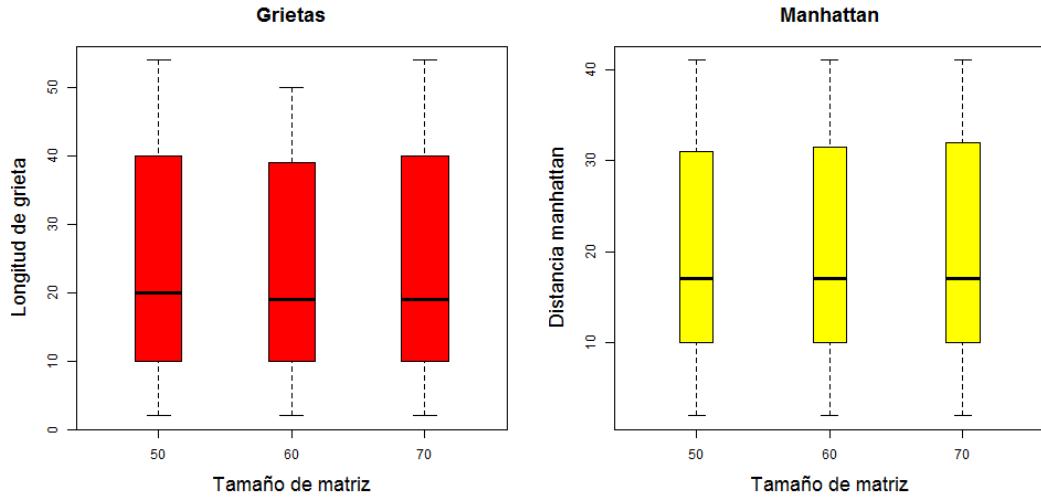


Figura 4: Distancias de grietas y distancia Manhattan

Debido a la poca variación de la cantidad de semillas en la matriz, no se puede observar claramente la disminución de la distancia Manhattan y el largo de las grietas generadas.

De la misma manera se observó la influencia de la cantidad de semillas y la generación de grietas en las distintas matrices, (figura 5 – 6).

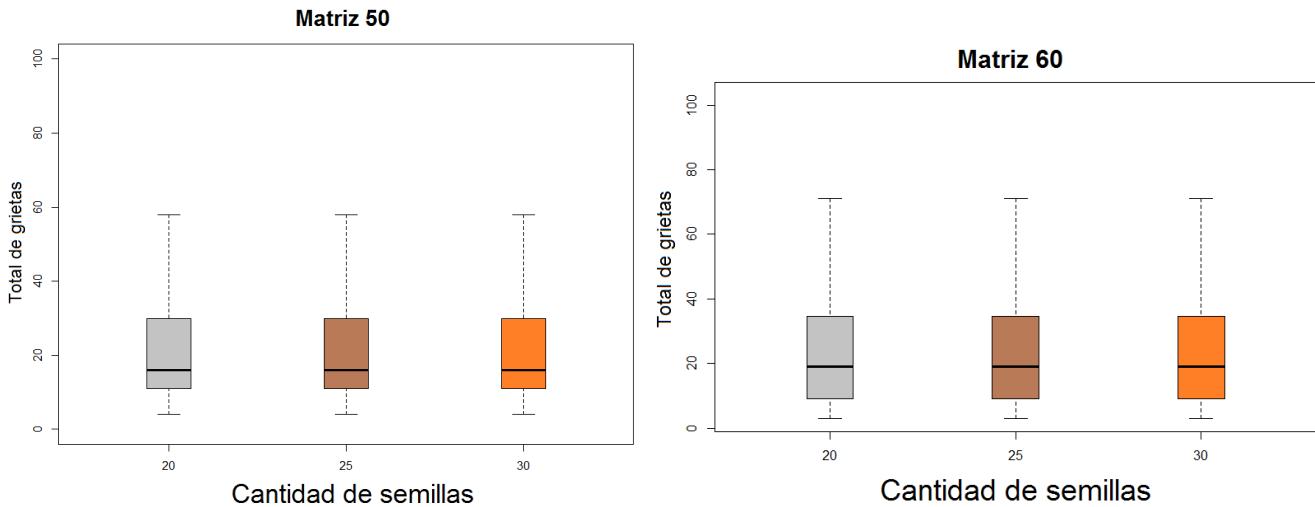


Figura 5: Cantidad de grietas generadas en la matriz de 50 – 60.

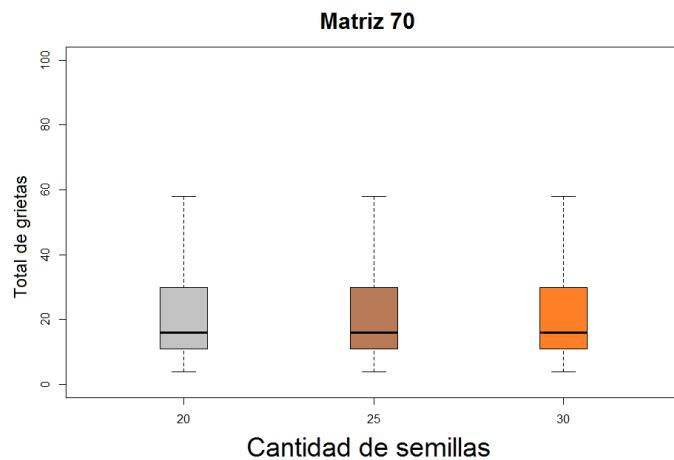


Figura 6: Cantidad de grietas generadas en la matriz de setenta.

Debido a la baja variación en los parámetros, las diferencias son mínimas en las figuras. Lo cuál abre paso a una examinación extra con parámetros distintos.

Al final se realizó un análisis de varianza (ANOVA):

```

1 anova<-data.frame(datos)
2 stacked_groups<-stack(anova)
3 stacked_groups
4 resultadosa<-aov(values ~ ind, data = stacked_groups)
5 summary(resultadosa)

```

Arrojando los siguientes resultados:

	Df	Sum Sq	Mean Sq	F value	Pr(>F)						
ind	4	14407296	3601824	3813	<2e-16 ***						
Residuals	15995	15109818	945								
Signif. codes:	0	***	0.001	**	0.01	*	0.05	.	0.1	"	1

Al realizar esta prueba arroja que tiene un valor inferior de 0,05 de $\text{Pr}(<\text{F})$, indicando que hay poca diferencia entre las medias del largo de las grietas. Para poder concluir que si hubo diferencias entre las medias se requeriría un análisis con más datos para poder compararlos.

3. Conclusiones

Conforme aumenta el número de semillas las grietas tienden a aumentar de tamaño, la presencia de un ambiente con mayor número de fronteras propiciara la propagación de las grietas con mayor facilidad. Con un número bajo en variables las diferencias son muy pocas, en comparación con lo que ya se ha reportado con parámetros mayores, tanto en cantidad de semillas como en largo de matrices. Entre mayor sea la matriz y menor cantidad de semillas se coloquen, la grieta avanzará con mayor dificultad.

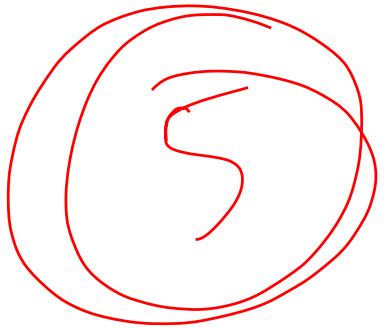
Referencias

- [1] Yessica Reyna Fernandez. Diagramas de voronoi, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/>.
- [2] Satu Elisa Schaeffer. Diagramas de voronoi, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.

Método Monte-Carlo

Abraham Azael Morales Juárez 1422745

26 de febrero de 2019



1. Introducción

El método Montecarlo permite resolver problemas matemáticos mediante simulación de variables aleatorias y es idóneo para situaciones en las cuales algún valor o alguna distribución no se conocen y resulta complicado de determinar de forma analítica [1].

2. Objetivos

Determinar el tamaño de muestra que se requiere para obtener el valor estimado de la integral de Wolfram Alpha, con una precisión de hasta 7 decimales.

sí se

3. Metodología y resultados

Para la realización de la práctica el código base fue proporcionado Wolfram [1], de el cual se muestra la parte necesaria para el entendimiento de está práctica.

```
inicio <- 0
final <- -inicio
paso <- 0.25
x <- seq(inicio, final, paso)
f <- function(x) { return(1 / (exp(x) + exp(-x))) }
png("p5f.png")
plot(x, (2/pi) * (1/(exp(x)+exp(-x))))
lines(x, (2/pi) * (1/(exp(x)+exp(-x))), type="l")
graphics.off()
suppressMessages(library(distr))
print((pi / 2) * integral)
```

El código se modificó para agregar la cantidad de muestras necesarias con la finalidad de acercarse al valor obtenido por Wolfram Alpha. Los tamaños de muestra fueron, 100, 1,000, 10,000, 100,000. Estas determinan la cantidad de variables a tomar en cuenta en cada repetición, se llevan a cabo 30 repeticiones para cada tamaño de muestra.

Además se modificó m para poder incluir un ciclo "for" para los diferentes tamaños de muestra, un fragmento de este código puede observarse a continuación.

*Next +
lem*

```

for (Tamano in c(100,1000,10000,100000)) {
  print(paste("Tamano", Tamano))
  for (repeticiones in 1:30) {
    montecarlo <- foreach(i = 1:cuantos, .combine=c) %dopar% parte()
    integral <- sum(montecarlo) / (cuantos * Tamano)
    resultados <- (pi / 2) * integral
    diferencia <- (Wolfram - resultados)
    lista <- rbind(lista, c(repeticiones, Tamano, Wolfram, resultados, diferencia))
  }
}
stopImplicitCluster()

```

Tras realizar las iteraciones los resultados se graficaron para poder ser analizados, (figura 1).

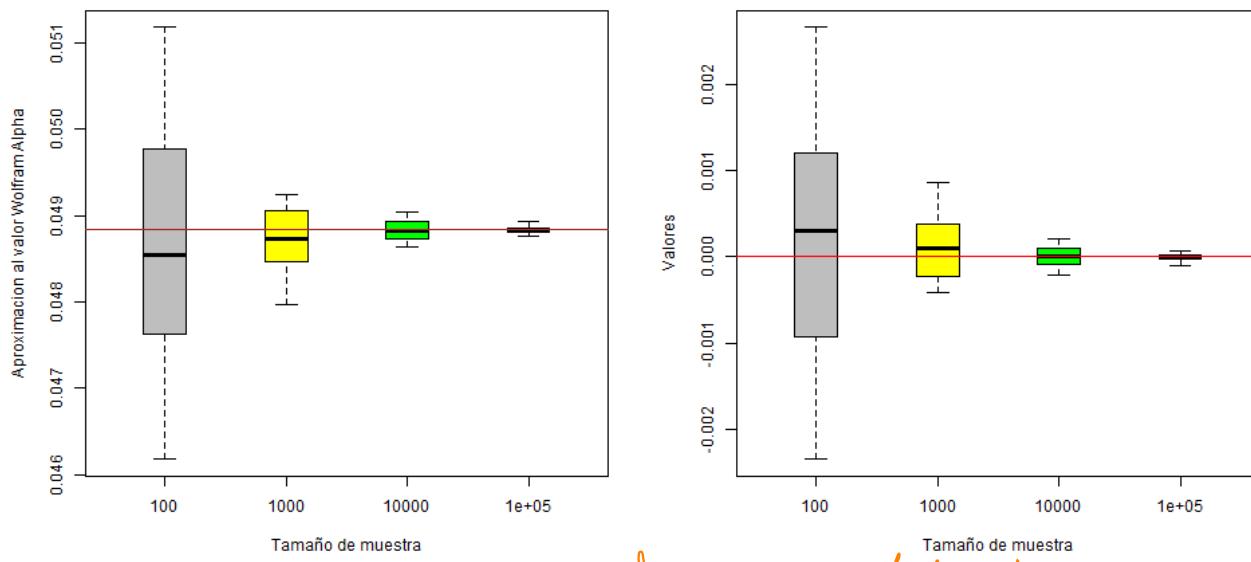


Figura 1: Se puede observar una línea roja, que es un valor que se predeterminó, uno siendo el valor de Wolfram Alpha colocado en la gráfica izquierda que nos indica la variación del resultado de cada muestra. En la gráfica derecha el valor predeterminado se colocó en cero, para observar la variación en los decimales en cada muestra.

En la figura se pueden observar los resultados del método Montecarlo en comparación con la aproximación Wolfram Alpha. Además, se observa claramente como el aumento del tamaño de las muestras influye el acercamiento al valor calculado, es decir, disminuye la diferencia en el resultado de los análisis conforme la muestra sea mayor. También se observa las diferencias en los resultados en decimales, y viendo que a partir del tamaño de muestra de 100,000 los resultados cada vez aumenta su precisión, indicando que para cualquier muestra que sea mayor, estas tenderán a igualar al obtenido por Wolfram Alpha.

4. Conclusiones

Con el método Montecarlo se logró obtener resultados aproximados al valor de Wolfram Alpha y entre mayor sea el número de dígitos que contenga la muestra aumentara la precisión del valor calculado.

Referencias

- [1] Satu Elisa Schaeffer. Método montecarlo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p5.html>.

Método Monte-Carlo

Abraham Azael Morales Juárez 1422745

5 de junio de 2019

1. Introducción

El método Monte-Carlo permite resolver problemas matemáticos mediante simulación de variables aleatorias y es idóneo para situaciones en las cuales algún valor o alguna distribución no se conocen y resulta complicado de determinar de forma analítica [2].

2. Objetivos

Determinar el tamaño de muestra que se requiere para obtener el valor estimado de la integral de Wolfram Alpha, con una precisión de hasta siete decimales.

3. Metodología y resultados

Para la realización de la práctica el código base fue proporcionado [2], y de cual fue modificado en base a uno previamente reportado [1].

```
1 f <- function(x) { return(1 / (exp(x) + exp(-x))) }
2 suppressMessages(library(distr))
3 g <- function(x) { return((2 / pi) * f(x)) }
4 generador <- r(AbscontDistribution(d = g)) # creamos un generador
5 muestra <- generador(50000) # sacamos una muestra
6 desde <- 3
7 hasta <- 7
8 pedazo <- 300
9 cuantos <- 100
10 Wolfram <- 0.04883505
11 poblacion <- c(100,1000,10000,20000,50000)
12 lista <- data.frame()
13 lista1 <- data.frame()
14 parte <- function() {
15   valores <- generador(Tamano)
16   return(sum(valores >= desde & valores <= hasta))
17 }
```

El código se modificó para agregar la cantidad de muestras necesarias con la finalidad de acercarse al valor obtenido por Wolfram Alpha. Los tamaños de muestra fueron, 100, 1,000, 10,000, 20,000, 50,000. Estas determinan la cantidad de variables a tomar en cuenta en cada repetición, se llevan a cabo treinta repeticiones para cada tamaño de muestra.

Además se modificó para poder incluir un ciclo for para los diferentes tamaños de muestra, un fragmento de este código puede observarse a continuación.

```

1 for(Tamano in pedazo) {
2   for (cuantos in poblacion) {
3     print(paste("Tamano" , Tamano))
4     for (repeticiones in 1:30) {
5       suppressMessages(library(doParallel))
6       cluster <- makeCluster(2)
7       registerDoParallel(cluster)
8       montecarlo <- foreach(i = 1:cuantos , .combine=c) %dopar % parte()
9       stopCluster(cluster)
10
11      integral <- sum(montecarlo) / (cuantos * Tamano)
12      resultados <- (pi / 2) * integral
13      aca <- c(resultados , cuantos , as.integer(Tamano))
14      print(cuantos)
15
16      listal <- rbind(listal , aca)
17
18      diferencia <- (Wolfram-resultados)
19      lista <- rbind(lista , c(repeticiones , Tamano, Wolfram, resultados , diferencia))
20    }
21  }
22
23 }
24 colnames(listal) <- c("val" , "Muestras" , "esp")
25 listal$esp <- as.integer(listal$esp)
26 listal$Muestras <- as.factor(listal$Muestras)

```

Tras realizar las iteraciones los resultados se graficaron para poder ser analizados, (figura 1).

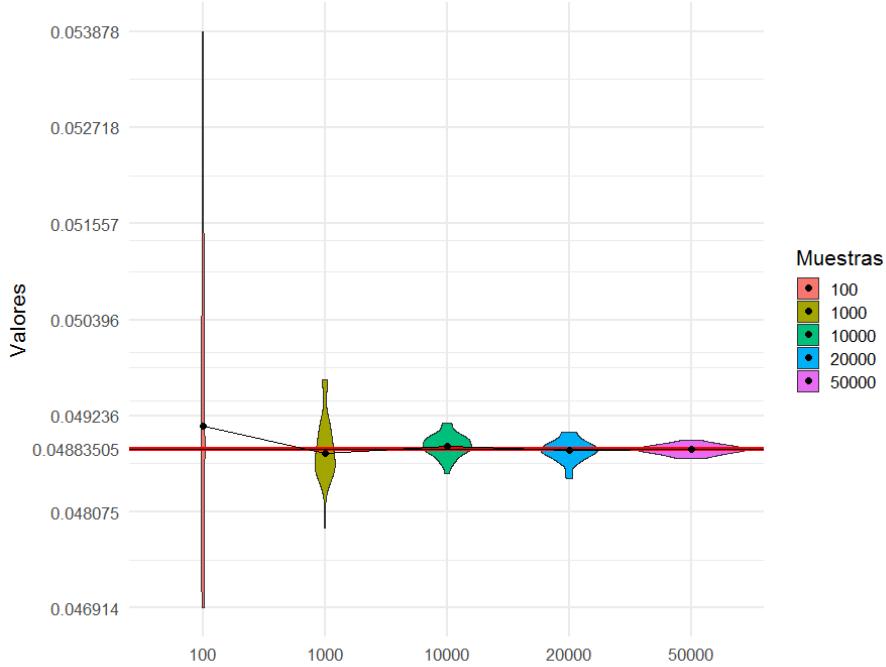


Figura 1: Aproximación de los tamaños de muestra al valor de Wolfram Alpha (línea roja).

En la figura 1, se pueden observar los valores obtenidos de cada análisis comparándolos con el valor de Wolfram Alpha. Así mismo, se observa que conforme los tamaños de muestra son mayores estos se aproximan al valor de Wolfram. También se puede observar que a partir de análisis con muestras grandes los resultados serán cada vez más precisos.

4. Conclusiones

Con el método Montecarlo se logró obtener resultados aproximados al valor de Wolfram Alpha y entre mayor sea el número de dígitos que contenga la muestra aumentara la precisión del valor calculado.

Referencias

- [1] Astrid González. Método montecarlo, 2018. URL [https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/
Exercise_5/](https://sourceforge.net/p/gla-sim/exercises/HEAD/tree/Exercise_5/).
- [2] Satu Elisa Schaeffer. Método montecarlo, 2019. URL [https://elisa.dyndns-web.com/teaching/comp/par/p5.
html](https://elisa.dyndns-web.com/teaching/comp/par/p5.html).

Sistema multiagente

Abraham Azael Morales Juárez 1422745

4 de marzo de 2019

1. Introducción

Este es un sistema donde hay un conjunto de entidades, que interaccionan entre sí, con estados internos propios pero que son capaces de reaccionar y cambiar según cambien sus vecinos. Se manejan 3 estados internos, un modelo conocido como SIR, susceptibles, infectados y recuperados [2].

~~Leop~~

2. Objetivos

Al momento de crear a los agentes, hacer que algunos de ellos ya se encuentren vacunados contra la enfermedad y que desde el inicio formen parte del estado R.

Observar el efecto ocasionado por agregar individuos vacunados desde el inicio del análisis de la epidémia.

3. Resultados

Los parámetros y un fragmento del código [2] modificado y usado en la práctica están a continuación. Se observó la influencia en la cantidad de individuos (agentes) infectados y la correlación de individuos ya vacunados y que por ende ya no podrían contagiar la enfermedad, debido a que para que exista un contagio se necesita estar dentro de un rango de cercanía de otro agente previamente infectado.

```
1 l <- 1.5
2 n <- 150          #Cantidad de agentes
3 pi <- 0.8          #Probabilidad de infección
4 pr <- 0.02         #Probabilidad recuperarse
5 v <- 1 / 30        #Velocidad del agente
6 pv <- seq(0,1,0.1) #Probabilidad de vacunación y formar parte de R
7 r <- 0.1
8 agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(),
9 estado = character())
10 for (i in 1:n) {
11   e <- "S"
12   if (runif(1) <= pi) {
13     e <- "I"
14     if (runif(1) <= pr) {
15       e <- "R"
16     }
17   }
18   agentes <- rbind(agentes, data.frame(x = runif(1, 0, 1), y = runif(1, 0, 1),
19                               dx = runif(1, -v, v), dy = runif(1, -v, v),
20                               estado = e))
21   levels(agentes$estado) <- c("S", "I", "R")
22 }
```

Estos agentes vacunados se les proporciona el estado de R desde un comienzo cuando fueron creados y así observar la evolución del contagio. De la Figura 1 – 5 se observa el aumento de agentes recuperados en cada uno de los porcentajes de vacunación que se usaron e incluso como la infección tomaba más presencia nuevamente en algunos agentes en etapas más avanzadas de recuperación, esto se observa en las iteraciones de 0.5 en adelante. También, a diferencia de lo que se

reportó en otras prácticas [1] donde variaban otros valores, la variación presentada en nuestro caso es mayor. Estos resultados corresponden al finalizar el experimento. Más adelante se mencionarán los porentajes mayores de infección, es decir, al comienzo de la epidemia.

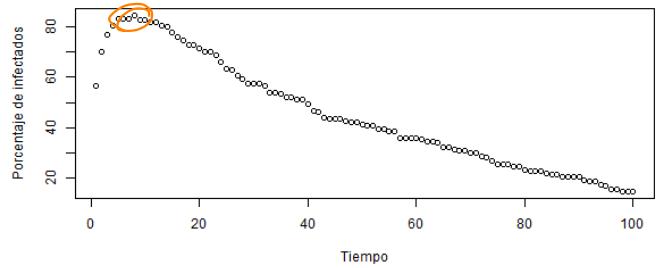
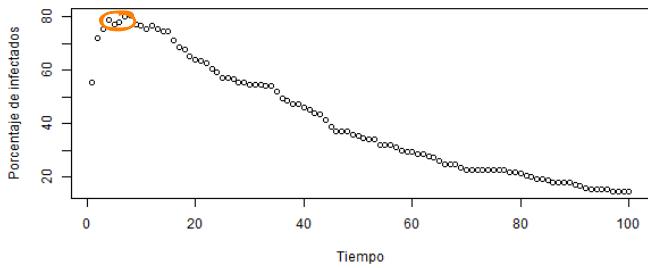


Figura 1: Porcentaje agentes infectados a través del tiempo con $pv = 0$ y 0.1 .

R_u
\$p_v\$

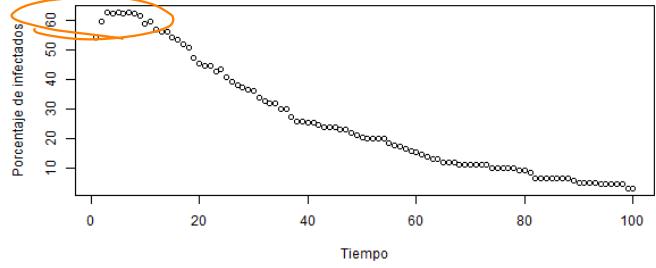
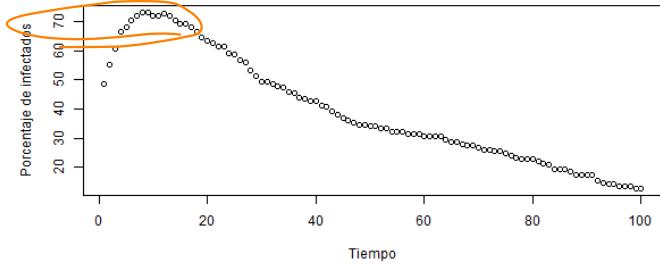


Figura 2: Porcentaje agentes infectados a través del tiempo con $pv = 0.2$ y 0.3 .

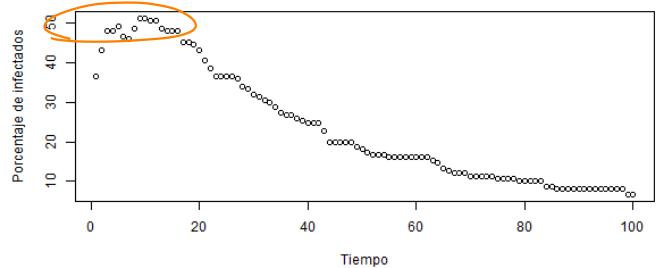
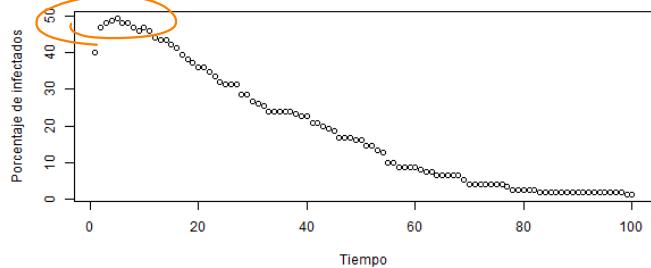


Figura 3: Porcentaje agentes infectados a través del tiempo con $pv = 0.4$ y 0.5 .

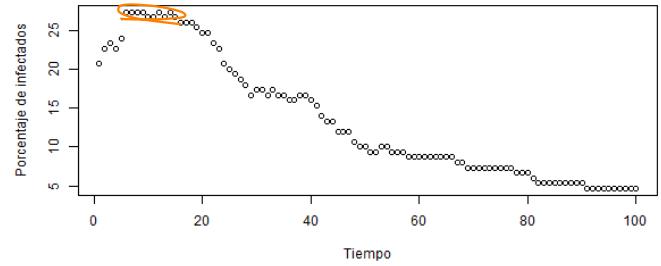
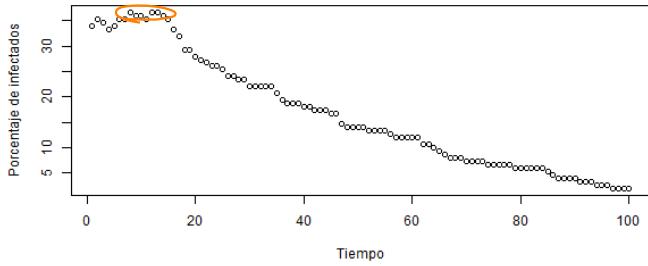


Figura 4: Porcentaje agentes infectados a través del tiempo con $p_v = 0.6$ y 0.7 .

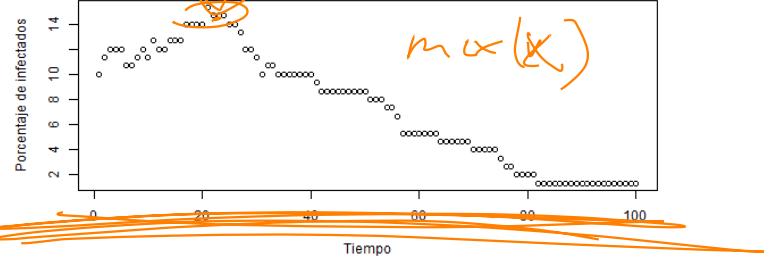
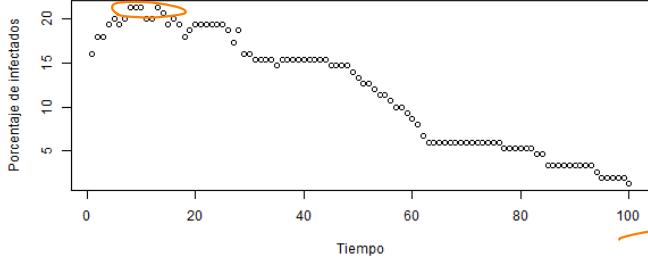
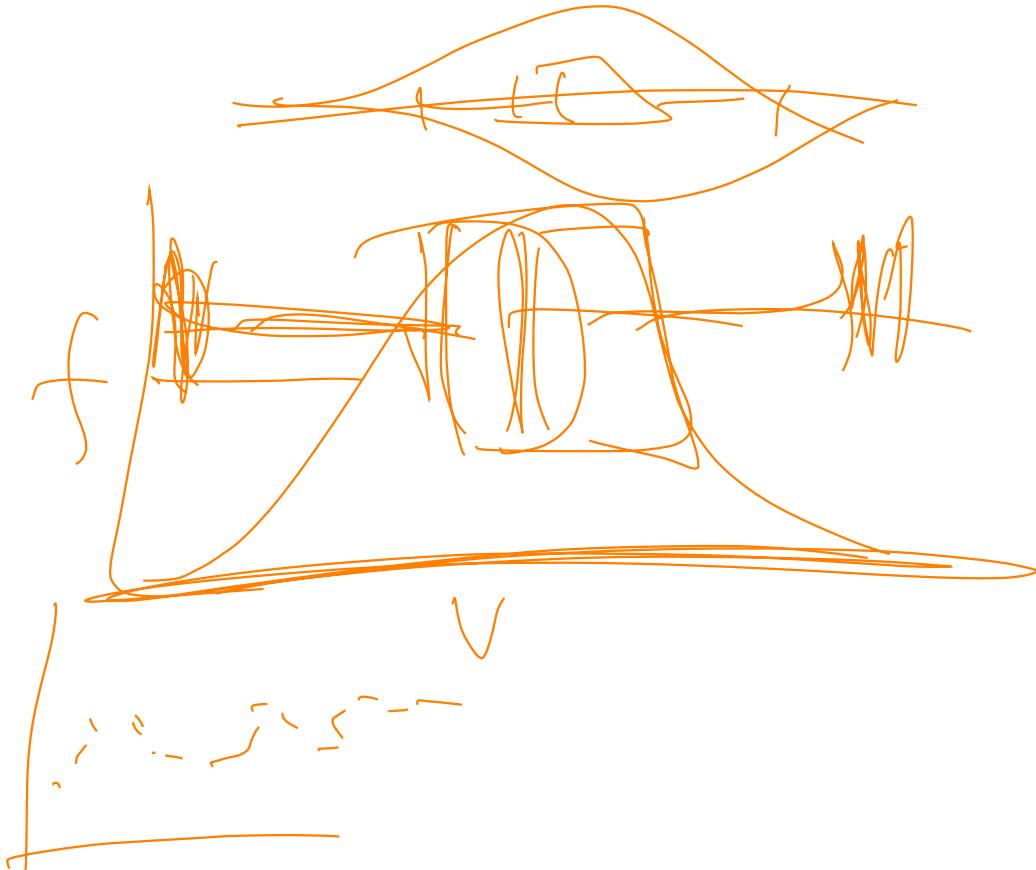


Figura 5: Porcentaje agentes infectados a través del tiempo con $p_v = 0.8$ y 0.9 .

En la Figura 6, se observa el comportamiento de la distribución de los agentes en cada iteración de p_v usado. En la iteración 0.4 mostró un comportamiento más homogéneo entre los estados, en el cual la presencia de infección, susceptibilidad y recuperación de los agentes es muy similar, posteriormente le siguen las iteraciones de 0.6 y 0.3.



Agentes infectados, susceptibles y recuperados

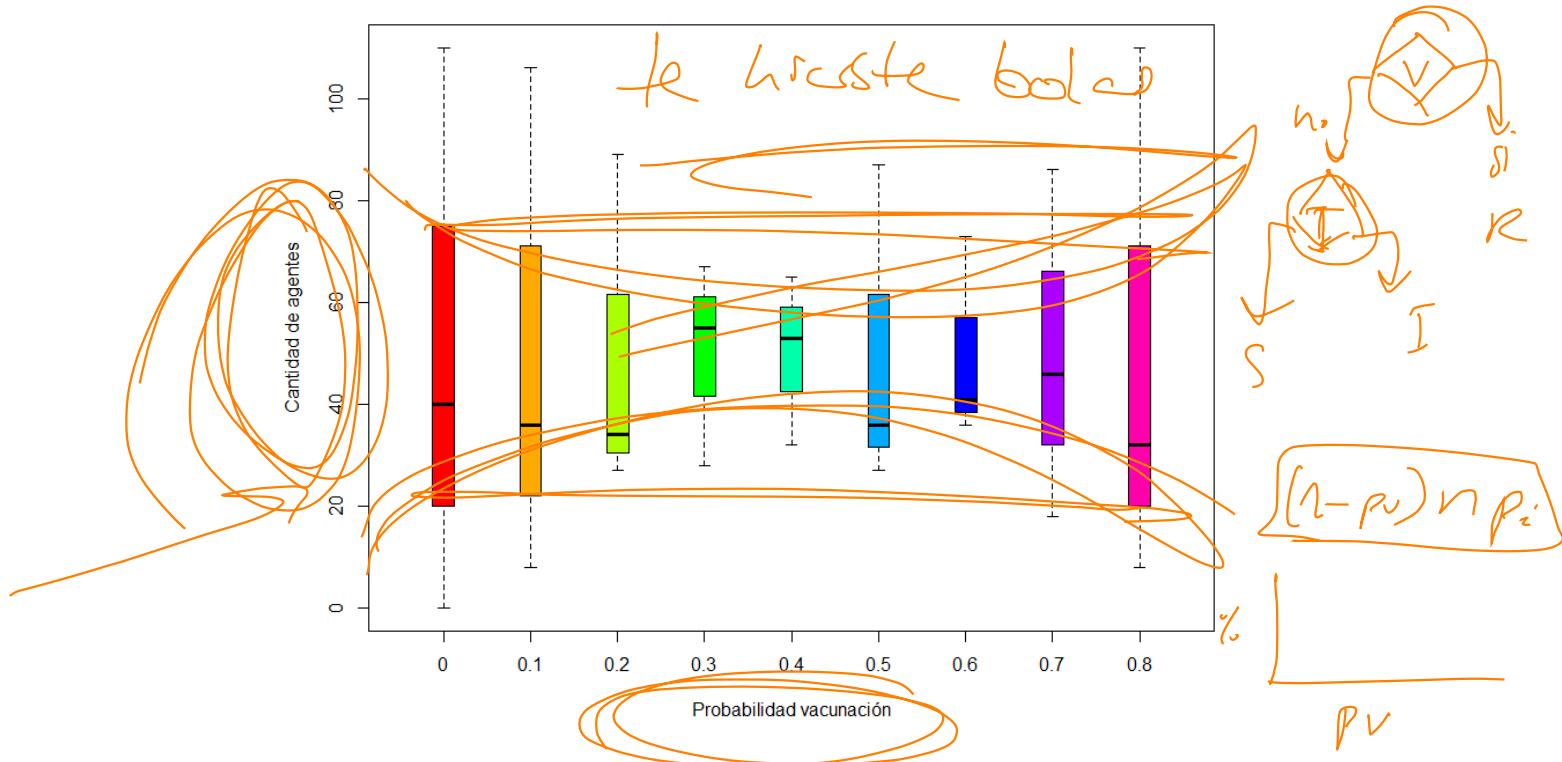


Figura 6: Comparación de la distribución de agentes con respecto a la probabilidad de estar vacunados, en donde se observa que la mejor distribución de la población se encuentra en el rango 0.4.

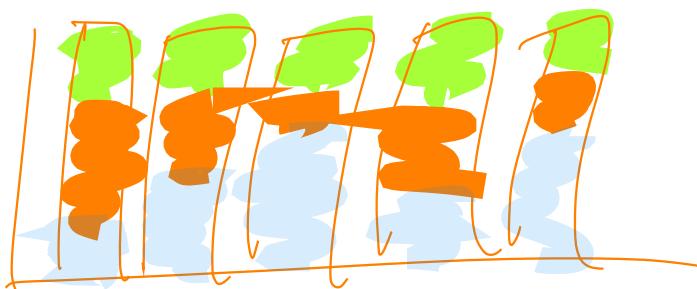
Además, en la Figura 6 se observa el cambio gradual que ocurre, principalmente en las 5 primeras iteraciones, y es en donde se comienza la redistribución del número de agentes en los diferentes estados y comienzan a colocarse en el estado R, esto se puede interpretar ya que los primeros resultados corresponden a una mayor cantidad de agentes infectados debido a que no había muchos vacunados, y esto se interpreta como una mayor distribución al estado I, el cual después de la iteración 0.4 se puede entender que cambia favoreciendo al del estado R.

4. Conclusiones

La iteración que mostró una distribución más homogénea fue la de 0.4. El comportamiento de la epidemia se mantuvo constante en las primeras 5 iteraciones, posterior a esta su variación es mayor, como resultado la epidemia tiene casos en donde disminuye su frecuencia de aparición, sin embargo, ésta puede aumentar.

Referencias

- [1] Alexia Ibarra. Práctica 6: sistema multiagente. *Universidad Autónoma de Nuevo León*, 2013.
- [2] Satu Elisa Schaeffer. Sistema multiagente, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.



Sistema multiagente

Abraham Azael Morales Juárez 1422745

5 de junio de 2019

1. Introducción

Este es un sistema donde hay un conjunto de entidades, que interaccionan entre sí, con estados internos propios pero que son capaces de reaccionar y cambiar según cambien sus vecinos. Se manejan tres estados internos, un modelo conocido como SIR, susceptibles, infectados y recuperados [2].

2. Objetivos

Al momento de crear a los agentes, hacer que algunos de ellos ya se encuentren vacunados contra la enfermedad y que desde el inicio formen parte del estado R.

Observar el efecto ocasionado por agregar individuos vacunados desde el inicio del análisis de la epidémia.

3. Resultados

El código que fue proporcionado [2] fue modificado, tomando como base un código ya reportado en clase [4]. Se observó que la cantidad de agentes infectados y su correlación con agentes vacunados afectaría el porcentaje de una determinada población para poder contagiar a otros individuos, debido a que se requiere estar cercano a un individuo enfermo.

```
1 l <- 1.5
2 n <- 150
3 pi <- 0.05
4 pr <- 0.02
5 v <- 1 / 30
6 pv <- seq(0, 1, 0.1)
7 r <- 0.1
8 tmax <- 100
9
10 maximos <- c()
11
12 for(i in pv) {
13   for (rep in 1:30) {
14
15     agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(), estado = character(
16       0))
17     for (i in 1:n) {
18       e <- "S"
19       if (runif(1) <= pi) {
20         e <- "I"
21         if (runif(1) <= pv) {
22           e <- "R"
23         }
24       agentes <- rbind(agentes, data.frame(x = runif(1, 0, 1), y = runif(1, 0, 1),
25                                             dx = runif(1, -v, v), dy = runif(1, -v, v),
26                                             estado = e))
27       levels(agentes$estado) <- c("S", "I", "R")
28     }
29     epidemia <- integer()
30     digitos <- floor(log(tmax, 10)) + 1
31     for (tiempo in 1:tmax) {
```

```

32  infectados <- dim(agentes[agentes$estado == "I",]) [1]
33  epidemia <- c(epidemia, infectados)
34  if (infectados == 0) {
35    break
36 }

```

A los agentes vacunados se les colocó en el estado R (recuperados) desde el momento de su creación y así poder observar la evolución de la pandemia. En la figura 1 se observa el comportamiento de la infección en sus máximas probabilidades con respecto a la probabilidad de estar vacunados desde el inicio, se puede observar claramente como la infección disminuye cuando la probabilidad de estas vacunados aumenta.

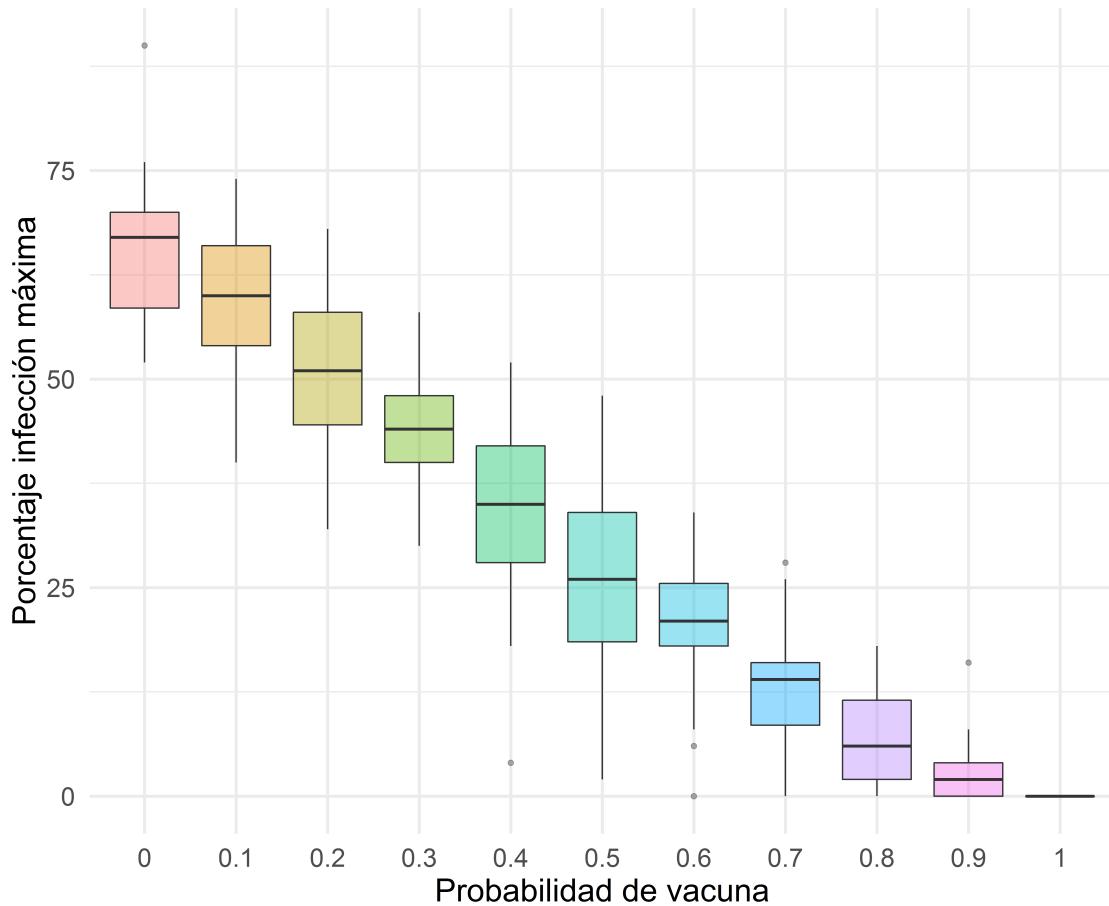


Figura 1: Porcentaje de cambio de infectados máximos alcanzados contra la probabilidad de agentes vacunados.

Además, se observan los porcentajes, tanto de infección como de vacuna, donde hay un 75 porciento como máximo de infectados y de cero a uno con pasos de 0,01 para los vacunados. Este experimento se realizó con un total de cincuenta agentes, 30 repeticiones, porcentaje de infección de 0,05 y un porcentaje de recuperación de 0,02.

Por último se calcula si los datos son estadísticamente significativos, esta prueba se realizó con la prueba **kruskall-wallis** la cuál arroja los siguientes datos:

- ji-cuadrada = 136.13.
- p-value = 1.86×10^{31}

Lo que indica que los datos si son estadísticamente significativos.

4. Conclusiones

Se puede concluir con los parámetros actuales que el comportamiento de infectados es de manera lineal decreciente conforme aumenta la probabilidad de vacunados.

Referencias

- [1] Edson Raúl Cepeda Márquez. Práctica 6: sistema multiagente. 2019. URL <https://sourceforge.net/p/systemssimulation/activity/?page=0&limit=100#5cac46b2f0d34738b97a97c1>.
- [2] Satu Elisa Schaeffer. Sistema multiagente, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.

Búsqueda Local

Abraham Azael Morales Juárez 1422745

19 de marzo de 2019

1. Introducción

Se generarán funciones capaces de detectar máximos locales. Primeramente, se realizará una función en una dimensión, donde se minimiza la función $f(x)$. Será necesario realizar una función bidimensional $g(x, y)$. Esta nueva función debe poder identificar los máximos locales. Es posible implementar métodos eficientes [2] para identificar los valores determinados por las máximas locales de una función [1].

2. Objetivos

Realizar combinaciones que proporcionan ocho posiciones posibles vecinos.

Graficar en tres dimensiones.

Crear una visualización animada de cómo proceden 15 réplicas simultáneas de la búsqueda encima de una gráfica de proyección plana.

3. Resultados

Se modificó el código final que fue proporcionado (1) para agregar la función $g(x, y)$.

```
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 }
```

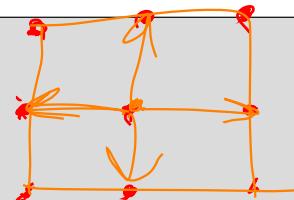
Además se modificó el código para la generación de vecinos, donde se colocó la opción para poder generar 8 distintas posiciones que pudiese elegir para posicionarse. Se coloca el fragmento del código a continuación.

```
1 replica <- function(t) {  
2   curr <- runif(2, low, high)  
3   best <- curr  
4   for (tiempo in 1:t) {  
5     delta_x <- runif(-1, 0, step)  
6     delta_y <- runif(-1, 0, step)  
7     up <- best + delta_y  
8     down <- best - delta_y  
9     right <- best + delta_x  
10    left <- best - delta_x  
11  
12    upright <- best + c(delta_x, delta_y)  
13    upleft <- best + c(-delta_x, delta_y)  
14  
15    downright <- best + c(delta_x, -delta_y)  
16    downleft <- best + c(-delta_x, -delta_y)  
17  }
```

for

incomplete

c.f. Edson



Los resultados obtenidos se pueden observar en la Figura 1 y 2, donde una línea verde representa el valor máximo real del experimento, y se puede observar como la línea va cambiando según los pasos que se estén dando. Hubo un inconveniente, no se pudo hacer que el código marcará los puntos del análisis en rojo.

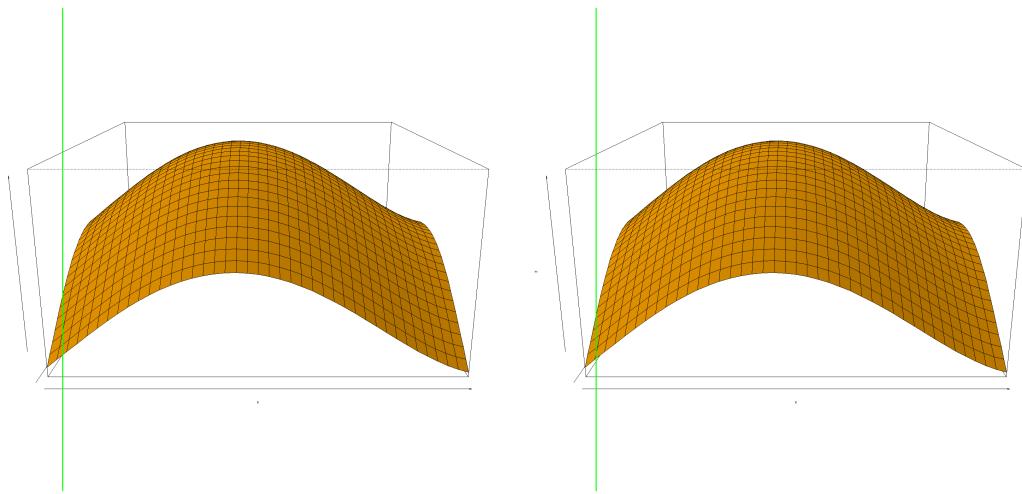


Figura 1: Lado izquierdo representa el análisis con 100 pasos y 1000 pasos el de la derecha

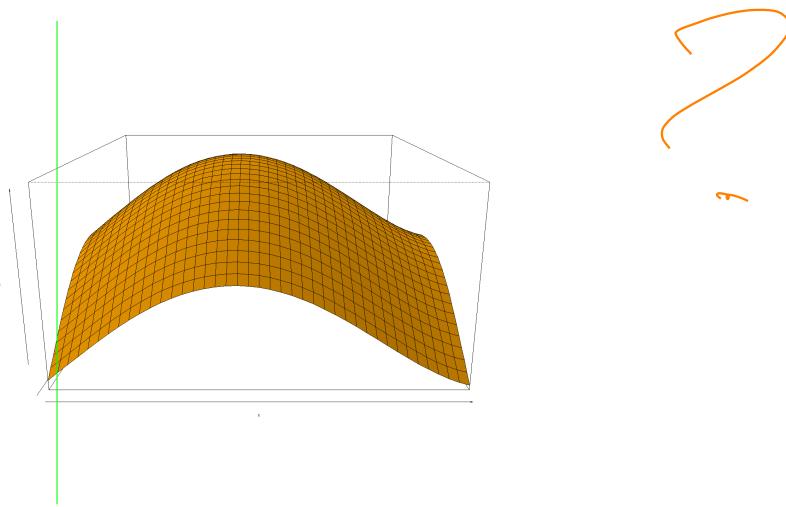


Figura 2: Análisis de 10000 pasos

Las gráficas se obtuvieron con el siguiente código.

```

1 suppressMessages(library(doParallel))
2 registerDoParallel(makeCluster(detectCores() - 1))
3
4 for (pot in 2:4) {
5   tmax <- 10^pot
6   resultados <- foreach(i = 1:replicas, .combine=c) %dopar% replica(tmax)
7   png(paste("grafica_3D", tmax, ".png", sep=""), width=1500, height=1500)
8   x <- seq(low, high, step)
9   y <- x
10  z <- outer(x, y, g)
11  persp(x,y,z, col='orange', expand = 0.5, shade=0.2) ← " lattice / image / heatmap "
12  valores <- f(resultados)
13  points(resultados, valores, pch=1600, col="red")
14  mejor <- which.max(valores)
15  abline(v =resultados [mejor], col="green", lwd=3) ← " points "
16  graphics.off()
17 }
18 stopImplicitCluster()

```

No se pudo realizar el mapa de calor (heatmap), el código es el siguiente:

```

1 suppressMessages(library(doParallel))
2 registerDoParallel(makeCluster(detectCores() - 1))
3
4 for(graf in 2:4) {
5   tmax <- graf
6   resultados <- foreach(i = 1:tmax, .combine=rbind, .replicate=tmax) %>
7     valores %>%
8       summarise(across(everything(), sum))
9   x <- seq(low, high, step)
10  y <- x
11  z <- outer(x, y, g)
12  dimnames(z) <- list(x, y)
13  d <- melt(z)
14  names(d) <- c("x", "y", "z")
15  points(resultados, valores, pch=16, col="red")
16  png("valores.png", width=900, height=900)
17  levelplot(z ~ x * y, data = d, main = "")
18  dev.off()
19 }
20
21 stopImplicitCluster()

```

4. Conclusiones

NA

R

Referencias

- [1] Alexia Ibarra. Práctica 7: Búsqueda local. *Universidad Autónoma de Nuevo León*, 2013.
- [2] Satu Elisa Schaeffer. Búsqueda local, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.

Búsqueda Local

Abraham Azael Morales Juárez 1422745

6 de junio de 2019

1. Introducción

Se generarán funciones capaces de detectar máximos locales. Primeramente, se realizará una función en una dimensión, donde se minimiza la función $f(x)$. Será necesario realizar una función bidimensional $g(x, y)$. Esta nueva función debe poder identificar los máximos locales. Es posible implementar métodos eficientes [2] para identificar los valores determinados por las máximas locales de una función [1].

2. Objetivos

Maximizar la función bidimensional de $g(x, y)$ en donde buscaremos y encontraremos el valor máximo otorgado a g por medio del método de búsqueda local.

3. Resultados

Primeramente, para poder identificar cual es el máximo valor local se modifica el código [2], el que también analiza de manera paralela las réplicas del experimento, donde cada una tendrá una cantidad determinada de pasos para obtener el punto máximo de la función, las modificaciones se agregaron al código base tomando como referencia otro reporte ya entregado [1].

Se coloca las restricciones de los valores los cuales están entre -3 a 3 . Cuando x, y tengan una posición estos tendrán ocho vecinos de los cuales se analizaran y tomará posición cuando localice el valor con mayor valor, y así continuará el análisis y la función g en cada uno obtendrá el mayor valor.

```
1 library("latticeExtra")
2 resultados<-data.frame()
3 g <- function(x, y) {
4   return(((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y^2 - 20 * y)/100)
5 }
6
7 low <- -3
8 high <- 3
9 step <- 0.25
10 replicas <- 100
```

```
1 replica <- function(t) {
2   curr <- runif(, min = low, max = high)
3   best <- curr
4   best1 <- curr1
5   for (tiempo in 1:t) {
6     delta <- runif(1, 0, step)
7     delta1 <- runif(1, 0, step)
8     left <- curr - delta
9     right <- curr + delta
10    up <- curr1 + delta1
```

```

12     down <- curr1 - delta1
13     if (right > 3){
14       right<-right - 6
15     } else {
16       right <- right
17     }
18     if (left < (-3)){
19       left <- left + 6
20     } else{
21       left <- left
22     }

```

Una vez comenzado el análisis con el código se evalúan los posibles vecinos escogiendo el mejor a cada paso, esto se puede observar en la figura 1 donde varios ejemplos de pasos de las réplicas para aproximarse al punto máximo (punto rojo).

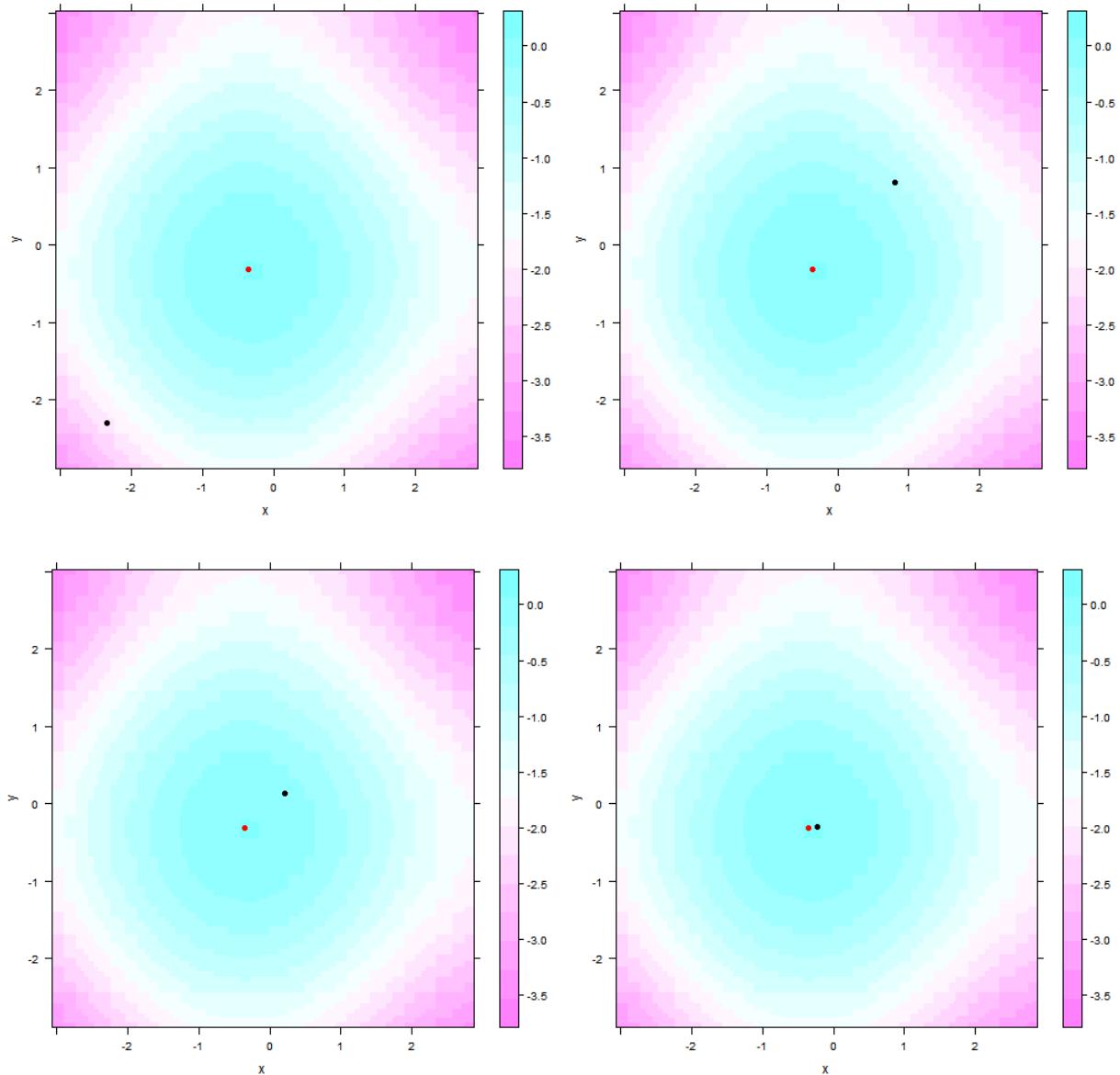


Figura 1: Valores seleccionados al maximizar la función g donde se puede observar el valor máximo obtenido por la simulación y que se marca con un pequeño punto oscuro y estos valores pueden estar cerca del centro o alejados debido a que el análisis comienza desde un punto al azar sobre la matriz.

```

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   mapa <- levelplot(z ~ x * y, data = funcion)
10  coordt <- xyplot(resultados[, 1] ~ resultados[, 2], pch=16, col="black")
11  coordb <- xyplot(funcion[mejor, 1,] ~ funcion[mejor,2,], pch=16, col="red")
12  mapas <- mapa + as.layer(coordt) + as.layer(coordb)
13  png(paste("p7-", tmax, ".png", sep="")), width=500, height=500)
14  print(mapas)
15  graphics.off()
16  for (pasos in 1:(dim(resultados)[1])){
17    puntos <- xyplot(resultados[pasos, 1] ~ resultados[pasos, 2], pch=16, col="black")
18    MV <- xyplot(funcion[mejor, 1,] ~ funcion[mejor,2,], pch=16, col="red")
19    mapapunto <- mapa + as.layer(MV) + as.layer(puntos)
20    png(paste("p7-", tmax, "-", pasos, ".png", sep="")), width=500, height=500)
21    print(mapapunto)
22    graphics.off()
23  }
24 }
```

4. Conclusiones

Mediante el método de búsqueda local se puede optimizar una función. Mediante gráficos de alta calidad se puede interpretar mejor estos tipos de análisis, donde se busca encontrar el máximo valor para la función. A su vez, entre mayor sea la generación de pasos cada vez se obtiene un valor más grande que el anterior encontrado.

Referencias

- [1] Yessica Reyna Fernández. Práctica 7: Búsqueda local. 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/>.
- [2] Satu Elisa Schaeffer. Práctica 7: Búsqueda local, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.

Búsqueda Local

3

Abraham Azael Morales Juárez 1422745

25 de marzo de 2019

1. Introducción

Está práctica se trata sobre fenómenos de coalescencia y fragmentación [2]. La coalescencia es la unión de dos elementos conformando uno sólo. Lo que ocasionara que la partícula con el paso del tiempo aumente de tamaño gradualmente [1]. Esto es relevante en distintos ámbitos, como, por ejemplo, en el tratamiento de aguas residuales con el uso de filtros.

2. Objetivos

Optimizar el código para que se ejecute en el menor tiempo por medio de paralelización.

Realizar una gráfica para comparar los tiempos obtenidos de ambos métodos y observar si el ahorro es significativo para diferentes combinaciones de k y n .

3. Resultados

Para la realización de esta práctica se tomaron en cuenta distintos parámetros:

Réplicas cincuenta.

Número de cúmulos k de 5,000, 10,000 y 15,000.

Número de partículas n de 500,000, 1,000,000 y 1,500,000.

La paralelización se realizó en la unión de los cúmulos y en la fragmentación de estos. Se coloca sólo un fragmento del código como ejemplo.

```
1 library(testit)
2 library(parallel)
3 paralelo <- data.frame()
4 k <- 15000
5 n <- 1500000
6 cluster <- makeCluster(detectCores() - 1) crece
7
8 romperse <- function(tam, cuantos) {
9   romper <- round(rotura(tam) * cuantos)
10  resultado <- rep(tam, cuantos - romper)
11  if (romper > 0) {
12    for (cumulo in 1:romper) {
13      t <- 1
14      if (tam > 2) {
15        t <- sample(1:(tam-1), 1)
16      }
17      resultado <- c(resultado, t, tam - t)
18    }
19  }
20  assert(sum(resultado) == tam * cuantos)
21  return(resultado)
22 }
```

Los resultados obtenidos son distintos a los esperados, debido a como se observa en la Figura 1, si hay diferencia en los tiempos de ambos métodos pero en la primera iteración, que corresponde a $k=5,000$ y $n=500,000$, se observa que el método de paralelización tarda más en comenzar y debido a que son pocos datos los que se están analizando, el método secuencial resultó ser el adecuado. Por otra parte, en la segunda y tercera iteración, se observa una reducción en los tiempos de la paralelización como se esperaba, esto es debido al aumento en los datos analizados.

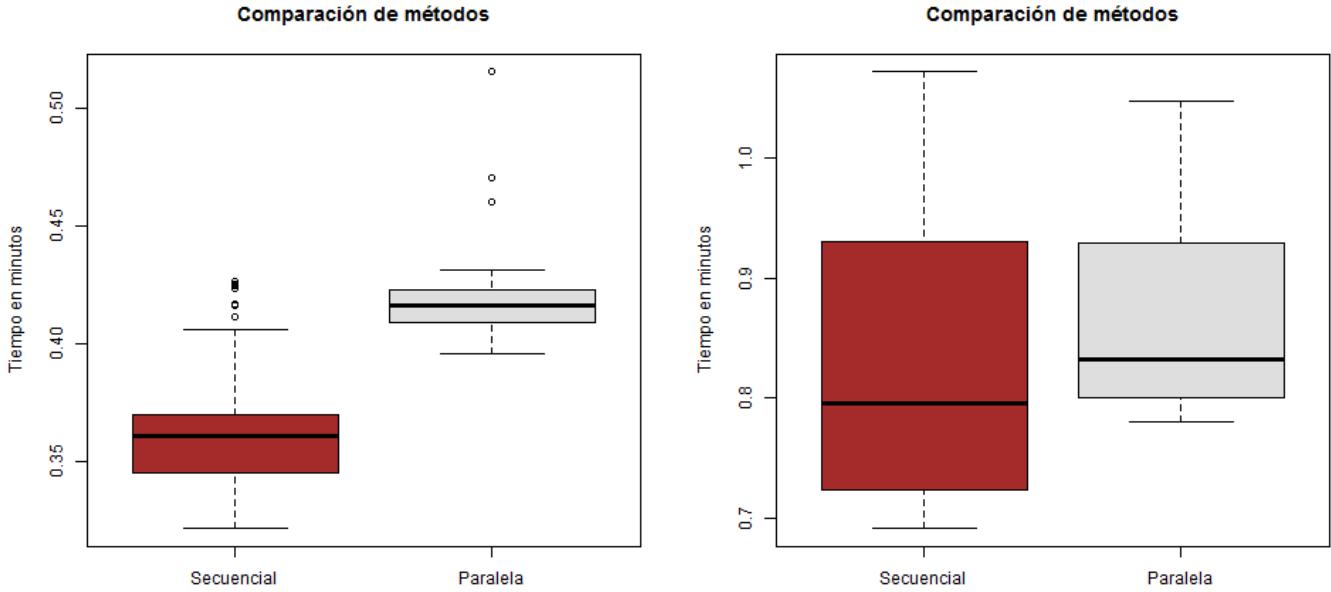


Figura 1: Comparación de tiempos de las iteraciones $k=5,000$ y de $n=500,000$ (lado izquierdo) y de $k=10,000$ y $n=1,000,000$ (lado derecho).

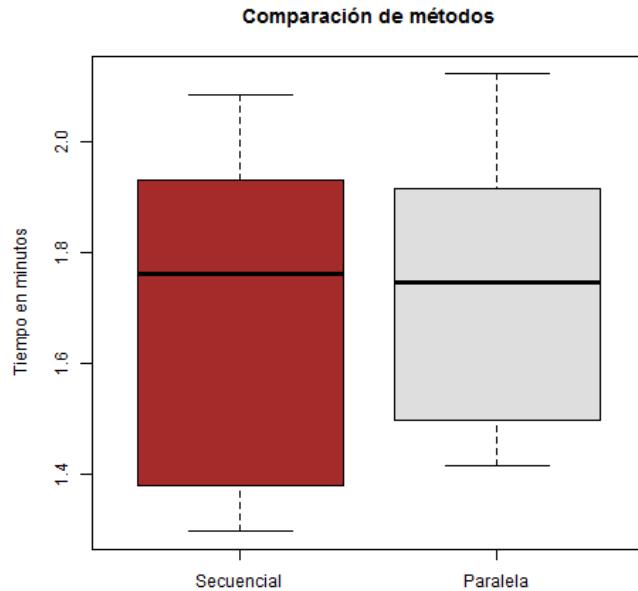


Figura 2: Comparación de tiempos de la iteración $k=15,000$ y de $n=1,500,000$.

Además se agregan fragmentos del código que corresponde al principio y al final del análisis.

```
1 for (replicas in 1:50) {  
2     inicial1 <- Sys.time()  
3     originales <- rnorm(k)  
4     cumulos <- originales - min(originales) + 1  
5     cumulos <- round(n * cumulos / sum(cumulos))  
6     assert(min(cumulos) > 0)  
7     diferencia <- n - sum(cumulos)
```

↑ Paralelo? ↓

```
1 termino1 <- Sys.time()  
2     paralelo <- rbind(paralelo , c(termino1-inicial1 , k, n, replicas))  
3 }  
4 names(paralelo) <- c("Tiempo" , "Moleculas" , "Cumulos" , "Replicas")  
5 paralelo$Nivel <- "P"  
6 totaltiempos <- rbind(secuencial , paralelo)  
7 png("Comparacion de metodos.png")  
8 boxplot(totaltiempos$Tiempo~totaltiempos$Nivel , main = "Comparacion de metodos" , ylab = "Tiempo en  
     minutos" , col = c("brown" , "grey87") , names = c("Secuencial" , "Paralela"))  
9 graphics.off()
```

4. Conclusiones

El método de paralelización es efectivo y rápido en comparación del secuencial, sólo si la cantidad de datos es muy grande, lo contrario ocurre cuando los datos son pocos.

Se logró la paralelización del código y se demostró la importancia de realizar este método.

El método tarda en comenzar debido a que primeramente se dividen las tareas en todos los núcleos para posteriormente realizar el análisis.

Referencias

- [1] Alexia Ibarra. Práctica 8: Modelo de urnas. *Universidad Autónoma de Nuevo León*, 2018.
- [2] Satu Elisa Schaeffer. Práctica 8: Modelo de urnas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p8.html>.

Modelo de urnas

Abraham Azael Morales Juárez 1422745

6 de junio de 2019

1. Introducción

Está práctica se trata sobre fenómenos de coalescencia y fragmentación [2]. La coalescencia es la unión de dos elementos conformando uno sólo, ocasionando que la partícula con el paso del tiempo aumente de tamaño gradualmente [1]. Esto es relevante en distintos ámbitos, por ejemplo, en el tratamiento de aguas residuales con el uso de filtros.

2. Objetivos

- Optimizar el código para que se ejecute en el menor tiempo por medio de paralelización.
- Realizar una gráfica para comparar los tiempos obtenidos de ambos métodos y observar si el ahorro es significativo para diferentes combinaciones de k y n .

3. Resultados

Para la realización de esta práctica se tomaron en cuenta distintos parámetros:

- Réplicas treinta.
- Número de cúmulos k de 5,000, 10,000 y 15,000.
- Número de partículas n de 500,000, 1,000,000 y 1,500,000.

El código base proporcionado [2] fue modificado para realizar esta práctica, tomando como base otro código ya reportado [1]. La paralelización se realizó en la unión de los cúmulos y en la fragmentación de estos. Se coloca sólo un fragmento del código como ejemplo.

```
1 library(testit)
2 library(parallel)
3 paralelo <- data.frame()
4
5 suppressMessages(library(doParallel))
6 cluster <- makeCluster(detectCores() - 1)
7 registerDoParallel(cluster)
8 k <- c(5000, 10000, 15000)
9 n <- c(500000, 1000000, 1500000)
10 romperse <- function(tam, cuantos) {
11   romper <- round(rotura(tam) * cuantos) # independientes
12   resultado <- rep(tam, cuantos - romper) # los demás
13   if (romper > 0) {
14     for (cumulo in 1:romper) { # agregar las rotas
15       t <- 1
16       if (tam > 2) { # sample no jala con un solo valor
17         t <- sample(1:(tam-1), 1)
18       }
19       resultado <- c(resultado, t, tam - t)
20     }
21   }
22 }
```

```

21 }
22 assert(sum(resultado) == tam * cuantos) # no hubo perdidas
23 return(resultado)
24 }
25 romperse <- function(tam, cuantos) {
26   romper <- round(rotura(tam) * cuantos) # independientes
27   resultado <- rep(tam, cuantos - romper) # los demás
28   if (romper > 0) {
29     for (cumulo in 1:romper) { # agregar las rotas
30       t <- 1
31       if (tam > 2) { # sample no jala con un solo valor
32         t <- sample(1:(tam-1), 1)
33       }
34       resultado <- c(resultado, t, tam - t)
35     }
36   }
37 assert(sum(resultado) == tam * cuantos) # no hubo perdidas
38 return(resultado)
39 }
40
41 rotura <- function(x) {
42   return (1 / (1 + exp((c - x) / d)))
43 }
44
45 r <- function(i){
46   return(as.vector(romperse(as.numeric(freq[i, 1]), as.numeric(freq[i, 2]))))
47 }
48
49 unirse <- function(tam, cuantos) {
50   unir <- round(union(tam) * cuantos) # independientes
51   if (unir > 0) {
52     division <- c(rep(-tam, unir), rep(tam, cuantos - unir))
53     assert(sum(abs(division)) == tam * cuantos)
54     return(division)
55   } else {
56     return(rep(tam, cuantos))
57   }
58 }
59
60 union <- function(x) {
61   return (exp(-x / c))
62 }
63
64 u <- function(i){
65   return(as.vector(unirse(as.numeric(freq[i, 1]), as.numeric(freq[i, 2]))))
66 }

```

Los resultados obtenidos son los esperados, debido a como se observa en la figura 1, hay gran diferencia en los tiempos de ejecución entre ambos métodos donde es claro que a medida que va aumentando el número de partículas y cúmulos estos tiempos van aumentando, sin embargo, el método de paralelización es mejor en los tiempos que el secuencial en el análisis de todos los tamaños de muestras.

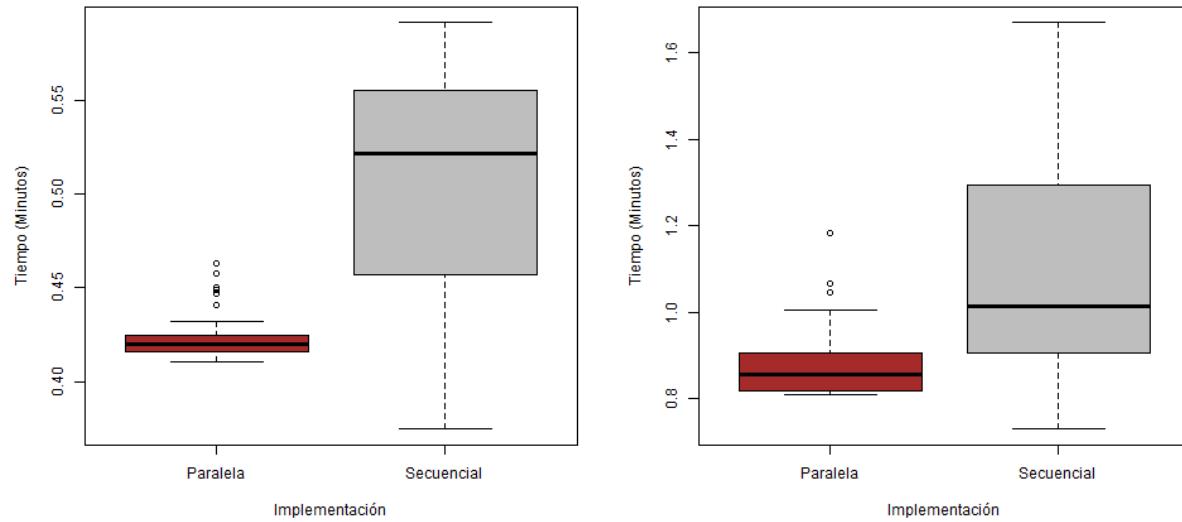


Figura 1: Comparación de tiempos de las iteraciones $k = 5,000$ y de $n = 500,000$ (lado izquierdo) y de $k = 10,000$ y $n = 1,000,000$ (lado derecho).

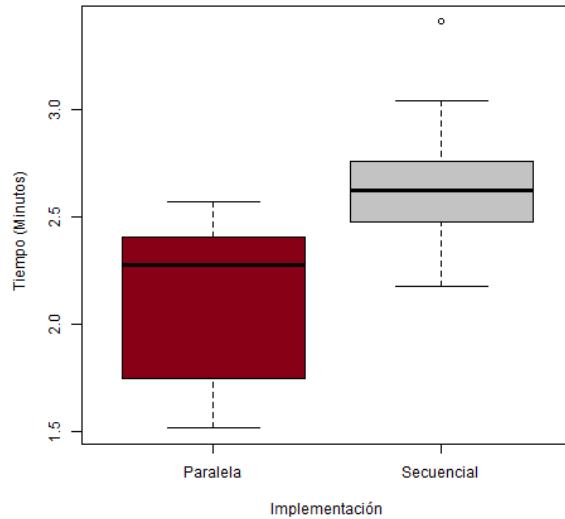


Figura 2: Comparación de tiempos de las iteraciones $k = 15,000$ y de $n = 1,500,000$

Además se agregan fragmentos del código que corresponde al principio y al final del análisis.

```

1 for (replicas in 1:30) {
2   inicial1<- Sys.time()
3   originales <- rnorm(k)
4   cumulos <- originales - min(originales) + 1
5   cumulos <- round(n * cumulos / sum(cumulos))
6   assert(min(cumulos) > 0)
7   diferencia <- n - sum(cumulos)

```

```

1 termino1 <- Sys.time()
2     paralelo <- rbind(paralelo , c(termino1-inicial1 , k, n, replicas))
3 }
4 names(paralelo) <- c("Tiempo" , "Moleculas" , "Cumulos" , "Replicas")
5 paralelo$Nivel <- "P"
6 totaltiempos <- rbind(secuencial , paralelo)
7 png("Comparacion de metodos.png")
8 boxplot(totaltiempos$Tiempo~totaltiempos$Nivel , main = "Comparacion de metodos" , ylab = "Tiempo (
    Minutos)" , col = c("brown" , "grey") , names = c("Paralela" , "Secuencial"))
9 graphics.off()

```

4. Conclusiones

El método de paralelización es efectivo y rápido en comparación del secuencial por lo tanto, es un método que se tiene que tomar en cuenta para poder aplicarlo a distintas áreas de estudio, donde se requiera el análisis de datos de manera rápida.

Se logró la parelización del código y se demostró la importancia de realizar este método.

Referencias

- [1] Yessica Reyna Fernández. Práctica 8: Modelo de urnas. 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/>.
- [2] Satu Elisa Schaeffer. Práctica 8: Modelo de urnas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p8.html>.

Interacciones entre partículas

5 -

Abraham Azael Morales Juárez 1422745

2 de abril de 2019

1. Introducción

En esta práctica se realizará un modelo simplificado para los fenómenos de atracción y repulsión, uno de estos fenómenos se le conoce fuerza electrostática [2]. En donde aplica la ley de Coulomb, partículas con cargas iguales se repelen y con cargas opuestas se atraen. Además de que su fuerza es proporcional a la diferencia de la magnitud de las cargas e inversamente proporcional a la distancia euclíadiana entre las partículas. Se crearán partículas que tendrán estas características de atracción o repulsión [1].

2. Objetivos

Agregar a cada partícula masa y generar que esta masa afecte las fuerzas de atracción.

Estudiar la distribución de las velocidades de las partículas y verificar que existe una relación entre los factores: velocidad, carga y masa.

3. Resultados

En la figura 1 se observan los resultados de las 50 partículas generadas donde el color representa la magnitud de la carga.

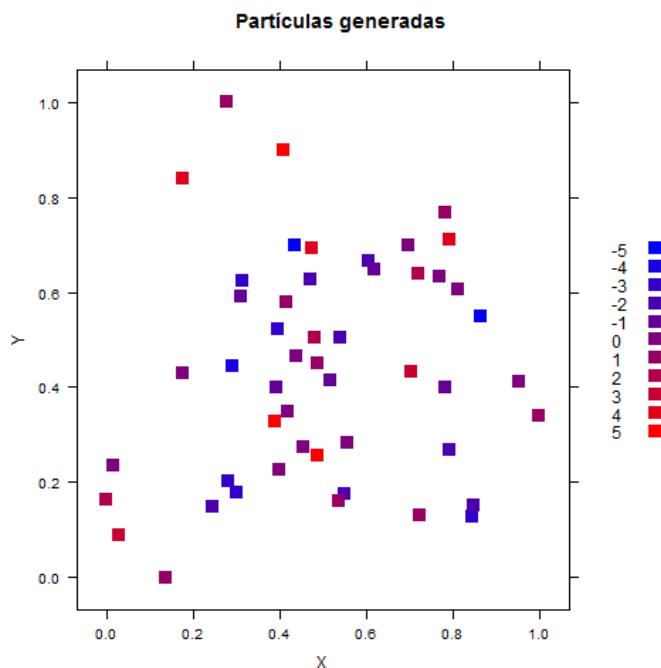


Figura 1: Partículas generadas.

En el código se agregaron los parámetros con los cuales se trabaja, entre ellos la variable nueva que es la masa.

```

1 n <- 50
2 p <- data.frame(x = rnorm(n), y=rnorm(n), c=rnorm(n), m=rnorm(n))
3 mmax <- max(p$m)
4 mmin <- min(p$m)
5 p$m <- ((p$m - mmin)*(p$m - mmin) / (mmax - mmin))+1
6 xmax <- max(p$x)
7 xmin <- min(p$x)
8 p$x <- (p$x - xmin) / (xmax - xmin) # ahora son de 0 a 1
9 ymax <- max(p$y)

```

Además se agregan las líneas del código donde estas muestran el efecto que tienen sobre las fuerzas de atracción entre las partículas así como la velocidad de estas.

```

1 fuerzas <- data.frame()
2 for (iter in 1:tmax) {
3   aux <- c()
4   f <- foreach(i = 1:n, .combine=c) %dopar% fuerza(i)
5   delta <- 0.02 / max(abs(f)) # que nadie desplace una paso muy largo
6   mf <- delta*f
7   p$x <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$x + delta * f[c(TRUE, FALSE)][i], 1), 0)
8   p$y <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$y + delta * f[c(FALSE, TRUE)][i], 1), 0)
9   for(t in seq(1, (length(mf)-1), by = 2)){
10     aux <- c(aux, (sum((mf[t])^2, (mf[t+1])^2))^(1/2))
11   }
12   fuerzas <- rbind(fuerzas, aux)
13   t1 <- paste(iter, "", sep="")
14   while (nchar(t1) < digitos) {
15     t1 <- paste("0", t1, sep="")
16   }

```

En la figura 2 y 3 se puede observar el comportamiento de estas partículas a lo largo del experimento, tomando como ejemplos distintas etapas para poder observar mejor el avance de estas.

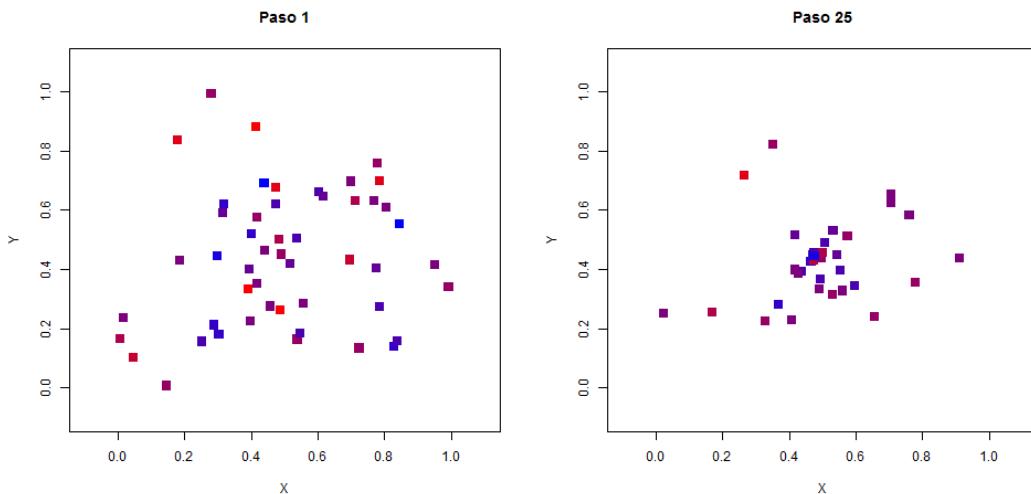


Figura 2: Simulación de atracción de las partículas en los pasos 1 y 25.

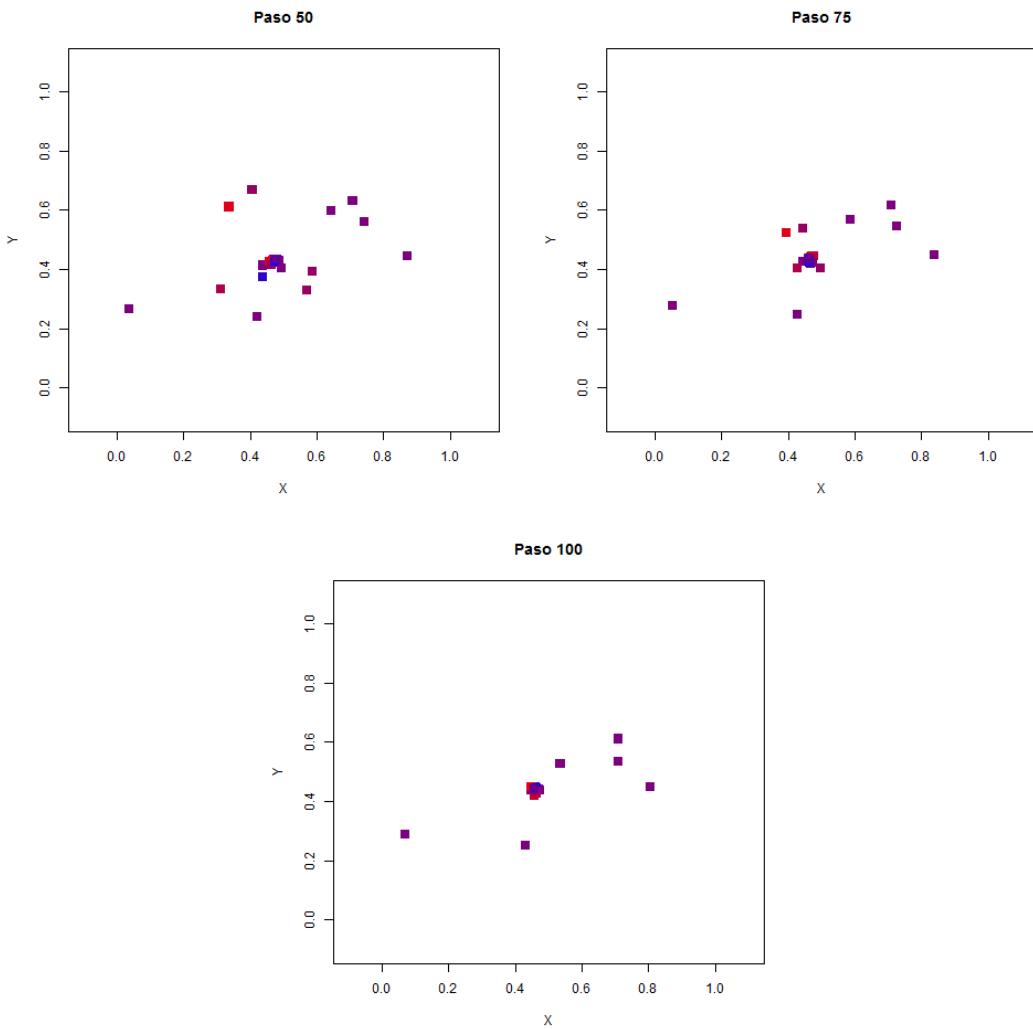


Figura 3: Simulación de atracción de las partículas en los pasos 50, 75 y 100.

De la figura 4 - 7 se muestran las velocidades de distintas secciones del experimento, y ~~estas observaciones que van obteniendo~~ que van obteniendo la misma velocidad conforme avanzan los pasos.

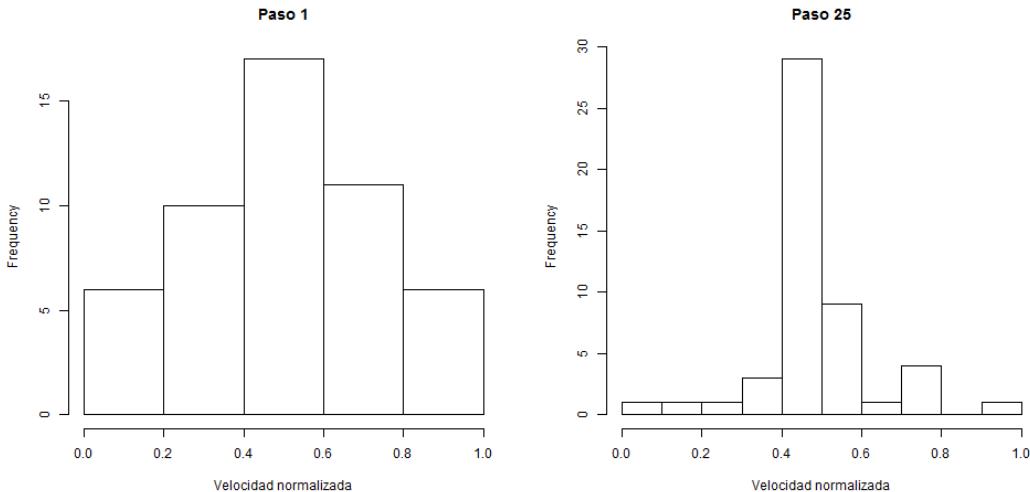


Figura 4: Velocidad de las partículas con respecto al eje X para el paso 1 y 25

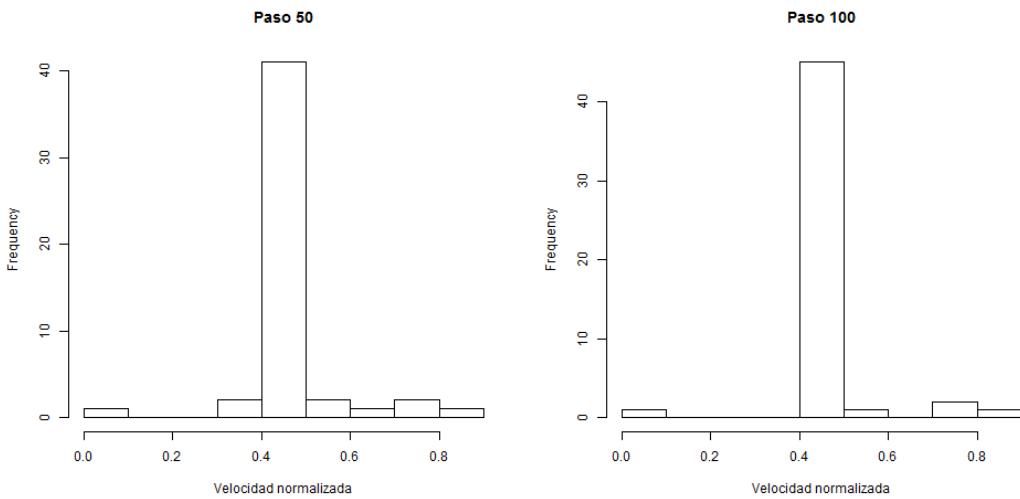


Figura 5: Velocidad de las partículas con respecto al eje X para el paso 50 y 100

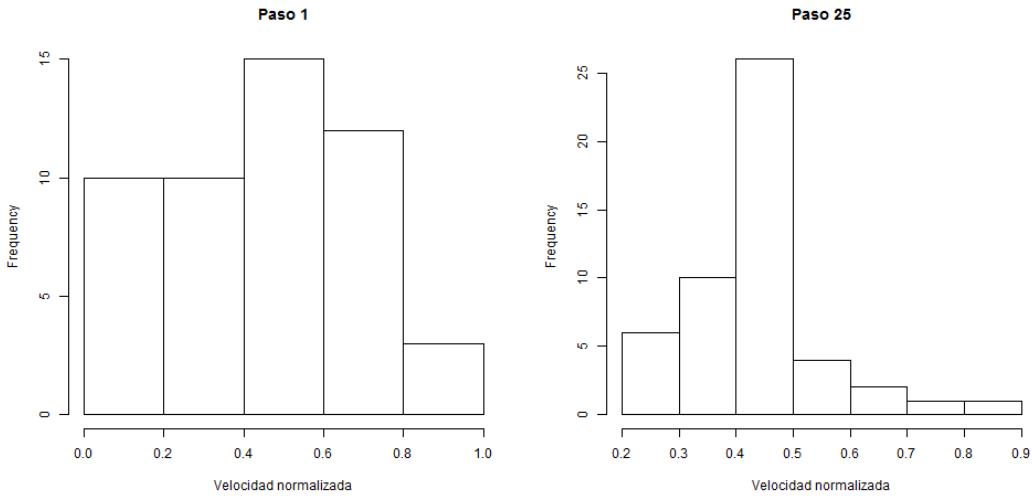


Figura 6: Velocidad de las partículas con respecto al eje Y para el paso 1 y 25

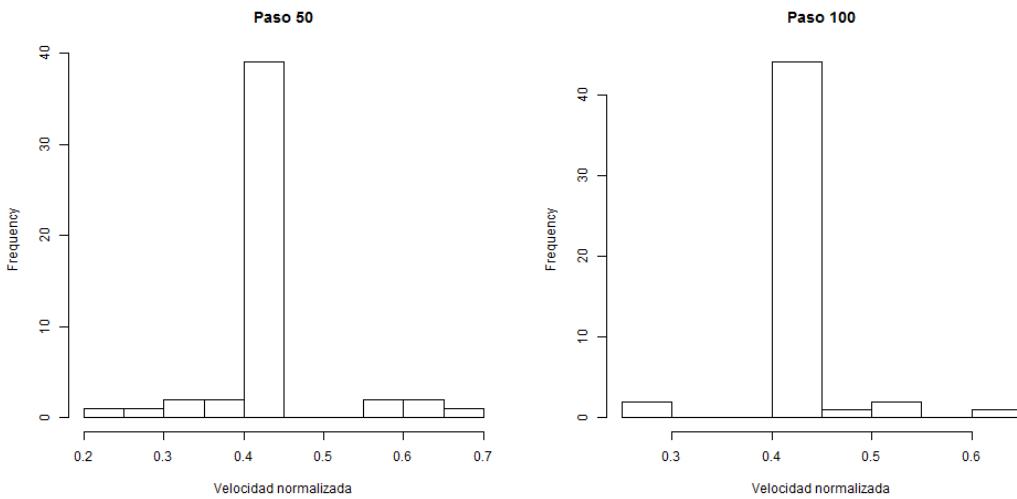


Figura 7: Velocidad de las partículas con respecto al eje Y para el paso 50 y 100

En la figura 8 y 9 se observa la relación existente entre las variables que se tienen. Se aplicó la función pairs para obtener las gráficas. Se pudo determinar una relación existente entre la velocidad de X y Y por ese motivo se puede observar este tipo de comportamiento a lo largo de este experimento. Y también se observó que con las variables de carga y masa no presentan una relación, por eso la gráfica con el paso del tiempo no tuvo cambio alguno.

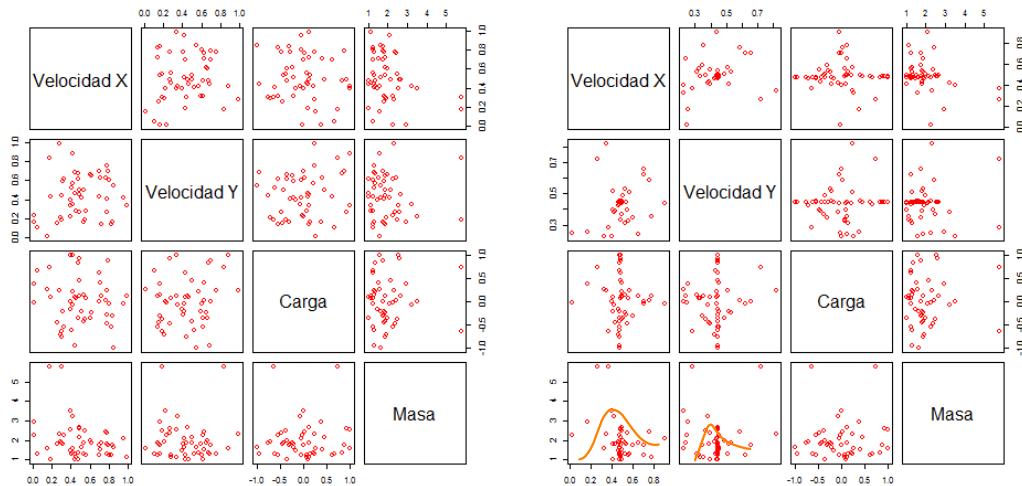


Figura 8: Relación entre las diferentes variables, paso 1 y 25

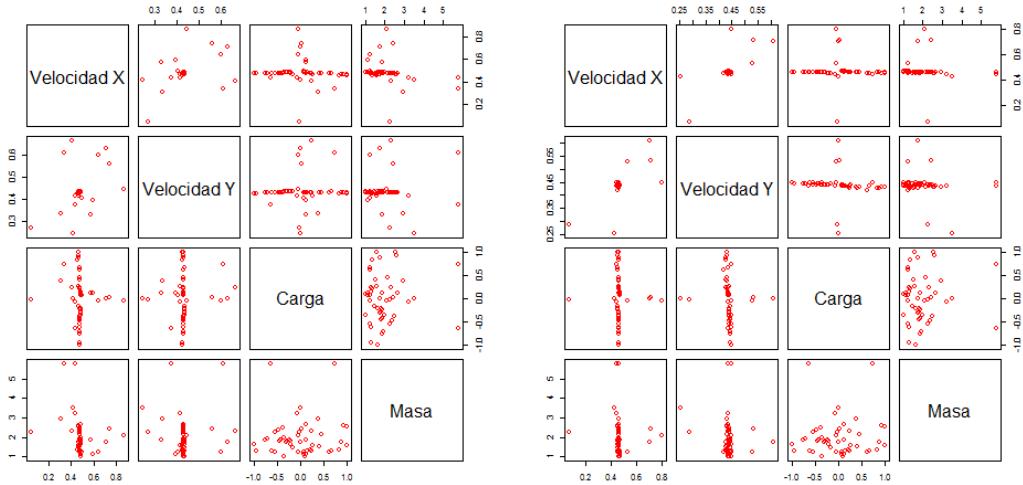


Figura 9: Relación entre las diferentes variables, paso 50 y 100

4. Conclusiones

Sin importar la masa de las partículas ~~esta~~ llegan a juntarse y mantenerse estables.

Se pudo apreciar que no hubo una relación entre la masa y la carga de las partículas y que no afectó de manera negativa a las demás variables.

Referencias

- [1] Yessica Reyna Fernández. Práctica 9: Interacciones entre partículas. *Universidad Autónoma de Nuevo León*, 2018.
- [2] Satu Elisa Schaeffer. Práctica 9: Interacciones entre partículas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.

Interacciones entre partículas

Abraham Azael Morales Juárez 1422745

11 de junio de 2019

1. Introducción

En esta práctica se realizará un modelo simplificado para los fenómenos de atracción y resulsión, uno de estos fenómenos se le conoce fuerza electrostática [2]. En donde aplica la ley de Coulomb, partículas con cargas iguales se repelen y con cargas opuestas se atraen. Además de que su fuerza es proporcional a la diferencia de la magnitud de las cargas e inversamente proporcional a la distancia euclíadiana entre las partículas. Se crearán partículas que tendrán estas características de atracción o repulsión [1].

2. Objetivos

- Agregar a cada partícula masa y generar que esta masa afecte las fuerzas de atracción.
- Estudiar la distribución de las velocidades de las partículas y verificar que esté presente una relación entre los factores: velocidad, carga y masa.

3. Resultados

En la figura 1 se observan los resultados de las 50 partículas generadas donde el color representa la magnitud de la carga.

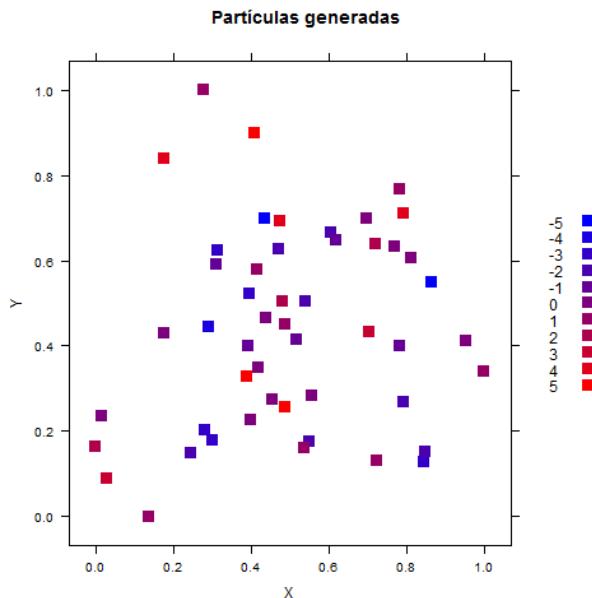


Figura 1: Partículas generadas.

En el código se agregaron los parámetros con los cuales se trabajo, entre ellos la variable nueva que es la masa.

```

1 n <- 50
2 p <- data.frame(x = rnorm(n), y=rnorm(n), c=rnorm(n), m=rnorm(n))
3 mmax <- max(p$m)
4 mmin <- min(p$m)
5 p$m <- ((p$m - mmin)*(p$m - mmin) / (mmax - mmin))+1
6 xmax <- max(p$x)
7 xmin <- min(p$x)
8 p$x <- (p$x - xmin) / (xmax - xmin) # ahora son de 0 a 1
9 ymax <- max(p$y)

```

Además se agregan las líneas del código donde éstas muestran el efecto que tienen sobre las fuerzas de atracción entre las partículas así como la velocidad de éstas.

```

1 fuerzas <- data.frame()
2 for (iter in 1:tmax) {
3   aux <- c()
4   f <- foreach(i = 1:n, .combine=c) %dopar% fuerza(i)
5   delta <- 0.02 / max(abs(f)) # que nadie desplace una paso muy largo
6   mf <- delta*f
7   p$x <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$x + delta * f[c(TRUE, FALSE)][i], 1), 0)
8   p$y <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$y + delta * f[c(FALSE, TRUE)][i], 1), 0)
9   for(t in seq(1, (length(mf)-1), by = 2)){
10     aux <- c(aux, (sum((mf[t])^2, (mf[t+1])^2))^(1/2))
11   }
12   fuerzas <- rbind(fuerzas, aux)
13   tl <- paste(iter, "", sep="")
14   while (nchar(tl) < digitos) {
15     tl <- paste("0", tl, sep="")
16   }

```

En la figura 2 – 3 se puede observar el comportamiento de estas partículas a lo largo del experimento, tomando como ejemplos distintas etapas para poder observar mejor el avance de estas.

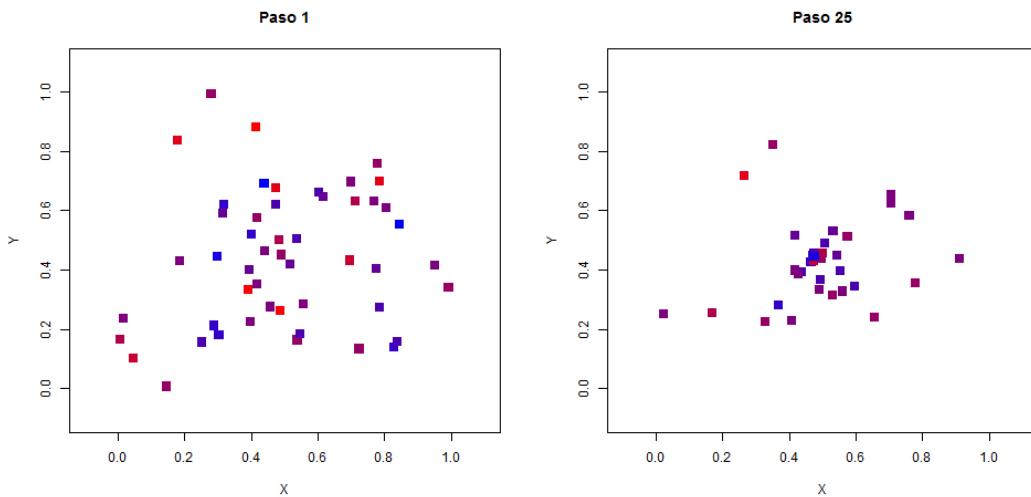


Figura 2: Simulación de atracción de las partículas en los pasos 1 y 25.

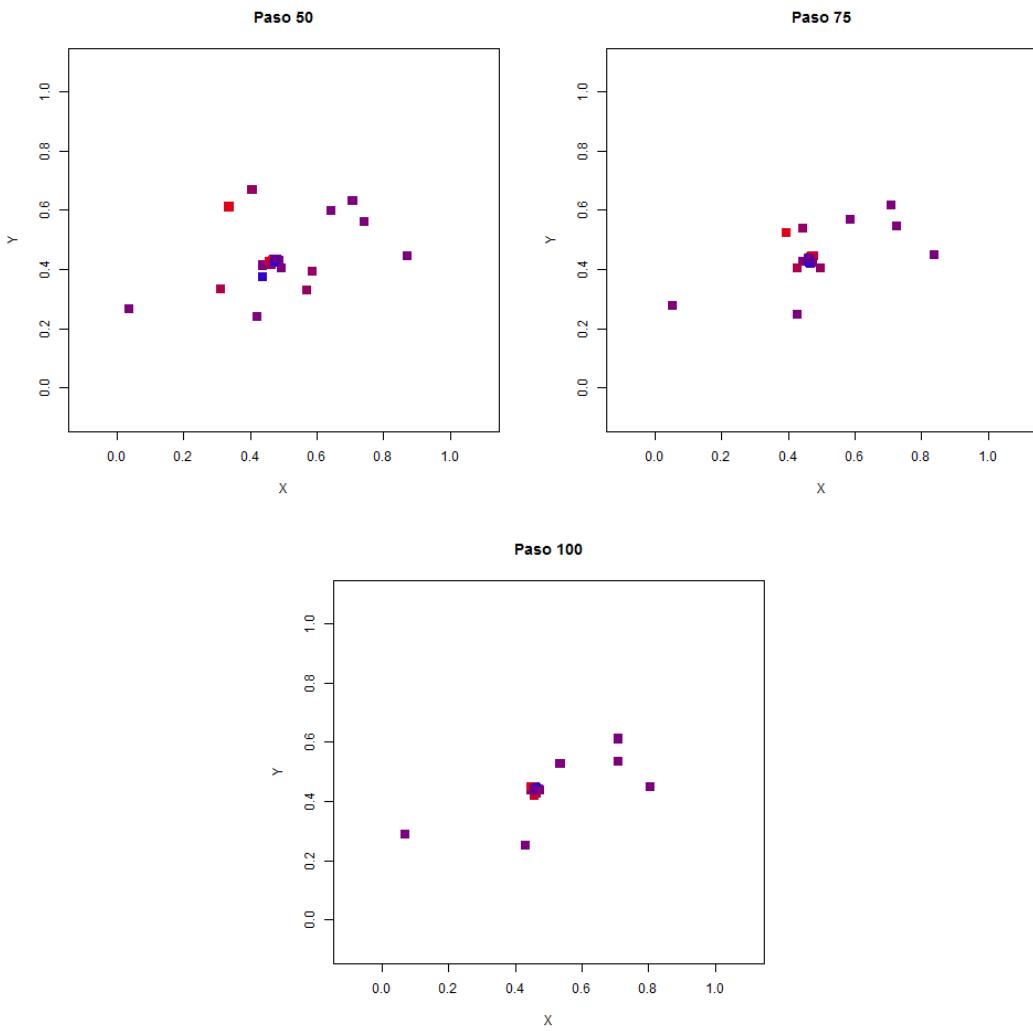


Figura 3: Simulación de atracción de las partículas en los pasos 50, 75 y 100.

De la figura 4 – 7 se muestran las velocidades de distintas secciones del experimento, y se observa que van obteniendo la misma velocidad conforme avanzan los pasos.

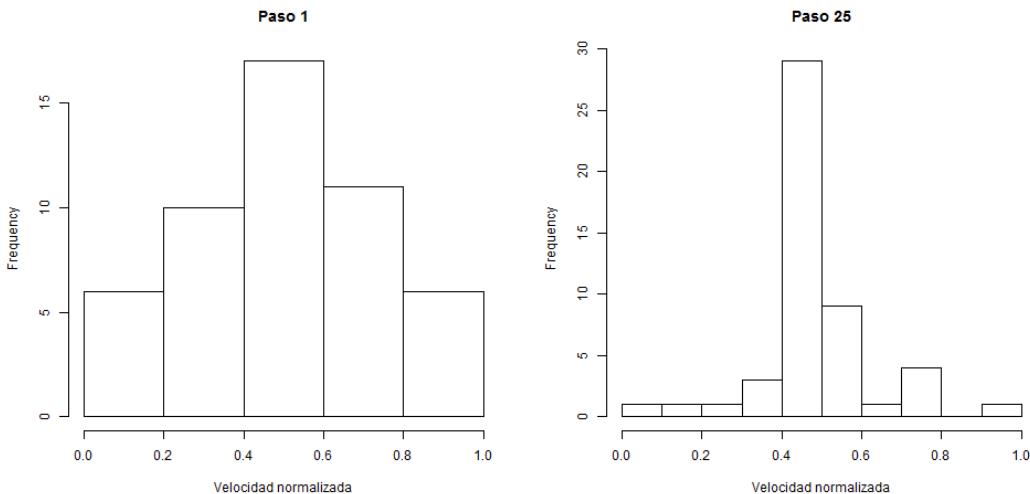


Figura 4: Velocidad de las partículas con respecto al eje X para el paso 1 y 25

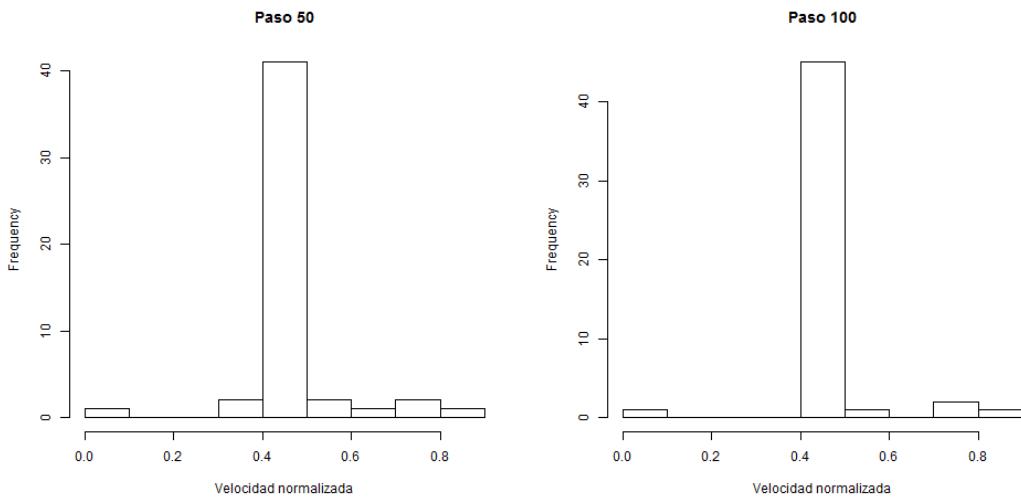


Figura 5: Velocidad de las partículas con respecto al eje X para el paso 50 y 100

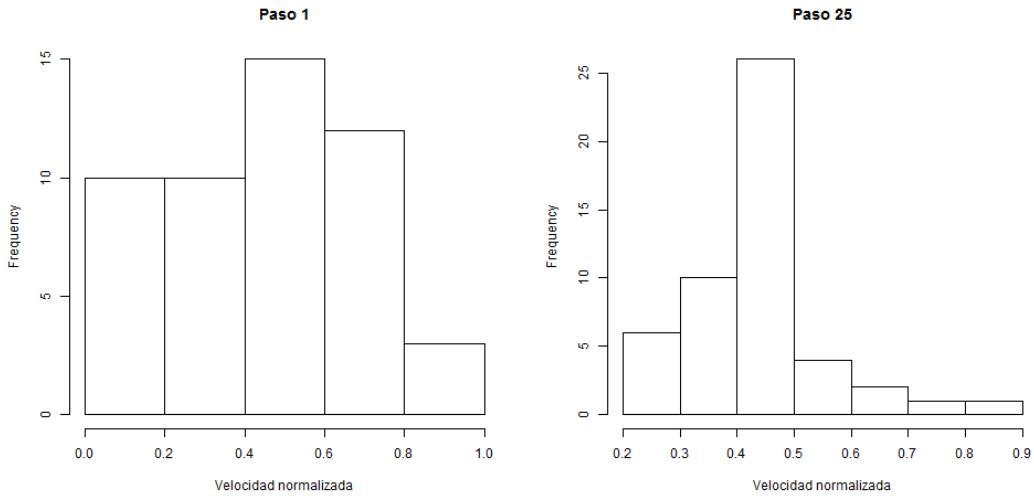


Figura 6: Velocidad de las partículas con respecto al eje Y para el paso 1 y 25

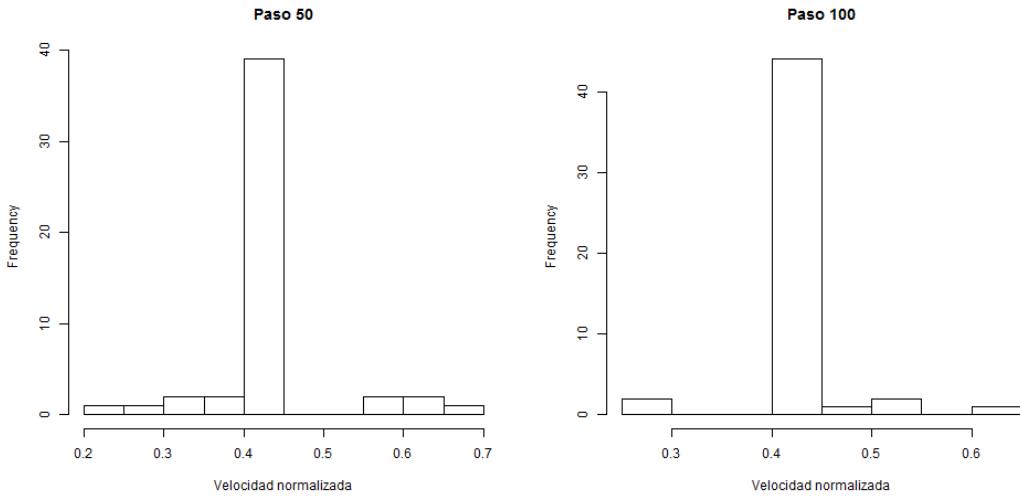


Figura 7: Velocidad de las partículas con respecto al eje Y para el paso 50 y 100

En las figuras 8 – 9 se observa la relación existente entre las variables que se tienen. Se aplicó la función pairs para obtener las gráficas. Se pudo determinar una relación existente entre la velocidad de X y Y por ese motivo se puede observar este tipo de comportamiento a lo largo de este experimento. Y también se observó que con las variables de carga y masa no presentan una relación, por eso la gráfica con el paso del tiempo no tuvo cambio alguno.

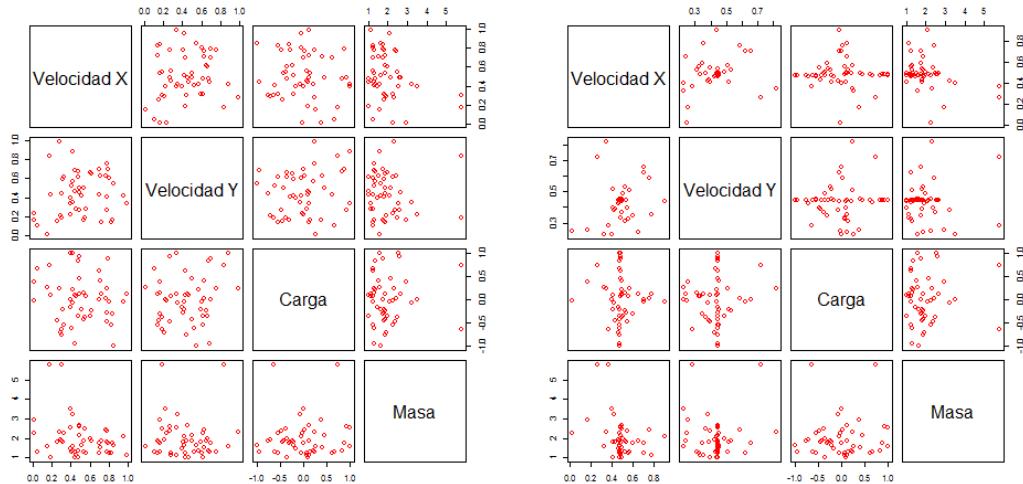


Figura 8: Relación entre las diferentes variables, paso 1 y 25

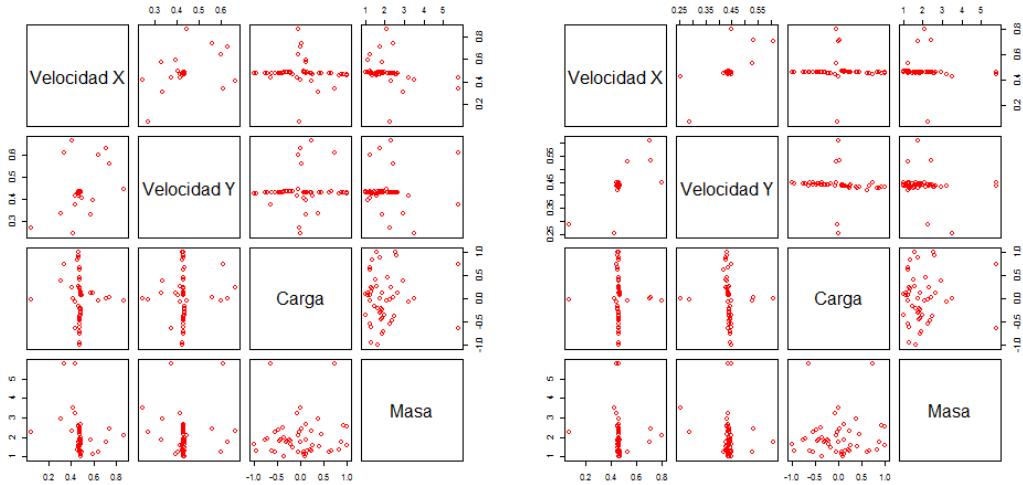


Figura 9: Relación entre las diferentes variables, paso 50 y 100

4. Conclusiones

Se pudo apreciar que no hubo una relación entre la masa y la carga de las partículas y que no afecto de manera negativa a las demás variables.

Referencias

- [1] Yessica Reyna Fernández. Práctica 9: Interacciones entre partículas, 2018. URL <https://sourceforge.net/p/simulacion-de-sistemas/activity/?page=0&limit=100#5bf4746bee24ca3d9a5fd61e>.
- [2] Satu Elisa Schaeffer. Práctica 9: Interacciones entre partículas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.

Algoritmo genético



Abraham Azael Morales Juárez 1422745

9 de abril de 2019

1. Introducción

En esta práctica se trata de la implementación de un algoritmo genético y compararlo con un algoritmo exacto que en este caso es el de mochila (en inglés "knapsack"), que es un problema clásico de optimización, el cual consiste en seleccionar objetos de un grupo en específico pero que no exceda la capacidad de soporte en peso que tenemos y que además el valor de los objetos sea el máximo posible [1].

2. Objetivos

Paralelizar el algoritmo genético con el algoritmo exacto para poder observar las diferencias entre los tiempos de estas y observar en que casos el tamaño de la instancia del algoritmo genético es mejor que el algoritmo exacto en términos de valor total obtenido por segundo de ejecución [1].

Generar condiciones con estas 3 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.

3. Resultados

Se tomaron en cuenta varios parámetros para realizar esta práctica los cuales se observan en el siguiente fragmento del código donde además se le agrego un código para que porporece el tiempo en que tarda en realizar los análisis.

```
1 for(init in c(20, 50, 80)){
2   print(init)
3   for(replica in 1:5){
4     n <- 50
5     pesos <- generador.pesos(n, 2, 30)
6     valores <- generador.valores(pesos, 10, 200)
7     capacidad <- round(sum(pesos) * 0.65)
8     optimo <- knapsack(capacidad, pesos, valores)
9     pm <- 0.05
10    rep <- 50
11    tmax <- 50
12    startpar=Sys.time()
13    p <- as.data.frame(t(parSapply(cluster, 1:init, function(i){return(round(runif(n))))}))
14    clusterExport(cluster, "p")
15    mutan=sample(1:tam,round(pm*tam)) #Elegir cuales van a mutar con pm
16    p <- rbind(p,(t(parSapply(cluster,mutan,function(i){return(mutacion(unlist(p[i,]), n)))))))
17
18    clusterExport(cluster, "tam")
19    clusterExport(cluster, "p")
20    padres <- parSapply(cluster, 1:rep, function(x){return(sample(1:tam, 2, replace=FALSE))})
21    clusterExport(cluster, "padres")
22    hijos <- parSapply(cluster, 1:rep, function(i){return(as.matrix(unlist(reproduccion(p[ padres[1, i
23      ],], p[ padres[2, i],], n),ncol=n)))})
24    p = rbind(p,hijos)
```

```

1 startSec=Sys.time()
2   p <- poblacion.inicial(n, init)
3   tam <- dim(p)[1]
4   pm <- 0.05
5   rep <- 50
6   tmax <- 50
7   mejores <- double()
8   for (iter in 1:tmax) {
9     p$obj <- NULL
10    p$fact <- NULL
11    for (i in 1:tam) { # cada objeto puede mutarse con probabilidad pm
12      if (runif(1) < pm) {
13        p <- rbind(p, mutacion(p[i,], n))
14      }
15    }
16    for (i in 1:rep) { # una cantidad fija de reproducciones
17      padres <- sample(1:tam, 2, replace=FALSE)
18      hijos <- reproduccion(p[padres[1],], p[padres[2],], n)
19      p <- rbind(p, hijos[1:n]) # primer hijo
20      p <- rbind(p, hijos[(n+1):(2*n)]) # segundo hijo
21    }

```

Al momento de realizar la paralelización se varió la población inicial y los resultados se pueden observar en la figura 1, en donde se puede observar que el paralelismo realmente disminuyó los tiempos los cuales se realiza el análisis.

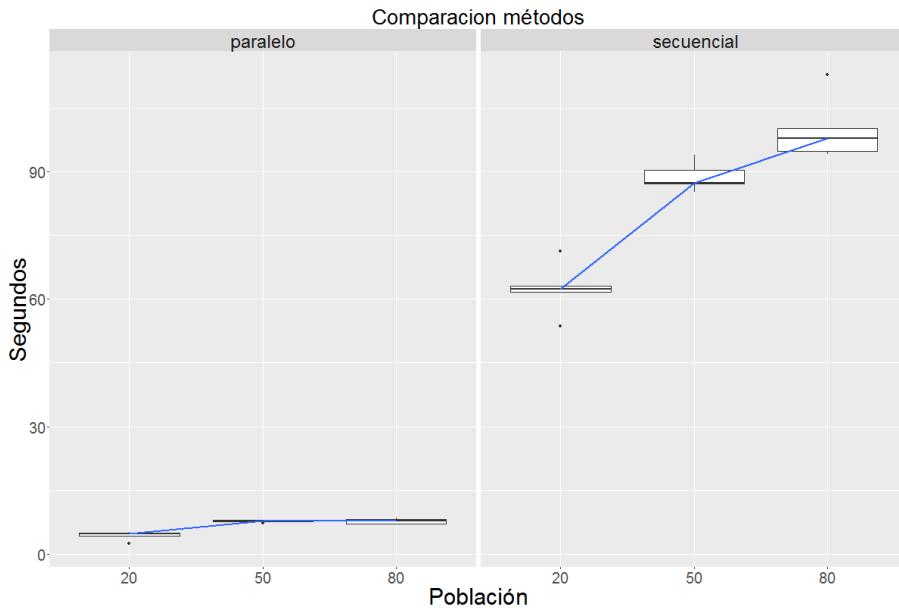


Figura 1: Comparación de los tiempos entre los métodos secuencial y paralelo. Línea azul une las medias de las poblaciones.

4. Conclusiones

Se logró una optimización en los tiempos del análisis de los logaritmos, lo cuál se demuestra la importancia que tiene este método para el análisis de datos.

Referencias

- [1] Satu Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

Algoritmo genético

Abraham Azael Morales Juárez 1422745

13 de junio de 2019

1. Introducción

En esta práctica se trata de la implementación de un algoritmo genético y compararlo con un algoritmo exacto que en este caso es el de mochila (en inglés "knapsack"), que es un problema clásico de optimización, el cual consiste en seleccionar objetos de un grupo en específico pero que no exceda la capacidad de soporte en peso que tenemos y que además el valor de los objetos sea el máximo posible [2].

2. Objetivos

Paralelizar el algoritmo genético con el algoritmo exacto para poder observar las diferencias entre los tiempos de estas y observar en que casos el tamaño de la instancia del algoritmo genético es mejor que el algoritmo exacto en términos de valor total obtenido por segundo de ejecución [2].

Generar condiciones con estas 3 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.

3. Resultados

Se tomaron en cuenta varios parámetros para realizar esta práctica los cuales se observan en el siguiente fragmento del código, el cual se basó del trabajo realizado por otro compañero [1], donde además se le agrega un código para que proporcione el tiempo en que tarda en realizar los análisis.

```
1 for(init in c(20, 50, 80)){  
2   print(init)  
3   for(replica in 1:5){  
4     n <- 50  
5     pesos <- generador.pesos(n, 2, 30)  
6     valores <- generador.valores(pesos, 10, 200)  
7     capacidad <- round(sum(pesos) * 0.65)  
8     optimo <- knapsack(capacidad, pesos, valores)  
9     pm <- 0.05  
10    rep <- 50  
11    tmax <- 50  
12    startpar=Sys.time()  
13    p <- as.data.frame(t(parSapply(cluster,1:init ,function(i){return(round(runif(n))))}))  
14      clusterExport(cluster ,”p”)  
15      mutan=sample(1:tam,round(pm*tam)) #Elegir cuales van a mutar con pm  
16      p <- rbind(p,(t(parSapply(cluster ,mutan, function(i){return(mutacion(unlist(p[i,]), n)))))))  
17      clusterExport(cluster ,”tam”)  
18      clusterExport(cluster ,”p”)  
19      padres <- parSapply(cluster ,1:rep ,function(x){return(sample(1:tam, 2, replace=FALSE) )})
```

```

21 clusterExport(cluster , "padres")
22 hijos <- parSapply(cluster ,1:rep , function(i){return(as.matrix(unlist(reproduccion(p[ padres[1 , i
23 ] ,] , p[ padres[2 , i ] ,] , n)) , ncol=n)))})
      p = rbind(p, hijos)

```

```

1 startSec=Sys.time()
2   p <- poblacion.inicial(n, init )
3   tam <- dim(p)[1]
4   pm <- 0.05
5   rep <- 50
6   tmax <- 50
7   mejores <- double()
8   for (iter in 1:tmax) {
9     p$obj <- NULL
10    p$fact <- NULL
11    for (i in 1:tam) { # cada objeto puede mutarse con probabilidad pm
12      if (runif(1) < pm) {
13        p <- rbind(p, mutacion(p[i ,] , n))
14      }
15    }
16    for (i in 1:rep) { # una cantidad fija de reproducciones
17      padres <- sample(1:tam, 2, replace=FALSE)
18      hijos <- reproduccion(p[ padres[1] ,] , p[ padres[2] ,] , n)
19      p <- rbind(p, hijos[1:n]) # primer hijo
20      p <- rbind(p, hijos[(n+1):(2*n)]) # segundo hijo
21    }
}

```

Al momento de realizar la paralelización se varió la población inicial y los resultados se pueden observar en la figura 1, en donde se puede observar que el paralelismo realmente disminuyó los tiempos los cuales se realiza el análisis.

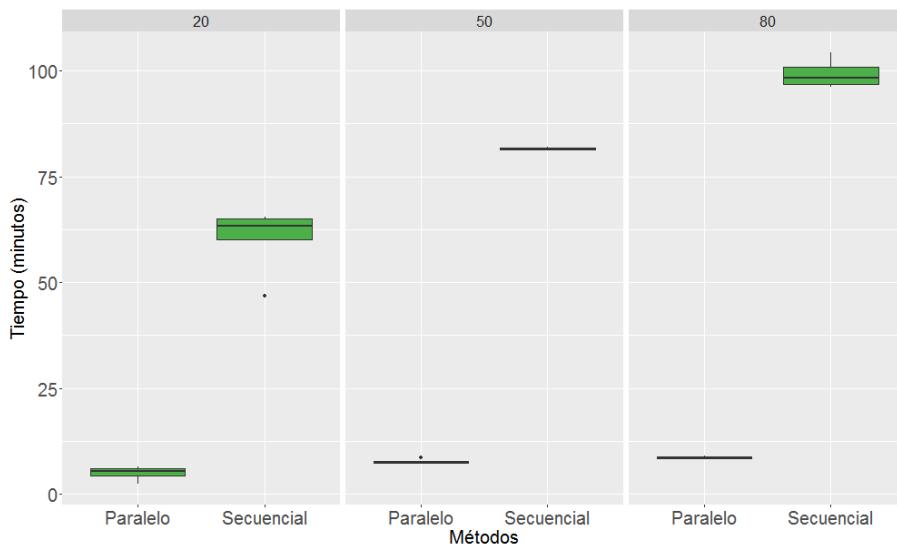


Figura 1: Comparación de los tiempos entre los métodos secuencial y paralelo.

4. Conclusiones

Se logró una optimización en los tiempos del análisis de los logaritmos, lo cuál se demuestra la importancia que tiene este método para el análisis de datos.

Referencias

- [1] Serna Mendoza Jorge Armando. Práctica 10: algoritmo genético, 2018. URL <https://sourceforge.net/p/simulacionenr/activity/?page=0&limit=100#5c06ae1fe8ba7c1fe275b514>.
- [2] Satu Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.



Frentes de Pareto

Abraham Azael Morales Juárez 1422745

30 de abril de 2019

1. Introducción

Un problema multicriterio es el cual tiene un conjunto de variables que le asignan valores de tal forma que se optimicen más de ~~2~~ funciones objetivos. Aunque uno de los problemas típicos es que una función puede mejorar mientras que otra puede empeorar, además de otro tipo de restricciones que en esta práctica no se toma en cuenta[2]. El generador de funciones objetivo en este caso será un generador de polinomios, estos a su vez tienen una variable por término y un término por variable. Aquí es donde se toma en cuenta la dominancia de Pareto que ésta es una clasificación de soluciones, es decir, si una solución domina a otra si no empeora ninguno de los objetivos y mejora a por lo menos uno[2].

2. Objetivos

Realizar una paralelización del cálculo donde convenga.

Graficar el porcentaje de soluciones de Pareto como función del número de funciones objetivo como diagramas de violín combinados con diagramas caja-bigote, verificando que las diferencias observadas sean estadísticamente significativas. Razona en escrito a que se debe.

3. Resultados

La simulación se llevó a cabo con polinomios de 5 términos, 4 variables y un grado máximo de 3, sólo se consideró 150 soluciones iniciales para buscar el porcentaje del frente de Pareto, variando de 2 hasta 10 funciones objetivos con 30 réplicas en el código en paralelo[1].

Se incluye un fragmento del código donde se observa la parte de la paralelización

```
1 vc <- 4
2 md <- 3
3 tc <- 5
4 funciones <- c("pick.one", "poli", "eval", "domin.by")
5 variables <- c("vc", "md", "tc")
6 replicas <- 30
7
8 library(parallel)
9 mc <- makeCluster(detectCores() - 1)
10 clusterExport(mc, funciones)
11 clusterExport(mc, variables)
12 resultados <- data.frame(fun = integer(), rep = integer(), SolNoDom = integer(), Porcentaje = numeric())
13 n <- 150
14 for (k in 2:10) {
15   clusterExport(mc, "k")
16   for (r in 1:replicas) {
17     obj <- parSapply(mc, 1:k, function(i) {
18       return(list(poli(md, vc, tc)))
19     })
20
21     minim <- (runif(k) > 0.5)
22     sign <- (-1)^minim
23 }
```

Los resultados de la simulación se observan en la figura 1, donde se observa una tendencia hacia el máximo porcentaje de 100 al momento de considerar una mayor cantidad de funciones objetivos.

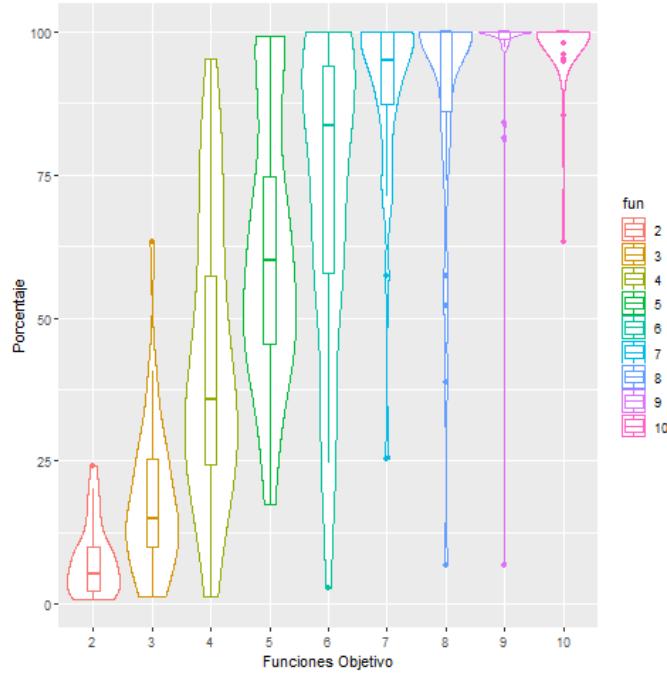


Figura 1: Resultado de la simulación, comparando la cantidad de funciones objetivo contra el porcentaje.

También se realiza una prueba de Dunn para examinar si los datos cumplen con normalidad, con un nivel de significancia de 0.05, concluyendo que los porcentajes de soluciones no dominadas son a partir de la octava función objetivo.

También se realiza una prueba de Dunn para examinar si los datos cumplen con normalidad, figura 2.

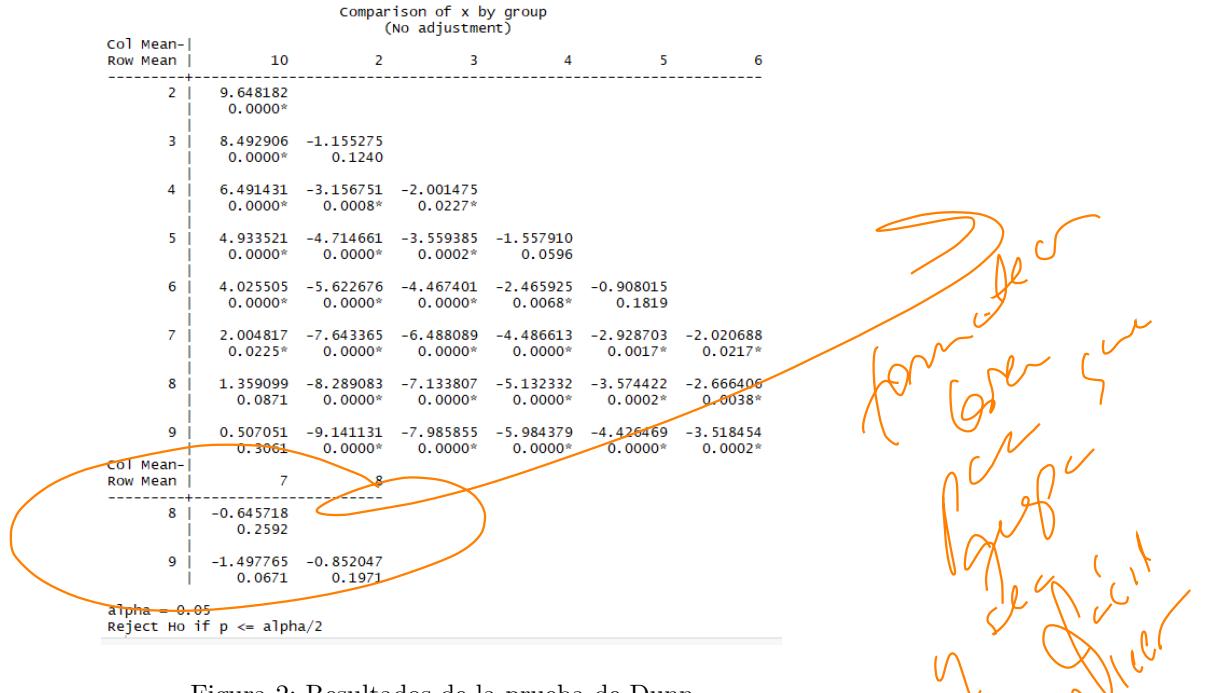


Figura 2: Resultados de la prueba de Dunn.

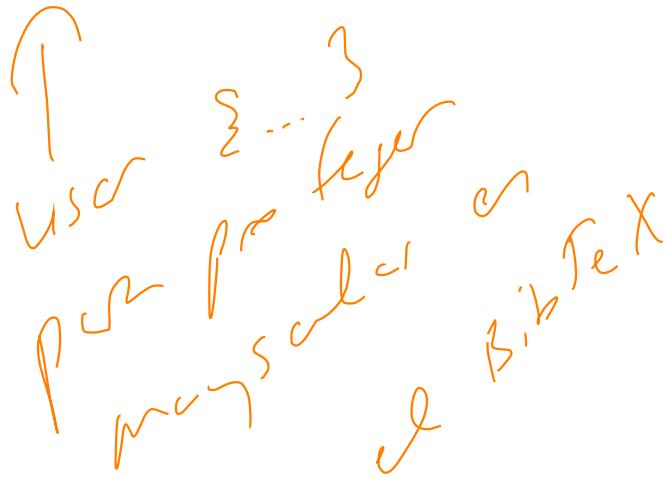
También se realiza una prueba de Dunn para examinar si los datos cumplen con normalidad, con un nivel de significancia de 0.05, concluyendo que los porcentajes de soluciones no dominadas son a partir de la octava función objetivo.

4. Conclusiones

Se concluye que considerar 8 o una cantidad mayor de funciones objetivo no existen diferencias significativas entre los porcentajes del frente de Pareto. Mientras que de 7 o menores solo no existen diferencias significativas cuando se toman consecutivas las funciones.

Referencias

- [1] Angel Moreno. Práctica 11: frentes de Pareto. 2018. URL <https://github.com/angisabel44/Simulation/tree/master/Homework11>.
- [2] Satu Elisa Schaeffer. Práctica 11: frentes de Pareto, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.



Práctica 12: red neuronal

Abraham Azael Morales Juárez 1422745

7 de mayo de 2019



1. Introducción

En esta práctica se trata de lo que se conoce como aprendizaje a máquina o una red neuronal, este es un proceso que permite a una computadora aprender a interpretar el comportamiento o naturaleza de un conjunto de datos a partir de observaciones o muestras [2]. El elemento básico de una red neuronal es un perceptrón que es un hiperplano que se coloca en la frontera de la separación de entradas verdaderas y falsas [3]. Con la dimensión del perceptrón d dada a lo largo de un vector x de entrada y representando el estado del perceptrón por w que tiene los pesos [3].

2. Objetivos

Paralelizar el código y estudiar de manera sistemática el desempeño de la red neuronal para los diez dígitos en función de las tres probabilidades asignadas (ngb) variando estos valores.

3. Resultados

El código base [3] se modificó para paralelizar utilizando como referencia el código de un compañero [1]. Además de la paralelización se midió el desempeño del código tomando en cuenta la varianza con respecto a la normal de los valores obtenidos. Los resultados se pueden observar en las figuras 1 y 2.

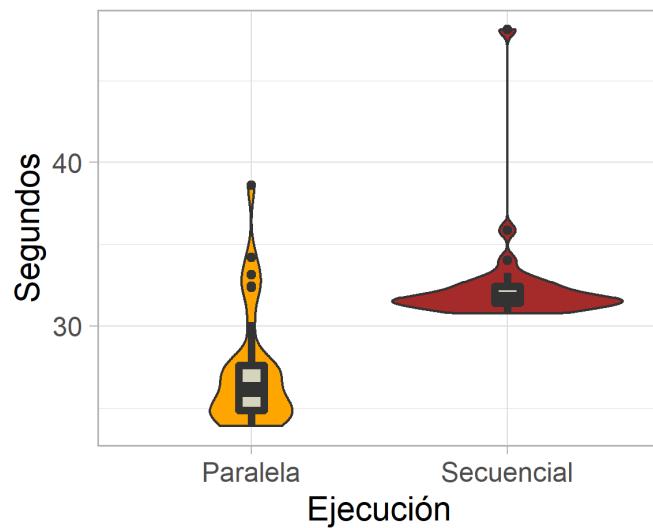


Figura 1: Se realizó la comparación de los métodos en secuencial y en paralelo, los tiempos ~~arrojaron~~ que el paralelo fue el mejor.

Se agrega un fragmento del código con el cual se obtuvo estos resultados.

```

1 suppressMessages(library(doParallel))
2 Mc <- makeCluster(detectCores() - 1)
3 registerDoParallel(Mc)
4
5 prueba1 <- microbenchmark( {
6   contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
7   rownames(contadores) <- 0:tope
8   colnames(contadores) <- c(0:tope, NA)
9   conta <- foreach(t = 1:300, .combine = "rbind", .export = c("tope", "dim", "modelos", "neuronas", "pixeles")) %dopar% newr()
10  for (i in 1:300){
11    contadores[conta[i, 1], conta[i, 2]] <- contadores[conta[i, 1], conta[i, 2]] + 1
12  }
13 }
14 times = repe, unit = "s")
15 stopCluster(Mc)
16
17 entrenamientos <- cbind("Secuencial" = entrena$time)
18 pruebas <- cbind("Secuencial" = prueba$time, "Paralela" = prueba1$time)
19 library(ggplot2)
20 pruebas <- melt(pruebas)
21 ggplot(tests, aes(x=as.factor(Var2), y=value/10000000)) + geom_violin(aes(fill=as.factor(Var2))) +
22   geom_boxplot(fill = "#d6d4bc", width=0.1, lwd = 1.5) + scale_fill_manual(values=c("orange", "brown"))
23   +
24   labs(x = "Ejecucion", y = "Segundos") + theme_light(base_size = 14) + guides(fill=FALSE)
25 ggsave("Tiempo.png")

```

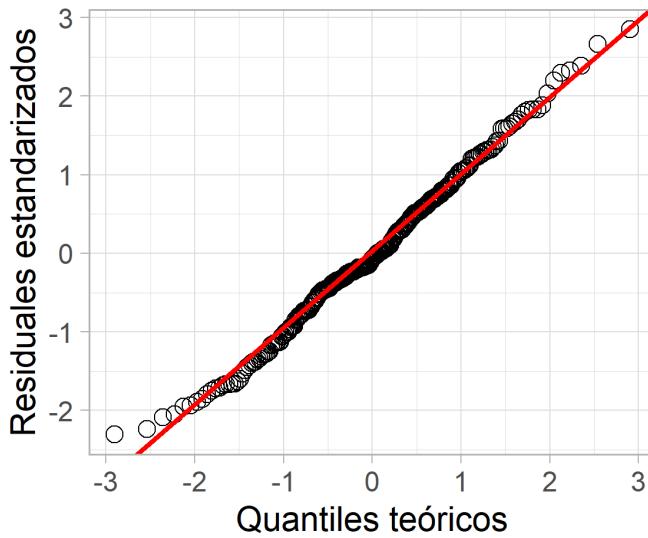


Figura 2: Normalidad de los resultados obtenidos

En la figura 2 se puede observar que el mayor porcentaje de los valores dados se encuentran muy cerca de la normalidad. Para obtener mayor información se realizó un análisis de varianza, cuyos resultados se presentan en la figura 3 se presenta una tabla, en la cual se observan que la mayor presencia del color de píxeles son el blanco y negro, del cual el píxel negro presento una mayor prevalencia.

Color pixel	Df	Sum Sq	Mean Sq	F value	Pr (>F)
Negro	2	4.78	2.39	65.53	0.0000
Gris	2	0.67	0.33	9.17	0.0001
Blanco	2	4.73	2.37	64.86	0.0000
Negro/Gris	4	0.32	0.08	2.20	0.0693
Negro/Blanco	4	0.48	0.12	3.27	0.0123
Gris/Blanco	4	0.05	0.01	0.34	0.8475
Negro/Gris/Blanco	8	0.47	0.06	1.60	0.1240
Residuales	243	8.87	0.04		

Figura 3: Resultados del análisis de varianza

También se presenta un fragmento del código del cual se uso para poder obtener la normalidad de los valores.

```
1 print(xtable(summary(av), type='latex'), file="Tabla_valores.txt")
2 ggQQ <- function(lm) {
3   # https://stackoverflow.com/questions/4357031/qqnorm-and-qqline-in-ggplot2/19990107#19990107
4   d <- data.frame(std.resid = rstandard(lm))
5   y <- quantile(d$std.resid [!is.na(d$std.resid)], c(0.25, 0.75))
6   x <- qnorm(c(0.25, 0.75))
7   slope <- diff(y)/diff(x)
8   int <- y[1L] - slope * x[1L]
9
10  p <- ggplot(data=d, aes(sample=std.resid)) +
11    stat_qq(shape=1, size=3) +           # open circles
12    labs(title=NULL,                  # plot title
13          x="Quantiles te\uf3f3ricos",    # x-axis label
14          y="Residuales estandarizados") + # y-axis label
15    geom_abline(slope = slope, intercept = int, col = "red", lwd = 1) # dashed reference line
16  p <- p + theme_light(base_size = 14)
17  return(p)
18}
19 ggQQ(lm.model)
20 ggsave("Normalidad.png")
```

4. Conclusiones

Se redujo los tiempos del procesamiento al usar el método paralelo. Los datos arrojados por los experimentos muestran que los datos son normalmente distribuidos, pero no tienen la misma proporción de distribución porque las varianzas no son iguales a 1. Además, que los píxeles con mayor presencia con los blancos y negros, del cual resalta por poco el píxel negro.

Referencias

- [1] Ricardo Rosas Macías. Práctica 12: Red neuronal. 2018. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP12>.
- [2] Eduardo Coca Reyna. Práctica 12: Red neuronal. *Universidad Autónoma de Nuevo León*, 2018. 
- [3] Satu Elisa Schaeffer. Práctica 12: Red neuronal, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

Práctica 12: red neuronal

Abraham Azael Morales Juárez 1422745

12 de junio de 2019

1. Introducción

En esta práctica se trata de lo que se conoce como aprendizaje a máquina o una red neuronal, este es un proceso que permite a una computadora aprender a interpretar el comportamiento o naturaleza de un conjunto de datos a partir de observaciones o muestras [2]. El elemento básico de una red neuronal es un perceptrón que es un hiperplano que se coloca en la frontera de la separación de entradas verdaderas y falsas. Con la dimensión del perceptrón d dada a lo largo de un vector x de entrada y representando el estado del perceptrón por w que tiene los pesos [3].

2. Objetivos

Paralelizar el código y estudiar de manera sistemática el desempeño de la red neuronal para los diez dígitos en función de las tres probabilidades asignadas (ngb) variando estos valores.

3. Resultados

El código base [3] se modificó para paralelizar utilizando como referencia el código de un compañero [1]. Además de la paralelización se midió el desempeño del código tomando en cuenta la varianza con respecto a la normal de los valores obtenidos. Los resultados se pueden observar en las figuras 1 y 2.

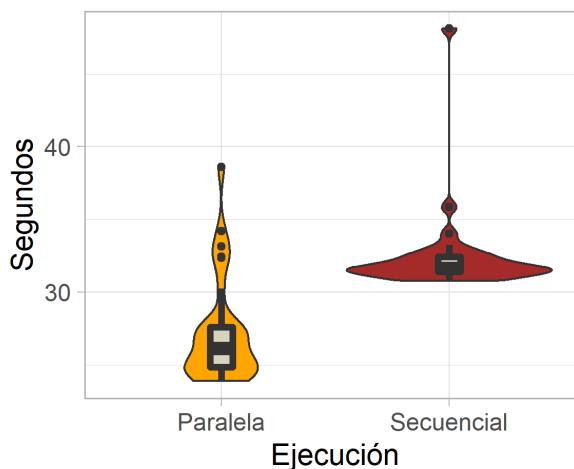


Figura 1: Se realizó la comparación de los métodos en secuencial y en paralelo, los tiempos indican que el paralelo fue el mejor.

Se agrega un fragmento del código con el cual se obtuvo estos resultados.

```

1 suppressMessages(library(doParallel))
2 Mc <- makeCluster(detectCores() - 1)
3 registerDoParallel(Mc)
4
5 prueba1 <- microbenchmark({
6   contadores <- matrix(rep(0, k*(k+1)), nrow=k, ncol=(k+1))
7   rownames(contadores) <- 0:tope
8   colnames(contadores) <- c(0:tope, NA)
9   conta <- foreach(t = 1:300, .combine = "rbind", .export = c("tope", "dim", "modelos", "neuronas", "pixeles")) %dopar% newr()
10  for (i in 1:300){
11    contadores[conta[i, 1], conta[i, 2]] <- contadores[conta[i, 1], conta[i, 2]] + 1
12  }
13}
14 times = repe, unit = "s")
15 stopCluster(Mc)
16
17 entrenamientos <- cbind("Secuencial" = entrena$time)
18 pruebas <- cbind("Secuencial" = prueba$time, "Paralela" = prueba1$time)
19 library(ggplot2)
20 pruebas <- melt(pruebas)
21 ggplot(tests, aes(x=as.factor(Var2), y=value/10000000)) + geom_violin(aes(fill=as.factor(Var2))) +
22   geom_boxplot(fill = "#d6d4bc", width=0.1, lwd = 1.5) + scale_fill_manual(values=c("orange", "brown"))
23   +
24   labs(x = "Ejecucion", y = "Segundos") + theme_light(base_size = 14) + guides(fill=FALSE)
25 ggsave("Tiempo.png")

```

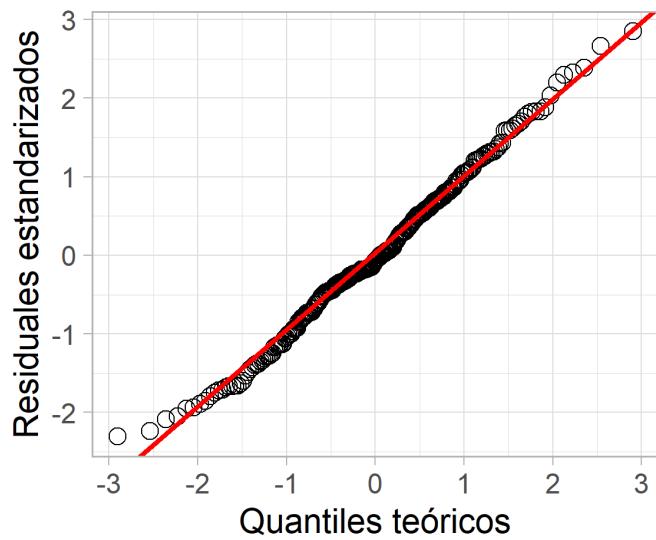


Figura 2: Normalidad de los resultados obtenidos

En la figura 2 se puede observar que el mayor porcentaje de los valores dados se encuentran muy cerca de la normalidad. Para obtener mayor información se realizó un análisis de varianza, cuyos resultados se presentan en la figura 3 se presenta una tabla, en la cual se observan que la mayor presencia del color de píxeles son el blanco y negro, del cual el píxel negro presentó una mayor prevalencia.

Figura 3: Resultados del análisis de varianza

Color píxel	Df	Sum Sq	Mean Sq	F value	Pr (>F)
Negro	2	4.78	2.39	65.53	0.0000
Gris	2	0.67	0.33	9.17	0.0001
Blanco	2	4.73	2.37	64.86	0.0000
Negro/Gris	4	0.32	0.08	2.20	0.0693
Negro/Blanco	4	0.48	0.12	3.27	0.0123
Gris/Blanco	4	0.05	0.01	0.34	0.8475
Negro/Gris/Blanco	8	0.47	0.06	1.60	0.1240
Residuales	243	8.87	0.04		

En la figura 3 se observa los valores obtenidos y en donde el valor de F es de 65,53 para el píxel negro y de 64,86 para blanco, demostrando que la diferencia no es significativa, pero en comparación con la del cuadro gris que es de 9,17 si lo es, además tienen un valor de Pr por debajo de 0,05, lo que indica que tiene una desviación estándar baja. En otras palabras, la probabilidad de presencia de pixeles negros es mayor que la de pixeles blancos y grises.

También se presenta un fragmento del código del cual se usó para poder obtener la normalidad de los valores.

```

1 print(xtable(summary(av), type='latex'), file="Tabla_valores.txt")
2 ggQQ <- function(lm) {
3   # https://stackoverflow.com/questions/4357031/qqnorm-and-qqline-in-ggplot2/19990107#19990107
4   d <- data.frame(std.resid = rstandard(lm))
5   y <- quantile(d$std.resid[!is.na(d$std.resid)], c(0.25, 0.75))
6   x <- qnorm(c(0.25, 0.75))
7   slope <- diff(y)/diff(x)
8   int <- y[1L] - slope * x[1L]
9
10  p <- ggplot(data=d, aes(sample=std.resid)) +
11    stat_qq(shape=1, size=3) +          # open circles
12    labs(title=NULL,                  # plot title
13      x="Quantiles te\u00f3ricos",     # x-axis label
14      y="Residuales estandarizados") + # y-axis label
15    geom_abline(slope = slope, intercept = int, col = "red", lwd = 1) # dashed reference line
16  p <- p + theme_light(base_size = 14)
17  return(p)
18}
19 ggQQ(lm.model)
20 ggsave("Normalidad.png")

```

4. Conclusiones

Se redujo los tiempos del procesamiento al usar el método paralelo. Los datos arrojados por los experimentos muestran que los datos son normalmente distribuidos, pero no tienen la misma proporción de distribución porque las varianzas no son iguales a 1. Hay un incremento en la probabilidad de presencia de pixeles negros.

Referencias

- [1] Ricardo Rosas Macías. Práctica 12: Red neuronal, 2018. URL <https://github.com/RicardoRosMac/Simulation/tree/master/HWP12>.
- [2] Eduardo Coca Reyna. Práctica 12: Red neuronal, 2018. URL <https://sourceforge.net/p/simulacionnano/activity/?page=1&limit=100#5bc5c1e33241d209bca61e04>.
- [3] Satu Elisa Schaeffer. Práctica 12: Red neuronal, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

Efecto de los hábitos actuales sobre la expresión o silenciamiento de genes de regulación celular.

Abraham Azael Morales Juárez

4 de junio de 2019

Resumen

Este proyecto tiene como finalidad evidenciar la relación que existe entre factores externos o medioambientales en la expresión o represión de genes que regulan las funciones celulares, lo que se conoce como epigenética. Estas modificaciones ocurren mediante varios mecanismos que son considerados epigenéticos entre estas incluyen la metilación y acetilación de las histonas, modificaciones postraduccionales de las mismas, modificación de la cromatina dependiente de ATP y ARN no codificantes, entre otros. Aunque en la actualidad se conoce poco el cómo ocurren estas modificaciones que pueden apagar o encender la expresión genética, debido a que los múltiples tipos de modificaciones epigenéticas existentes conducen a un nivel complejo y dinámico los cuales pueden afectar la expresión a corto y a largo plazo.

Palabras clave: Epigenética, Expresión genética, Represión Genética, Software R, Hábitos.

1. Introducción

1.1. Epigenética, un diálogo entre el genoma y el medioambiente.

Darwin en su obra el origen de las especies nos da a entender en pocas palabras que los organismos tienen la capacidad de interactuar y responder adecuadamente a los cambios constantes del ambiente que condicionada las posibilidades de un organismo de transmitir sus genes más que las propiedades intrínsecas de los genes. El doctor Conrad Waddington fue el que inicialmente designó esta definición como, la capacidad del medioambiente de moldear el fenotipo como epigenética [1]. Actualmente este término no tiene una definición definitiva pero lo que se considera una definición amplia y bien aplicada es la siguiente son mecanismos modificadores de cromosomas que cambian la plasticidad fenotípica de una célula u organismo [1].

En la actualidad existen cada vez una mayor cantidad de trabajos que evidencian cómo modificaciones en los patrones epigenéticos durante los primeros años de vida condicionan el riesgo de desarrollar enfermedades crónicas no heredables. En las células eucariontes el ADN se encuentra en el núcleo aislado, condensado y en ocasiones plegado para permitir la expresión de genes. La información genética está confinada en un complejo proteico denominado nucleosoma que está constituida por 2 copias de 4 proteínas llamadas histonas [2], figura 1.

La expresión genética puede ser regulada a mediano y largo plazo modificando los sitios de unión de factores de transcripción (FT) y de la polimerasa de ARN (Pol II) sin mutar la secuencia de ADN, así como en el patrón de unión que tienen las proteínas de las histonas (figuras verdes y rojas). Las proteínas verdes representan histonas metiladas, acetiladas en regiones que permiten el correcto acceso a la polimerasa para la expresión de genes. En cambio, las proteínas rojas corresponden a histonas, metiladas, acetiladas de manera incorrecta y con otras modificaciones que evitan la accesibilidad a los factores de transcripción y a la polimerasa.

Las modificaciones de este ADN pueden ocurrir a nivel de las histonas generando nuevos sitios de replicación para la maquinaria molecular presente, lo cual conducen a un nivel complejo de interacción entre factores externos y proteínas de estructura. Las modificaciones que se consideran epigenéticas incluyen metilación del ADN, modificaciones postraduccionales de histonas, modificación de cromatinas dependiente de ATP y ARN no codificantes.

En general, en el ADN la metilación en las regiones promotoras de genes se asocia con el silenciamiento de estos, mientras que una mayor metilación en la región codificante de los genes estaría asociada con la expresión activa. Además, varios estudios que examinan la relación sitio- específica de metilación del ADN, por ejemplo, se han centrado en una serie de genes candidatos implicados en la obesidad, control del apetito, metabolismo, señalización de insulina, inmunidad, crecimiento, regulación, del ciclo

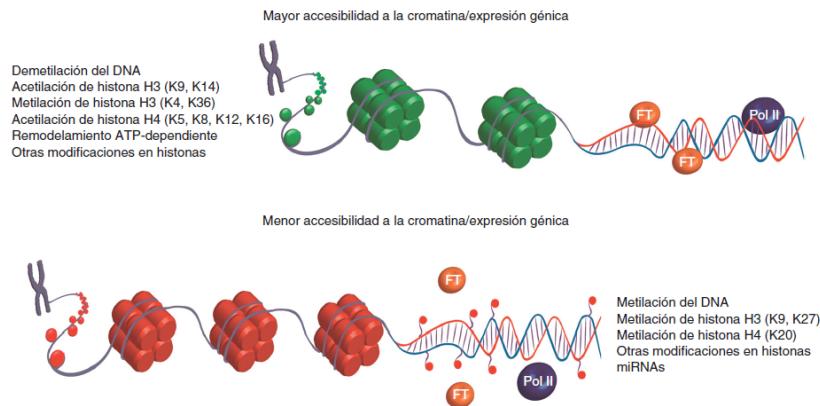


Figura 1: Mecanismos epigenéticos y sus efectos sobre la expresión genética.

circadiano y genes impresos, todos relacionados con marcadores genéticos de obesidad [2], figura 2.

Estos y muchos estudios más se han realizado para determinar la importancia de los factores epigenéticos. En conjunto, estos estudios proporcionan evidencia de que la obesidad está asociada con alteraciones de la regulación epigenética de un número importante de genes para el metabolismo.



Figura 2: Hipermethylación del ADN y su asociación con la obesidad. Además, como se relaciona con la regulación de expresión de otros genes.

La atención de la comunidad científica se ha centrado en el potencial de estas modificaciones epigenéticas en la patogenia del desarrollo de la obesidad. Aunque pocos trabajos se han realizado en humanos, la evidencia que nos aportan las investigaciones realizadas en animales han mostrado tener un impacto importante debido a que apoya el rol de la epigenética en la programación del desarrollo de enfermedades en etapas adultas [2].

Los cambios epigenéticos a diferencia de las enfermedades genéticas que se encuentran ligadas a secuencias de ADN específicas, estos son potencialmente reversibles y se refieren a modificaciones de las histonas y en ADN mismo, y sin cambiar la secuencia de este.

En la actualidad la obesidad es un factor de riesgo muy preponderante que causa muchas enfermedades como, diabetes tipo 2, hipertensión, enfermedades coronarias y cardiovasculares. Además otros estudios han determinado que

genes son candidatos y que se encuentren implicados en la obesidad, control apetito o metabolismo, señalización insulina, inmunidad, crecimiento y ciclo circadiano, todos relacionados con marcadores de obesidad, ver figura 2 donde se observan los genes están involucrados.

Todos estos estudios se han identificado con una baja metilación de factores de necrosis tumoral alfa (TNFa) en PBMC. A su vez se ha encontrado un aumento de metilación de leucocitos, en el gen receptor de aril hidrocarburos nucleares translocadoras y el coactivador de PPAR alfa tipo 1 en el músculo en obesos en comparación con las personas delgadas. Cabe destacar que en una subpoblación de la cohorte Early Bird Diabetes Study, el grado de metilación del promotor para el coactivador de PPAR alfa tipo 1 durante la infancia ha demostrado ser un predictor del nivel de adiposidad durante la pubertad [4].

2. Objetivos

Generar un código en R que permita realizar el análisis de los efectos de hábitos sobre la expresión genética, que a su vez ocasiona trastornos en la salud, el código contendrá las siguientes funciones:

Represente un factor que genere un valor que perjudique o fortalezca la calidad de vida del individuo y/o población.

Establecer de manera numérica el valor de vida.

Establecer un comportamiento según estadísticas reales y modificadas.

Proporcionar resultados interpretables que se puedan observar por períodos de tiempo determinados.

Ánalisis de comparación de eficiencia del código.

Se definen parámetros de iteraciones, tales como, rangos de selección de funciones, número de replicas, duración, muestras y valor de influencia sobre el individuo.

3. Metodología

El código generado se puede encontrar en el repositorio [6] el cual nos permite realizar un análisis de como los hábitos afectan la salud del individuo interpretando que los genes se ven afectados por esta acción. Además, observando como la calidad de vida disminuye debido a los hábitos que ha desarrollado. También, se le proporcionó capacidad para recuperar con la misma intensidad que con la que se ve afectado.

En este caso en específico se consideraron solo 4 factores, mala alimentación, depresión, falta de descanso y sedentarismo, a cada factor se le otorgó un porcentaje de importancia para la vida del individuo, y se produce el comportamiento de la población o persona para así observar como se desarrolla el proceso. En el desarrollo de este proyecto en no se tomaron en cuenta otros factores.

Las funciones creadas permitieron cuantificar las desiciones tomadas por los individuos o de la misma forma cuantificar las situaciones en las cuales un individuo es afectado por su entorno. En base a estos datos se realizaron los cálculos necesarios para poder definir cuantos puntos del total que representa cada gen son sumados o restados del porcentaje de vida.

4. Resultados

La creación del código fue en base a lo reportado por Schaeffer [7] y para realizar las funciones e iteraciones se basó del mismo código.

```
1 muestra <- 50
2 replicas <- 30
3 duracion <- 24
4 vida <- 100
5 engordar <- (0.35)*vida
6 decaido <- (0.15)*vida
7 insomnio <- (0.30)*vida
8 flojo <- (0.20)*vida
9
10 puntos <- replicas*duracion
11 pgor <- engordar/puntos
12 pdep <- decaido/puntos
13 pin <- decaido/puntos
14 psen <- flojo/puntos
15
16 comer <- function(n){
17   fat <- runif(1, desde, hasta)
18   if (fat <= 0.29){
19     return(TRUE)
20   }
21   if (fat > 0.29){
22     return(FALSE)
23   }
24   return(TRUE)
25 }
26
27 depre <- function(n){
28   dep <- runif(1, desdel, hasta1)
29   if (dep <= 0.35){
30     return(TRUE)
31 }
```

```
32   if (dep > 0.35){
33     return(FALSE)
34   }
35   return(TRUE)
36 }
```

```
1 suppressMessages(library(doParallel))
2 registerDoParallel(makeCluster
3 (detectCores() - 1))
4
5 buffet <- numeric()
6 depresion <- numeric()
7 dorm <- numeric()
8 cdntario <- numeric()
9
10 for (gente in 1:muestra) {
11   for (dias in 1:replicas) {
12     for (meses in 1:duracion) {
13       buffet <- c(buffet, foreach(n =
14         muestra, .combine=c) %dopar%
15         comer(n))
16       depresion <- c(depresion, foreach(n =
17         muestra, .combine=c) %dopar%
18         depre(n))
19       dorm <- c(dorm, foreach(n = muestra,
20         .combine=c) %dopar% no.dormir(n))
21       cdntario <- c(cdntario, foreach(n =
22         muestra, .combine=c) %dopar%
23         seden(n)))
24     }
25   stopImplicitCluster()
26 }
```

Al momento de realizar las iteraciones se consideran los siguientes parámetros: la cantidad de muestra son la cantidad de individuos, las réplicas correspondían a los días que son treinta, y la duración representan los meses que en este caso son veinticuatro, es decir, se obtuvo casi 2 años de información, figura 3.

Además, como cada factor tiene su propio valor de influencia sobre expresión de genes y por ende sobre la salud, los puntos que se obtuvieron a favor y en contra se restan, los negativos se restan de los positivos, la cantidad resultante no es interpretable directamente sino que se tiene que multiplicar por el valor otorgado a cada gen para tener el valor real de influencia sobre la salud del individuo o la población, cada gen tiene su valor que se puede apreciar en el primer código de la sección (ejemplo: pgor, es el valor otorgado al gen que se encarga de regular el apetito y saciedad).

Además, de colocar una muestra de cincuenta individuos al que corresponden los parámetros antes mencionados, se realizó otra iteración con un tamaño de muestra de solo un individuo con una duración de treinta y seis meses, figura 4.

Los parámetros con los cuales los genes iban a ser influenciados por el comportamiento de los individuos, no tomándose como referencia alguna estadística real de comportamiento, sino que son considerados dando cincuenta porciento de probabilidad para perjudicar o fortalecer.

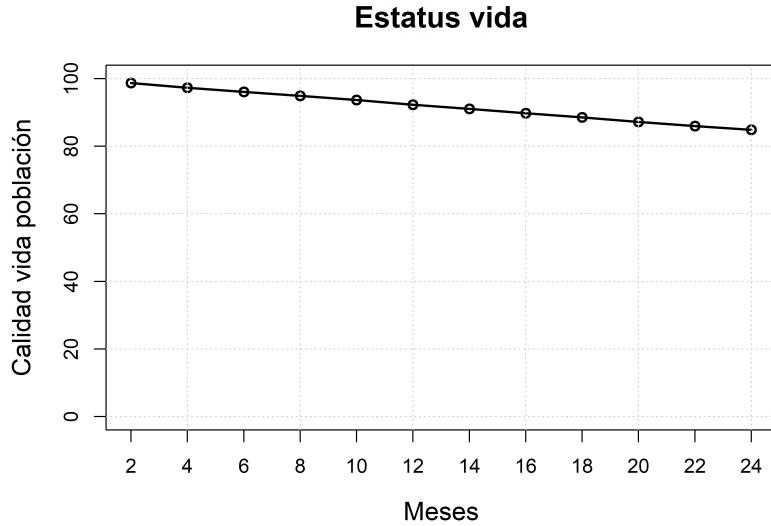


Figura 3: Resultados de simulación de una población con hábitos saludables y no saludables repartidos en misma medida, con una capacidad de regeneración. Tendencia de vida después de 2 años, con mediciones cada 2 meses.

```

1 desde <- 0 #mala alimentacion
2 hasta <- 0.45
3 desde1 <- 0.1 #depresion
4 hasta1 <- 0.6
5 desde2 <- 0.1 #falta de dormir
6 hasta2 <- 0.5
7 desde3 <- 0.4 #sedentarismo
8 hasta3 <- 0.8
9 no.dormir <- function(n){
10   desv <- runif(1, desde2, hasta2)
11   if (desv <= 0.3){
12     return(TRUE)
13   }
14   if (desv > 0.3){
15     return(FALSE)
16   }
17   return(TRUE)
18 }
19 seden <- function(n){
20   sed <- runif(1, desde3, hasta3)
21   if (sed <= 0.6){
22     return(TRUE)
23   }
24   if (sed > 0.6){
25     return(FALSE)
26   }
27   return(TRUE)
28 }
```

Se tiene que aclarar que solo se toman en cuenta estos factores, cuando en la realidad la expresión de los genes, del genoma y del epigenoma son complejos en todos los niveles [2].

Los resultados no son alentadores, a pesar de haber dado 50 de probabilidad para ambos factores, daño y regeneración, sobre todo a nivel poblacional se ve un efecto en declive de la salud, en tan solo veinticuatro meses decayó a un 84 porciento. Esto sin tomar en cuenta como factor las edades de los individuos, además, que estos parámetros de costumbres representan a individuos que no se descuidan

completamente, hacen ejercicio semanalmente en un promedio de 3 a 4 días por semana, comen bien la misma cantidad de días, descansan bien (tomando en cuenta el ritmo actual de vida) y por lo general se encuentran en estado anímico promedio, ver figura 3.

En contraste con la iteración que se hizo con un individuo con los mismos parámetros, figura 4, tienen un resultado similar pero este a los treinta y seis meses. Tomando como ejemplo de los genes, los cuales se han reportado que son determinantes y que se relacionan con la expresión de otros, el de obesidad, en México solo en los adultos se encuentra que el 73 porciento [4] padecen obesidad, aunque esta cifra fue del 2016, no se descarta que haya aumentado. Por tal motivo se hizo otra iteración donde se colocaron cifras con un poco de realidad para observar el avance de estas enfermedades, figura 5.

En esta figura se observan resultados del actual ritmo de vida, aunque solo limitándose a la afectación de estos 4 genes y sin relacionar el efecto de uno sobre el otro.

Los malos hábitos se potencian y por ende la regeneración es menor, aunque en la misma intensidad su frecuencia disminuye. Para la obesidad se coloca un marcado valor de 77 porciento de presencia en la población, depresión con un 70 porciento, falta de descanso con un 62,5 porciento y con un alto nivel de sedentarismo de un 75 porciento. Aunque suenan valores muy marcados en la actualidad marcan una tendencia hacia al alza de estos valores [3] [5]. En tan solo 2 años la salud de la población disminuyó considerablemente de un 100 porciento a tan solo un 60 porciento, indicando que con las condiciones actuales y sin hábitos que nos puedan que nos ayuden a mejorar la salud, sumando que las costumbres que tomamos de nuestra familia, amigos, compañeros, etc., no siempre nos benefician ocasionando que se repriman o se expresen genes que

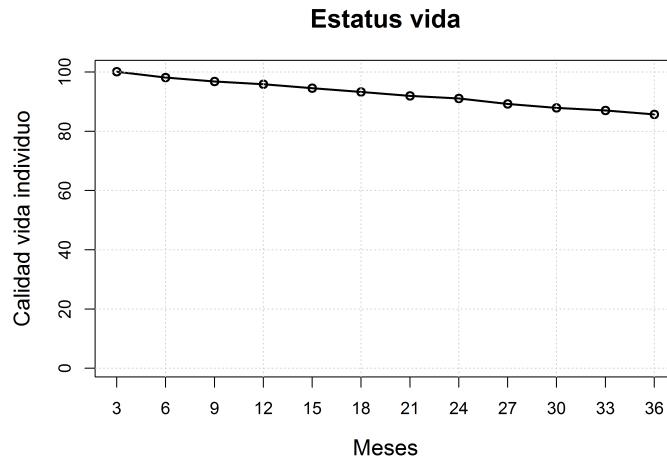


Figura 4: Resultados de la iteración de un solo individuo bajo los mismos parámetros de salud que la población anterior, solo que se realizó en un periodo de 3 años.

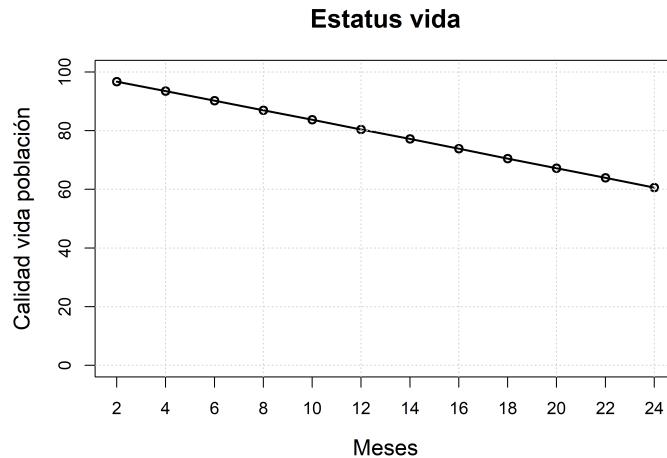


Figura 5: Resultados de una iteración con parámetros elevados y con cercanía a la realidad, en especialidad con la obesidad. Obteniendo cambios drásticos en la calidad de vida.

regulan el correcto funcionamiento celular [2] [4].

Al final del experimento se realizó una prueba para el código y poder saber si este tiene un diseño correcto. Para el cual se corrió en paralelo y posteriormente en secuencial, se midieron los tiempos y se compararon para observar si hubo mejora al momento de realizar la paralelización, figura 6.

5. Conclusiones

La programación epigenética puede ser un mecanismo esencial para la realización de procesos evolutivos importantes o de forma alterna es algo que coincide con el tiempo con los efectos tanto positivos como negativos.

Se logró realizar un estudio de comportamiento de indivi-

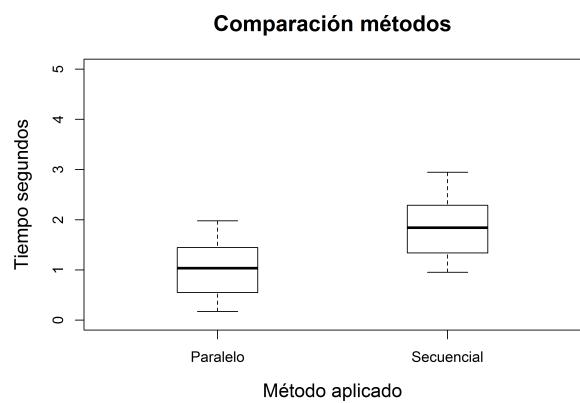


Figura 6: Comparación de métodos de análisis.

duos donde pierden calidad de vida correspondiendo a sus decisiones sin dejar a un lado la probabilidad de recuperarse.

Debido a los pocos factores que se tomaron en cuenta y a la complejidad de las interacciones entre factores que afectan la expresión genética, se debe de realizar un estudio con mayor profundidad y tomando en cuenta más factores. Se logró crear un código que puede simular lo que se buscaba.

Se comprobó la utilidad de este software para la aplicación de este tipo de análisis con el cual se pueden obtener información exacta, además de las aplicaciones que esta puede tener en el área de la epigenética son muy importantes.

Referencias

- [1] Ensanut Mc 2016. Encuesta nacional de salud y nutrición de medio camino 2016. *Instituto Nacional de Salud Pública*, 2016.
- [2] Casanello Paola et al. Epigenética y obesidad. *Revista chilena de pediatría*, 2016.
- [3] Cigarroa1 Igor et al. Efectos del sedentarismo y obesidad en el desarrollo psicomotor en niños y niñas: Una revisión de la actualidad latinoamericana. *Universidad y Salud*, 2016.
- [4] Krausea Bernardo et al. Conceptos generales de epigenética: proyecciones en pediatría. *Revista chilena de pediatría*, 2016.
- [5] Rivera Juan et al. Obesidad en méxico recomendaciones para una política de estado”. *Universidad y Salud*, 2013.
- [6] Abraham Azael Morales Juárez. Código simulación epigenética, 2019. URL <https://github.com/ELAZA27/Simulaciones>
- [7] Satu Elisa Schaeffer. Práctica 3: Teoría de colas, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>