

# **Maestría en Ciencias de la Ingeniería con Orientación en Nanotecnología**

**Simulación Computacional de Nanomateriales**

**Portafolio de Prácticas**

Docente: Dra. Satu Elisa Schaeffer

Alumno: José Ángel García Cedillo (1983175)

# Práctica 1

## Movimiento Browniano

### Revisión y Retroalimentación

- La práctica no la entregué pues fue difícil para mí adaptarme al lenguaje de R, debo poner más atención a que cosas se me dificultan más para distribuirle más tiempo y esfuerzo.
- Se implementaron el uso de varias figuras en un mismo renglón con el comando \”subfloat”

# Práctica 1: Movimiento Browniano

3175

29 de Enero de 2019

## 1. Introducción

Una partícula suficientemente pequeña como un grano de polen, inmersa en un líquido, presenta un movimiento aleatorio, observado primeramente por el botánico Brown en el siglo XIX. El movimiento browniano puede explicarse a escala molecular por una serie de colisiones en una dimensión en la cual, pequeñas partículas experimentan choques con una partícula mayor [1].

## 2. Objetivo

Observar el efecto de la dimensión en la probabilidad de regreso al origen en el fenómeno de movimiento browniano.

## 3. Metodología

Usando de base el código proporcionado [2], se simuló el movimiento de una partícula que se mueve de manera aleatoria varias dimensiones (de 1 hasta 8), variando el número de pasos que ésta da de  $2^6$  incrementando en linealmente en uno la potencia hasta el 12 y dando treinta repeticiones. El proceso se parallelizó mediante el uso de la librería *Parallel* tomando como base el código [3]:

```
1 for (dimension in 1:dim) {  
2   pOrigen <- 0  
3   clusterExport(cluster , "dimension")  
4   resultadoP <- parSapply(cluster , 1:repetir ,  
5     function(r) {  
6       pos <- rep(0, dimension)  
7       mayor <- 0  
8       pOrigen <- 0  
9       for (t in 1:duracion) { #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)) #Regres al origen?  
17         {  
18           contador <- contador + 1  
19         }
```

## 4. Resultados

Los resultados en las gráficas de las figuras 1 y 2 muestran los porcentajes de ocasiones en las que la partícula simulada volvió al origen en cada repetición con determinados pasos. Se observa en cada variación de cantidad de pasos que conforme aumentan las dimensiones disminuye la probabilidad de que la partícula vuelva a donde empezó.

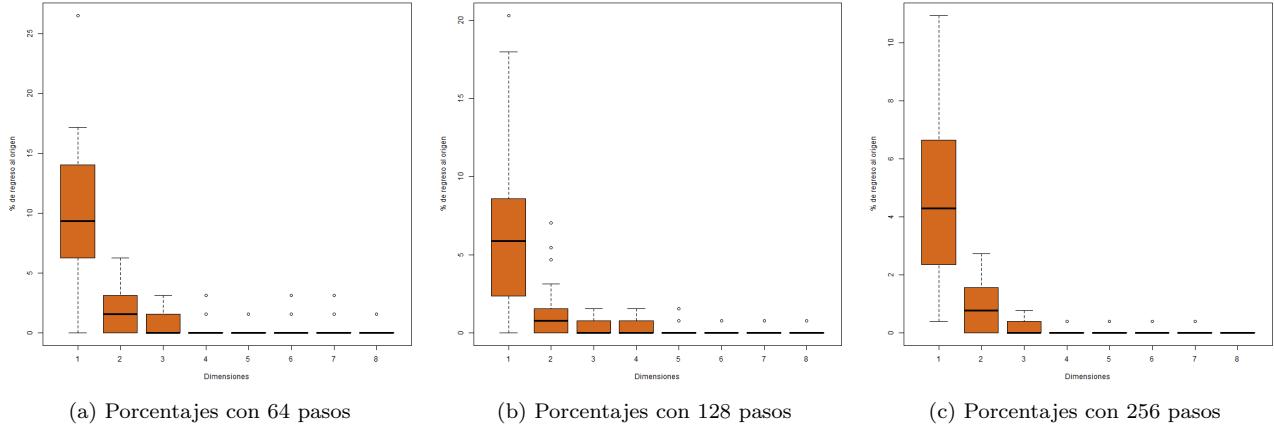


Figura 1: Gráficas de variaciones de pasos

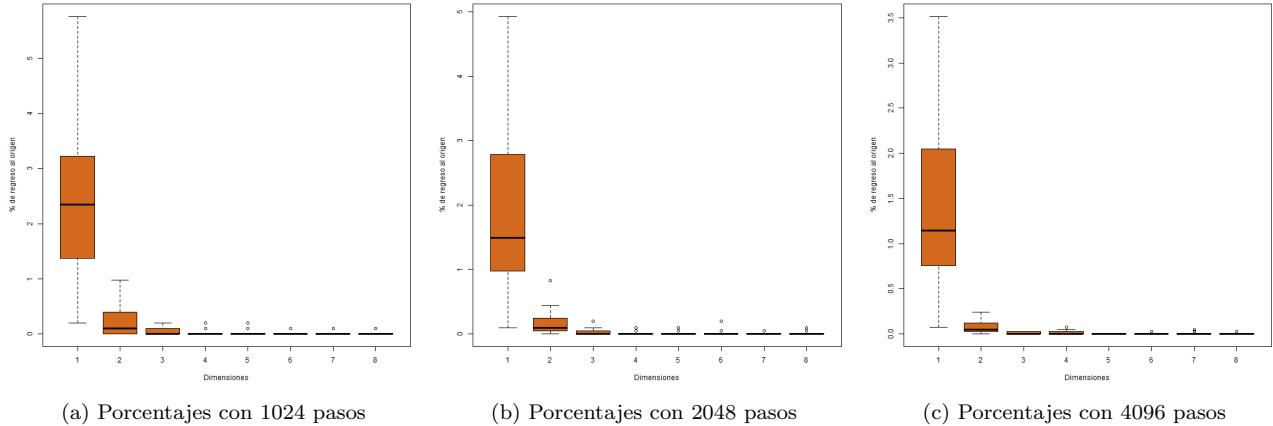


Figura 2: Gráficas de variaciones de pasos

## 5. Conclusiones

Las gráficas muestran una clara tendencia a no volver al origen cuando es mayor número de dimensiones, ésto es dado que la partícula al tener mas opciones para moverse la probabilidad de que regrese al lugar de donde se acaba de mover se va dividiendo, haciendo mas improbable el regresar.

## Referencias

- [1] Ángel Franco García. Movimiento browniano. Página Web: Fenómenos de transporte, 2017. URL <http://www.schuh.es/sbweb/fisica/transporte/brownian/brownian.htm>
- [2] Elisa Schaeffer. Práctica 1: Movimiento browniano. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p1.html>
- [3] Marco Antonio Guajardo Vigil. Movimiento browniano. Repositorio Source Forge, 2019. URL <https://sourceforge.net/p/simulaciondesistemas/code/ci/master/tree/P1/>

# Práctica 2

## Autómata Celular

### **Revisión y Retroalimentación**

- La práctica tampoco se entregó aún no me adaptaba a R y Latex.
- Se realizó la práctica usando paralelismo.

# Práctica 2: Autómata celular

3175

5 de Febrero de 2019

## 1. Introducción

Un autómata celular es un modelo matemático para un sistema dinámico que evoluciona a pasos discretos y tienen diversas aplicaciones como por ejemplo en la criptografía para el cifrado de información y el reparto de secretos. Sus orígenes se encuentran en los años 50's con el matemático *John Von Neumann* quien fue muy importante en la creación del primer ordenador [\[1\]](#).

## 2. Objetivo

Realizar un experimento en el cual se determine el número de iteraciones necesarias hasta que mueran todas las celdas variando la probabilidad inicial de que la celda viva o muera al inicio.

## 3. Metodología

Usando de base el código proporcionado [\[3\]](#), el experimento se desarrolla utilizando la librería *sna* en una malla de  $30 \times 30$  con una variación de probabilidad de la celda viva al inicio de 0 a 1 en pasos de 0,1. En la cual la celda vive si tres de sus vecinos viven, de lo contrario muere.

```
1 dim <- 30
2 num <- dim^2
3 lim <- 100
4 repeticiones <- 30
5 probabilidades <- seq(0,1,0.10)
6
7 cluster <- makeCluster(detectCores() - 1)
8 clusterExport(cluster, "dim")
9 clusterExport(cluster, "paso")
```

Revisando el código [\[2\]](#), se paralelizó la etapa de las treinta iteraciones llamando a las funciones “dim” y “paso” a los núcleos por medio del paquete *parSapply* y se realizaron treinta repeticiones y se colocaron condiciones para detener cuando y si todos están muertos o vivos:

```
1 for(p in probabilidades){
2   iteraciones <- list()
3   actual <- matrix(1 * (runif(num) < p), nrow=dim, ncol=dim)
4   for (rep in 1:repeticiones){
5     if(sum(actual) == 0){ #todos se generaron muertos
6       iteraciones <- c(iteraciones, 0)
```

```

7   iteraciones <- unlist(iteraciones)
8 } else if(sum(actual) == num){ #todos estan vivos
9   iteraciones <- c(iteraciones , 0)
10  iteraciones <- unlist(iteraciones)

```

## 4. Resultados

Los resultados muestran en la figura 1 una gráfica sin datos en los valores de probabilidad de 0 y 1. Y una mayor presencia de datos al centro de los valores.

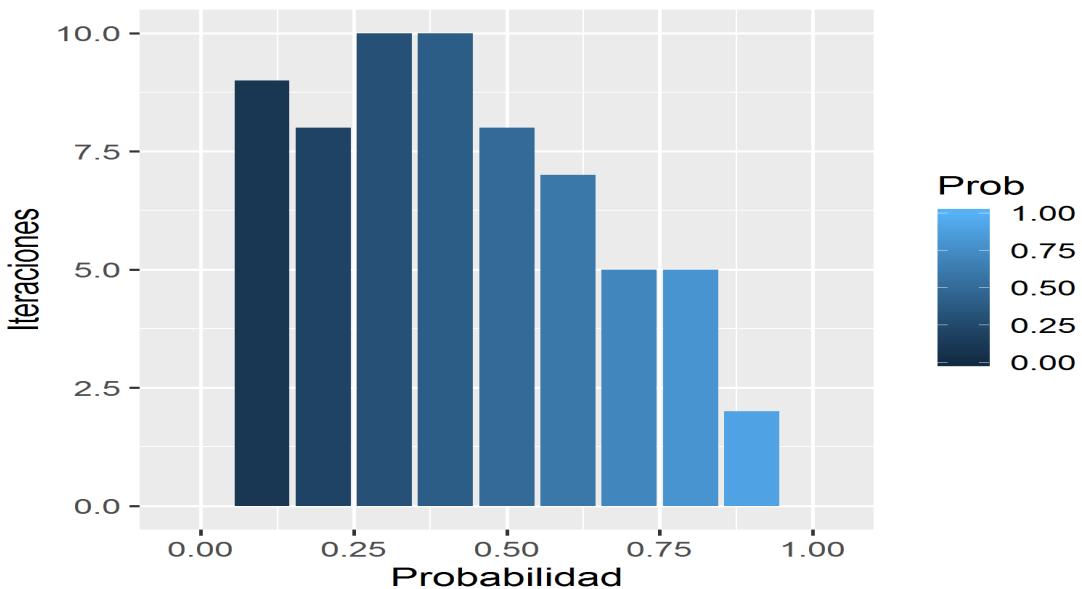


Figura 1: Gráfica de iteraciones con las variaciones de probabilidad de vida al inicio

## 5. Conclusiones

El comportamiento de los datos deja claro que es mas favorecida una probabilidad en el centro de los valores manejados (0.3, 0.4, 0.5) y una ausencia de iteraciones en los valores extremos, debido a la regla de supervivencia, que exige que deben estar tres vecinos vivos para que pueda vivir y al haber ninguno o todos al principio no puede continuar. Parece hasta lógico pensar que a los valores de probabilidad mejor distribuidos mejoran las oportunidades de supervivencia del Autómata Celular.

## Referencias

- [1] Angela Rojas Matas Alberto Cano Rojas. Autómatas celulares y aplicaciones. *Revista Iberoamericana de Educación Matemática*, 2016.
- [2] Edson Raúl Cepeda Márquez. Autómata celular. Repositorio Source Forge, 2019. URL <https://sourceforge.net/p/simulaciondesistemas/code/ci/master/tree/P1/>.

- [3] Elisa Schaeffer. Práctica 2: Autómata celular. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p2.html>.

# Práctica 3

## Teoría de Colas

### **Revisión y Retroalimentación**

- Se revisó la ortografía.
- Se arreglaron problemas con la bibliografía y título de la gráfica.

# Teoría de colas

Jose Angel García Cedillo 1983175

5

12 de febrero de 2019

## 1. Introducción

La teoría de colas describe un sistema en el que un conjunto de clientes o procesos llegan a un sistema buscando un servicio, dependiendo de la actividad del sistema la demanda puede o no ser cubierta inmediatamente, ésto provoca que se forme una cola de espera de clientes esperando a ser atendidos [3]. Esta teoría estudia factores como el tiempo de espera medio en las colas o la capacidad de trabajo del sistema sin colapsar.

El matemático danés Agner Erlang fué el primero que estudió éste sistema desde un punto de vista matemático, pues fue el primero en publicar un artículo sobre la teoría de colas, el cual abordaba el problema de dimensionamiento de líneas telefónicas para su servicio de llamadas. [2]

## 2. Objetivo

Analizar el comportamiento de la velocidad de procesamiento de una serie de datos, a modo de un sistema de colas variando el número de servidores o núcleos, criterios de análisis y la proporción de datos que contienen al vector.

## 3. Metodología

Se toma como proceso para el sistema de colas una función que revisa mediante varias condiciones si el número es primo o no [1] y se revisa cuánto le toma a el procesador determinar cierta cantidad de datos de un vector de determinada magnitud con varias repeticiones y obteniendo al final un promedio de éstos tiempos. Para el experimento se van a variar factores como: número de núcleos asignados para la tarea: de 1 a el máximo de núcleos lógicos, proporción de datos: igual cantidad de primos y no primos, 3/4 partes de datos eran primos y 1/4 parte primos; el criterio de procesar el vector de datos: de mayor a menor, de menor a mayor y en orden aleatorio. Así mismo, se definió el tamaño del vector de 100 datos con un tamaño de 6 dígitos y 20 repeticiones para dar más certeza a los datos recolectados. El experimento fue realizado en una computadora con procesador core i7 de dos núcleos reales y 4 lógicos.

## 4. Resultados

Se puede observar en los datos obtenidos en la gráfica 1, el promedio de tiempo mínimo ubicado en la corrida menor cantidad de primos en el vector, de menor a mayor utilizando 2 núcleos y el máximo en la corrida con mayor cantidad de números primos, de mayor a menor y utilizando solamente 1 núcleo.

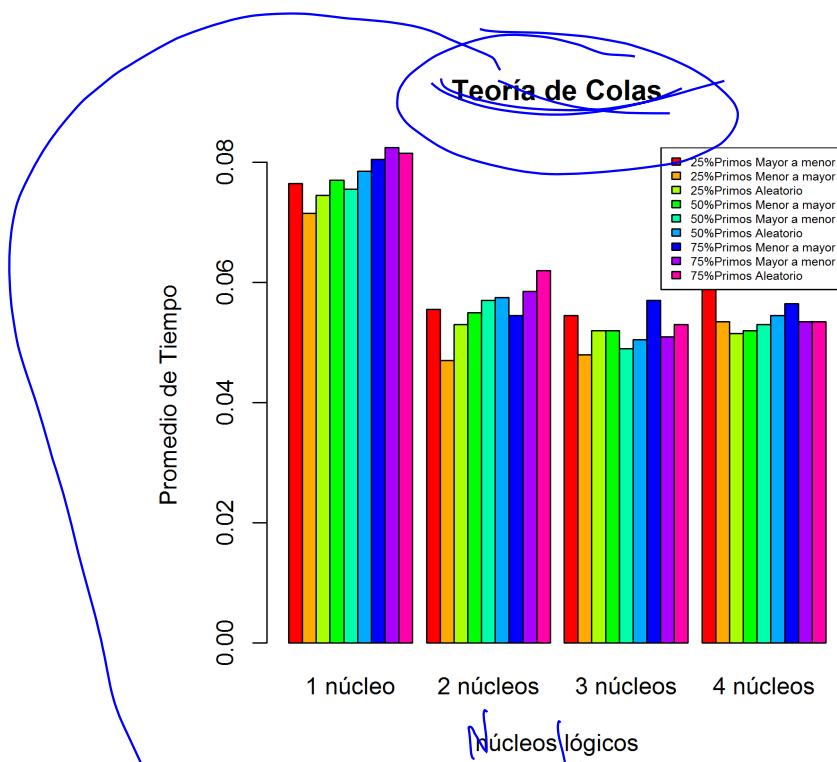


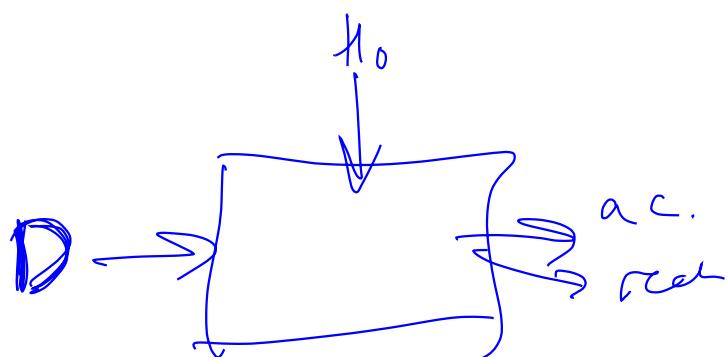
Figura 1. Gráfica de tiempos con variaciones de tipos de datos y núcleos asignados

## 5. Conclusiones

En base a los datos obtenidos se puede notar una tendencia a aumentar el tiempo de procesamiento en un vector de datos con más numeros primos, analizados de mayor a menor y utilizando menos núcleos para la tarea.

## Referencias

- [1] 01 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.
- [2] 02 2019. URL <https://plus.maths.org/content/os/issue2/erlang/index>.
- [3] Gerardo Fabian Peraza Siqueiros. *Tesis*, pages 9–10.



# Práctica 3: Teoría de Colas

3175

12 de Febrero de 2019

## 1. Introducción

La teoría de colas describe un sistema en el que un conjunto de clientes o procesos llegan a un sistema buscando un servicio, dependiendo de la actividad del sistema la demanda puede o no ser cubierta inmediatamente, ésto provoca que se forme una cola de espera de clientes esperando a ser atendidos [3]. Esta teoría estudia factores como el tiempo de espera medio en las colas o la capacidad de trabajo del sistema sin colapsar.

El matemático danés Agner Erlang fué el primero que estudió éste sistema desde un punto de vista matemático, pues fue el primero en publicar un artículo sobre la teoría de colas, el cual abordaba el problema de dimensionamiento de líneas telefónicas para su servicio de llamadas [1].

## 2. Objetivo

Analizar el comportamiento de la velocidad de procesamiento de una serie de datos, a modo de un sistema de colas variando el número de servidores o núcleos, criterios de análisis y la proporción de datos que contienen al vector.

## 3. Metodología

Se toma como proceso para el sistema de colas una función que revisa mediante varias condiciones si el número es primo o no [2] y se revisa cuanto le toma a el procesador determinar cierta cantidad de datos de un vector de determinada magnitud con varias repeticiones y obteniendo al final un promedio de éstos tiempos. Para el experimento se van a varían factores como: número de núcleos asignados para la tarea: de 1 a el máximo de núcleos lógicos, proporción de datos: igual cantidad de primos y no primos, tres cuartas partes de datos eran primos y 1/4 parte primos; el criterio de procesar el vector de datos: de mayor a menor, de menor a mayor y en orden aleatorio. Así mismo, se definió el tamaño del vector de 100 datos con un tamaño de 6 dígitos y 20 repeticiones para dar más certeza a los datos recolectados. El experimento fue realizado en una computadora con procesador core i7 de dos núcleos reales y cuatro lógicos.

## 4. Resultados

Se puede observar en los datos obtenidos en la gráfica 1, el promedio de tiempo mínimo ubicado en la corrida menor cantidad de primos en el vector, de menor a mayor utilizando 2 núcleos y el máximo en la corrida con mayor cantidad de números primos, de mayor a menor y utilizando solamente 1 núcleo.

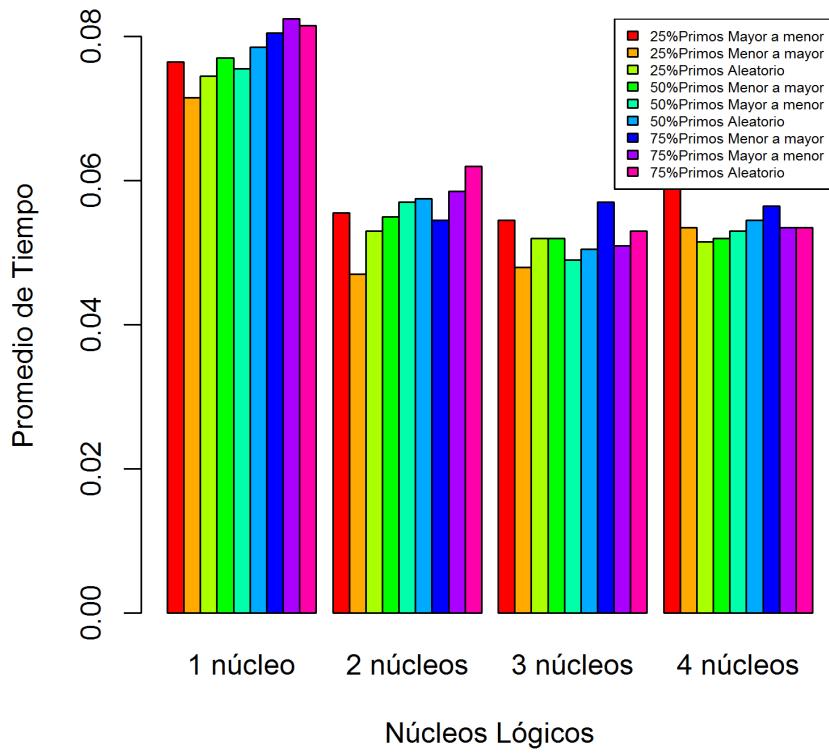


Figura 1: Gráfica de tiempos con variaciones de tipos de datos y núcleos asignados

## 5. Conclusiones

En base a los datos obtenidos se puede notar una tendencia a aumentar el tiempo de procesamiento en un vector de datos con más numeros primos, analizados de mayor a menor y utilizando menos núcleos para la tarea.

## Referencias

- [1] Plusadmin. Biography: Anger krarup erlang. Página Web: +Plus Magazine, 1997. URL <https://plus.maths.org/content/os/issue2/erlang/index>.
- [2] Elisa Schaeffer. Práctica 3: Teoría de colas. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p3.html>.
- [3] Gerardo Fabian Peraza Siqueiros. Introducción a la teoría de colas y su simulación. Tesis: Universidad de Sonora, 2013.

# Práctica 4

## Diagramas de Voronoi

### Revisión y Retroalimentación

- Se revisó la ortografía y se arreglaron problemas con las comillas usando `` "", se utilizaron signos de \$ para marcar variables y se utilizó la diagonal para proteger el signo %.
- Se agregó fragmentos del código R en la metodología.
- Se arreglaron problemas con la bibliografía.

# Diagramas de Voronoi

3175

19 de febrero de 2019

## 1. Introducción

También llamados planos de Thiessen, son una forma geométrica que permite construir una partición del plano euclídeo. Son llamados diagramas de Voronoi pues fueron estudiados por el matemático ruso Gueorgui Voronói en 1907 [2].

## 2. Objetivo

Evaluando si el tamaño de la zona de la matriz y la cantidad de semillas iniciales realizando variaciones de estos factores determina el tamaño de una grieta que se propaga con una preferencia por los límites de los clusters en la zona.

## 3. Metodología

Se inicia con el código proporcionado en la práctica 4 [1] para la generación del diagrama de voronoi y el modelo de propagación de la grieta. Posteriormente se realizaron iteraciones con variaciones de  $n=$ zona (30, 50, 70) y  $k=$ número de semillas iniciales (10, 12, 14, 16). Se definieron 200 repeticiones para que la muestra sea representativa y se evaluó en base a la longitud que la grieta presentada. Finalmente se realiza un análisis ANOVA de dos factores a los datos para determinar si la varianza de los datos entre los tratamientos es significativa.

## 4. Resultados

En los resultados representados en la gráfica 1 se puede observar que los promedios de la longitud de las grietas son cercanos entre los distintos tipos de tratamientos, sin embargo, se observa una tendencia ascendente en los máximos cuando el número de semillas aumenta así como el tamaño de la zona.

Se realizaron las hipótesis nulas en donde “El tamaño de la zona no determina el tamaño de la grieta formada”, “El número de semillas no determina el tamaño de la grieta formada” y “La relación entre los dos factores (El tamaño de la zona y de la semilla en conjunto no determina el tamaño de la grieta)”.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
TZonas	2	2657	1328.5	24.228	3.83e-11
Tsemillas	3	1011	336.9	6.144	0.00037
TZonas:Tsemillas	6	411	68.5	1.249	0.27794
Residuals	2388	130939	54.8		

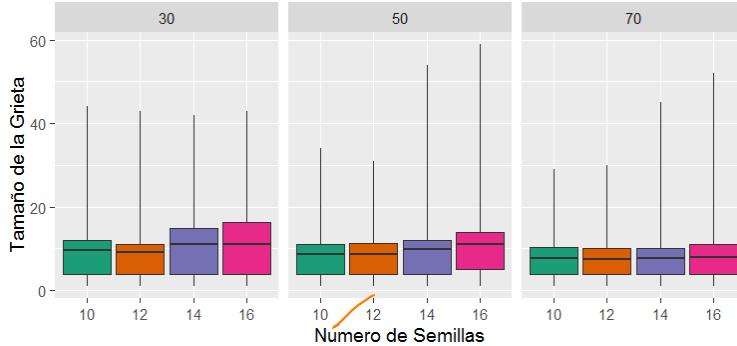


Figura 1: Gráfica de largos de grieta alternando en los diferentes tratamientos de tamaño de zona y cantidad de semillas iniciales

Debido a los resultados del análisis ANOVA se pueden descartar las hipótesis nulas de los tratamientos, sin embargo, se acepta la hipótesis nula en la que la variación de el tamaño de zona y la cantidad de semillas no influye en el tamaño de la grieta con una probabilidad de 27.7%

## 5. Conclusiones

Se puede concluir en base a los resultados que los tamaños de las grietas si están influenciadas predominantemente por el tamaño de las zonas y en cierta medida por la cantidad de semillas, debido a que ambas variables determinan el medio en dimensiones y configuración respectivamente por el cual la grieta se propaga. Adicionalmente, no hay variación significativa en los resultados entre las iteraciones de las combinaciones de tratamientos, se puede observar que los promedios de los longitudes de las grietas están muy cercanos entre si.

## Referencias

- [1] 01.2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p4.html>.
- [2] M. Berg, de, O. Cheong, M.J. Kreveld, van, and M.H. Overmars. *Computational geometry: algorithms and applications*. Springer, Germany, 3rd ed edition, 2008. ISBN 978-3-540-77973-5. doi: 10.1007/978-3-540-77974-2.

Even Kren + }

# Práctica 4: Diagramas de Voronoi

3175

19 de Febrero de 2019

## 1. Introducción

Los diagramas de Voronoi son también llamados planos de Thiessen y son una forma geométrica que permite construir una partición del plano euclídeo. Son llamados diagramas de Voronoi pues fueron estudiados por el matemático ruso Gueorgui Voronói en 1907 [1].

## 2. Objetivo

Evaluar si el tamaño de la zona de la matriz y la cantidad de semillas iniciales realizando variaciones de estos factores determina el tamaño de una grieta que se propaga con una preferencia por los límites de los clusters en la zona.

## 3. Metodología

Se inicia con el código proporcionado en la práctica 4 [2] para la generación del diagrama de Voronoi y el modelo de propagación de la grieta. Posteriormente se realizaron iteraciones con variaciones de  $n$ =zona (30, 50, 70) y  $k$ =número de semillas iniciales (10, 12, 14, 16). Se definieron 200 repeticiones para que la muestra sea representativa y se evaluó en base a longitud que la grieta presentada. Finalmente se realiza un análisis ANOVA de dos factores a los datos para determinar si la varianza de los datos entre los tratamientos es significativa.

```
1 Zonas <- c(30, 50, 70)
2 Semillas <- c(10, 12, 14, 16)
3 Info <- data.frame()
4 Largo <- c()
5 Nys <- c()
6 Kys <- c()
7 for (n in Zonas) {
8   for (k in Semillas){
9     zona <- matrix(rep(0, n * n), nrow = n, ncol = n)
10    x <- rep(0, k) # ocupamos almacenar las coordenadas x de las semillas
11    y <- rep(0, k) # igual como las 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
1
2 registerDoParallel( makeCluster( detectCores() - 1) )
3 largos <- foreach( r = 1:200 , .combine=c ) %dopar% propaga( r )

```

## 4. Resultados

En los resultados representados en la gráfica 1 se puede observar que los promedios de la longitud de las grietas son cercanos entre los distintos tipos de tratamientos, sin embargo, se observa un tendencia ascendente en los máximos cuando el número de semillas aumenta así como el tamaño de la zona.

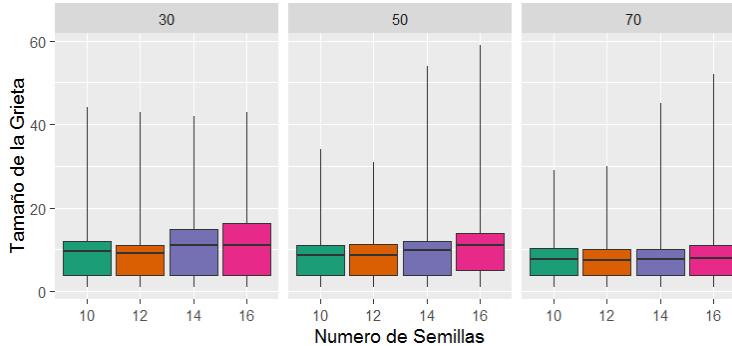


Figura 1: Gráfica de largos de grieta alternando en los diferentes tratamientos de tamaño de zona y cantidad de semillas iniciales

Se realizaron las hipótesis nulas en donde: “El tamaño de zona no determina el tamaño de la grieta formada”, “El número desemillas no determina el tamaño de la grieta formada” y la relación entre los dos factores “El tamaño de la zona y de la semilla en conjunto no determina el tamaño de la grieta”.

```

1
2 Df Sum Sq Mean Sq F value Pr(>F)
3 TZonas 2 2657 1328.5 24.228 3.83e-11
4 Tsemillas 3 1011 336.9 6.144 0.00037
5 TZonas:Tsemillas 6 411 68.5 1.249 0.27794
6 Residuals 2388 130939 54.8

```

Debido a los resultados del análisis ANOVA se pueden descartar las hipótesis nulas de los tratamientos, sin embargo, se acepta la hipótesis nula en la que la variación de el tamaño de zona y la cantidad de semillas no influye en el tamaño de la grieta con una probabilidad de 27.7%.

## 5. Conclusiones

Se puede concluir en base a los resultados que los tamaños de las grietas si están influenciadas predominantemente por el tamaño de las zonas y en cierta medida por la cantidad de semillas, debido a que ambas variables determinan el medio en dimensiones y configuración respectivamente por el cual la grieta se propaga. Adicionalmente, no hay variación significativa en los resultados entre las iteraciones de las combinaciones de tratamientos, se puede observar que los promedios de los longitudes de las grietas están muy cercanos entre si.

## Referencias

- [1] M. Berg, de, O. Cheong, M.J. Kreveld Van, and M.H. Overmars. *Computational geometry : algorithms and applications*. Springer, Germany, 3rd ed edition, 2008. ISBN 978-3-540-77973-5. doi: 10.1007/978-3-540-77974-2.
- [2] Elisa Schaeffer. Práctica 4: Diagramas de Voronoi. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p4.html>.

# Práctica 5

## El Método Monte-Carlo

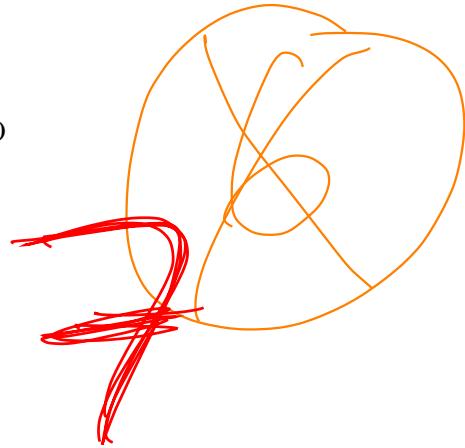
### Revisión y Retroalimentación

- Se corrigieron errores al colocar ecuaciones usando “\frac{.}{.}” y los signos de \$ para valores numéricos.
- Se arreglaron problemas con la bibliografía, faltaban nombres.
- Se revisó la ortografía.

# Método Monte-Carlo

3175

26 de febrero de 2019



## 1. Introducción

Los métodos Monte-Carlo son una colección de técnicas que permiten obtener soluciones de problemas matemáticos o físicos por medio de pruebas aleatorias repetidas. En la práctica, las pruebas son resultados de ciertos cálculos realizados con números aleatorios [2]. Un ejemplo de implementación de los métodos Monte-Carlo es la simulación de fenómenos físicos en la investigación en física de partículas y es una herramienta esencial para el diseño de las instalaciones y detectores, así como para el análisis de datos de resultados físicos [3].

## 2. Objetivo

Determinar el tamaño de muestra requerido para obtener una precisión determinada por el número de decimales correctos en una aproximación obtenida por el método Monte-Carlo.

## 3. Metodología

Se inicia con el código proporcionado en la práctica 5 [1] con el cual se genera una aproximación de la integral:

$$\int_{-7}^3 \frac{1}{e^x + e^{-x}} dx \quad (1)$$

Se introduce la función y se normaliza multiplicandola por  $2/\pi$ :

```
1 f <- function(x) { return(1 / (exp(x) + exp(-x))) }
2 suppressMessages(library(distr))
3 g <- function(x) { return((2 / pi) * f(x)) }
```

posteriormente se crea un generador, se definen los límites de los valores que son generados y que son de interés de la integral:

```
1 generador <- r(AbscontDistribution(d = g))
2 desde <- 3
3 hasta <- 7
4 parte <- function() {
5   valores <- generador(pedazo)
6   return(sum(valores >= desde & valores <= hasta))}
```

Finalmente se paralleliza tomando diferentes tamaños de muestra (100, 1000, 10000, 100000, 1000000), definiendo 50 repeticiones en grupos de 10 para poder observar un valor representativo en los resultados ya que los valores son pseudo

$$e^x \rightarrow e^x$$
$$e^{2x} \rightarrow e^{2x}$$

$$e^{\{2x\}} e^{2x}$$

aleatorios. El resultado obtenido en cada iteración finalmente es comparado con el resultado obtenido de Wolfram Alpha para poder observar su precisión en base a los decimales de la diferencia.

```

1 for (resultado in cant){
2   for (cantidad in repeticiones) {
3     for (repetir in 1:10) {
4       pedazo <- resultado
5       cuantos <- cantidad
6       montecarlo <- foreach(i = 1:cuantos, .combine=c) %dopar% parte()
7       stopImplicitCluster()
8       integral <- sum(montecarlo) / (cuantos * pedazo)
9       aproximar<-((pi / 2) * integral)
10      diferencia<-rbind(diferencia , c(abs(aproximar-0.0488340), resultado , cantidad))

```

## 4. Resultados

En los resultados representados en la figura 1 y 2 (su acercamiento) podemos observar el área debajo de las líneas coloreadas la precisión en cantidad de dígitos (entendiendo la precisión del primer dígito sobre la línea del segundo dígito)

Entonces, se observa que para esperar una aproximación con una precisión de dos dígitos es necesario usar una muestra de tamaño entre 100 y 1,000, para una de tres dígitos entre 10,000 y 100,000 y para cuatro 1,000,000.

## 5. Conclusiones

Observando los resultados se puede decir que el tamaño de muestra influye directamente en la precisión de la aproximación, siendo que a mayor tamaño de muestra la precisión aumenta. Al ser el método Monte-Carlo una generación de valores pseudo aleatorios dentro de los mismos grupos de variaciones de muestra se presentaron amplios rangos de resultados por lo que es de gran valor realizar repeticiones para poder observar un valor satisfactorio.

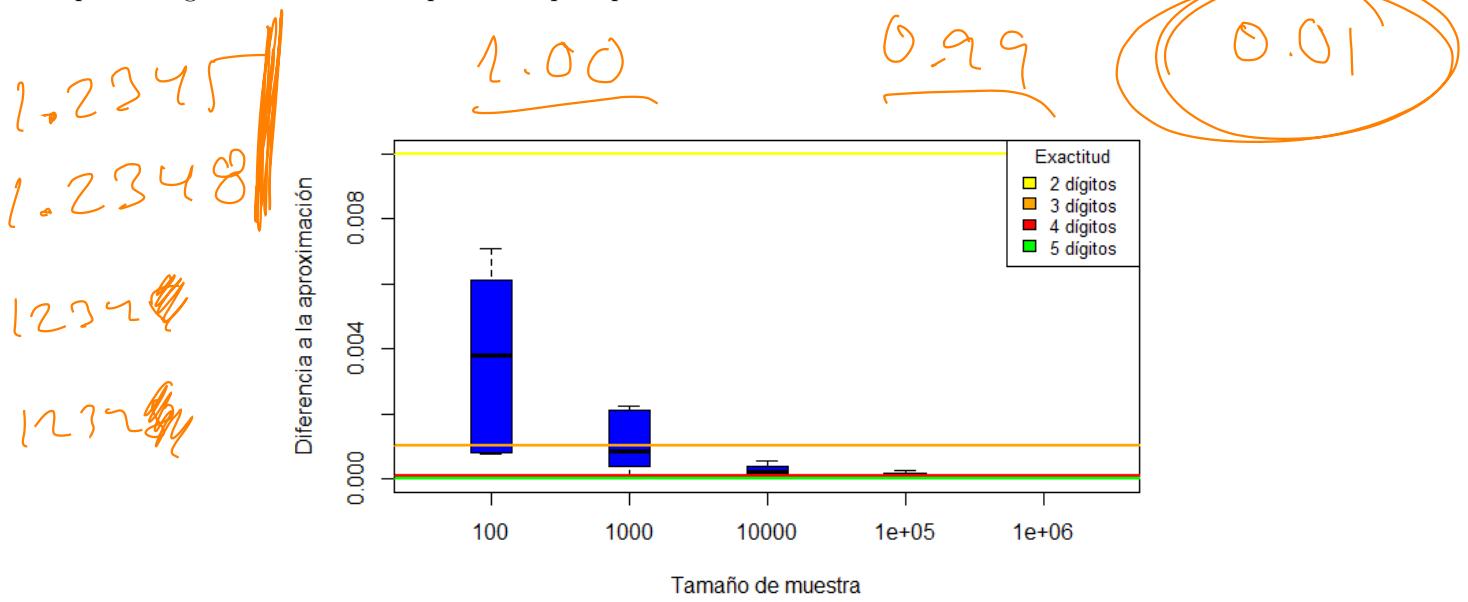


Figura 1: Gráfica con las diferencias entre el valor real y la aproximación de Monte-Carlo

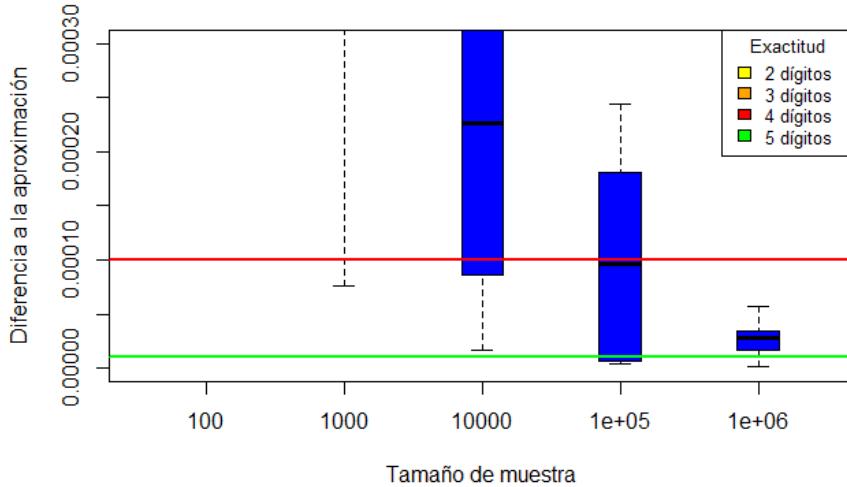


Figura 2: Acercamiento de la gráfica con las diferencias entre el valor real y la aproximación de Monte-Carlo

## 6. Reto 1

El primer reto consiste en implementar la estimación del valor  $\pi$  con el método de Kurt , paralelizando con el tamaño de muestra para encontrar la relación matemática entre éstas y la precisión obtenida en base a cantidad de decimales correctos.

La técnica de Kurt toma en cuenta el área del cuadrado =  $4r^2$  y el área del círculo =  $\pi(r^2)$  por lo que al efectuar la combinación y obtener un coeficiente se obtiene  $\pi/4$  pudiendo así obtener el valor de  $\pi$  al multiplicarlo por cuatro.

Considerando un cuadrado con  $r=1$  se realiza la función Monte-Carlo para realizar la aproximación:

```

1 apropi=function() {
2   xs <- runif(replicas, min= -0.5, max= 0.5)
3   ys <- runif(replicas, min= -0.5, max= 0.5)
4   in.circle <- xs^2 + ys^2 <= 0.5^2
5   mc.pi <- (sum(in.circle)/replicas)*4
6   return(mc.pi)
}

```

Finalmente se paralelizó con las siguientes consideraciones: variaciones de tamaño de muestra fueron 10, 100, 1000, 10000, 100000 y 1000000, se realizaron 500 réplicas en grupos de 20 repeticiones.

Los resultados obtenidos observados en las figuras 3 y 4 (que proceden de la misma gráfica) se pueden observar los rangos de la exactitud de cifras en base a la diferencia de los valores de las aproximaciones comparadas con el valor de  $\pi$  real. Encontrando el tamaño de muestra para la precisión de una cifra con un tamaño de muestra de 10, de dos dígitos con un tamaño de muestra entre 100 y 1000, de tres dígitos con un tamaño de muestra de entre 10,000 y 100,000, de cuatro dígitos con un tamaño de muestra de 1,000,000.

Se puede concluir que el método de Monte-Carlo está basado en que a mayor cantidad de tamaño de muestra o puntos generados, se incrementa el acercamiento con el valor real. Se observó un comportamiento parecido al presentado a la aproximación del área de la integral. Debido a ésto en la práctica el tamaño de muestra está determinado por la aplicación del valor o dato que se busca obtener.

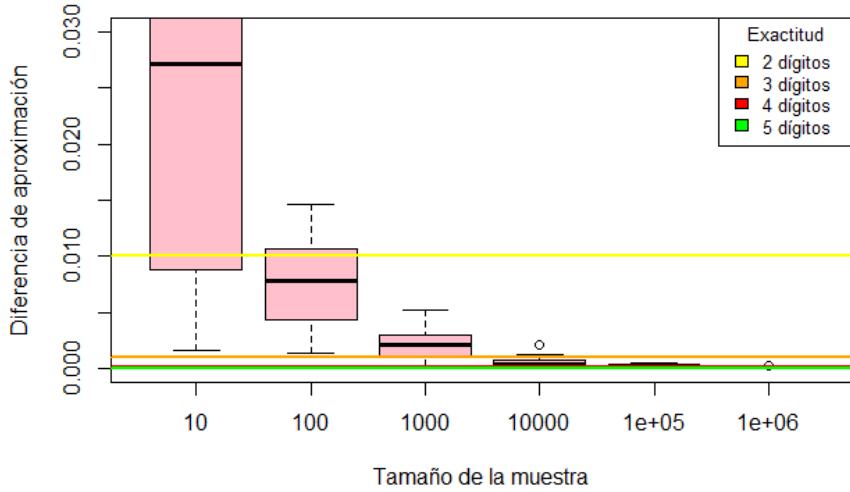


Figura 3: Gráfica de la relación de tamaño de muestra y aproximación de Monte-Carlo al valor  $\pi$

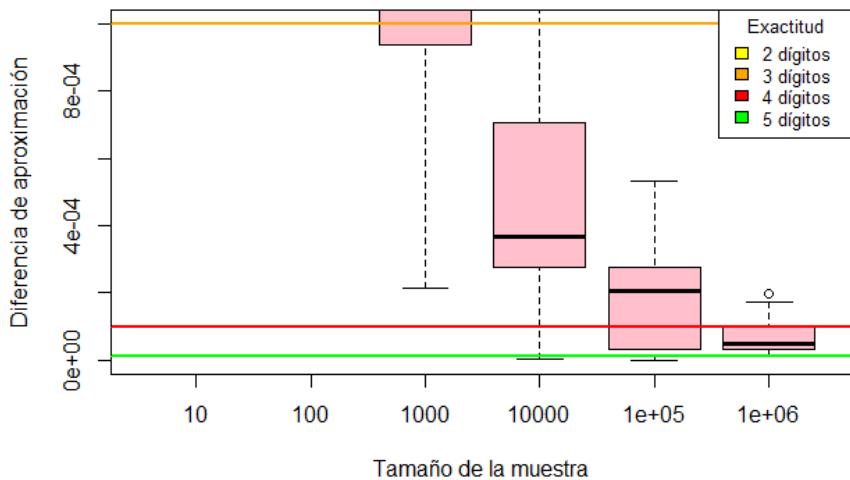


Figura 4: Acercamiento Gráfica de la relación de tamaño de muestra y aproximación de Monte-Carlo al valor  $\pi$

## Referencias

- [1] URL <https://elisa.dyndns-web.com/teaching/comp/par/p5.html>.
- [2] Ángel Franco García. Los métodos montecarlo. Página Web, 2009. URL [http://www.sc.ehu.es/sbweb/fisica\\_numerico/montecarlo/montecarlo.html](http://www.sc.ehu.es/sbweb/fisica_numerico/montecarlo/montecarlo.html).
- [3] Maria Grazia Pia and Georg Weidenspointner. Monte Carlo Simulation for Particle Detectors. Technical Report

arXiv:1208.0047, Agosto 2012. URL <http://cds.cern.ch/record/1471335>. ~~Comments:~~ CERN Council Open Symposium on European Strategy for Particle Physics, ~~10 - 12~~ September 2012, Krakow, Poland.

↑  
10 - 12

# Práctica 5: Método Monte-Carlo

3175

26 de Febrero de 2019

## 1. Introducción

Los métodos Monte-Carlo son una colección de técnicas que permiten obtener soluciones de problemas matemáticos o físicos por medio de pruebas aleatorias repetidas. En la práctica, las pruebas son resultados de ciertos cálculos realizados con números aleatorios [1]. Un ejemplo de implementación de los métodos Monte-Carlo es la simulación de fenómenos físicos en la investigación en física de partículas y es una herramienta esencial para el diseño de las instalaciones y detectores, así como para el análisis de datos de resultados físicos [2].

## 2. Objetivo

Determinar el tamaño de muestra requerido para obtener una precisión determinada por el número de decimales correctos en una aproximación obtenida por el método Monte-Carlo.

## 3. Metodología

Se inicia con el código proporcionado en la práctica 5 [3] con el cual se genera una aproximación de la integral:

$$\int_7^3 \frac{1}{e^x + e^{-x}} dx \quad (1)$$

Se introduce la función y se normaliza multiplicandola por  $2/\pi$ :

```
1 f <- function(x) { return(1 / (exp(x) + exp(-x))) }
2 suppressMessages(library(distr))
3 g <- function(x) { return((2 / pi) * f(x)) }
```

posteriormente se crea un generador, se definen los límites de los valores que son generados y que son de interés de la integral:

```
1 generador <- r(AbscontDistribution(d = g))
2 desde <- 3
3 hasta <- 7
4 parte <- function() {
5   valores <- generador(pedazo)
6   return(sum(valores >= desde & valores <= hasta))}
```

Finalmente se paralleliza tomando diferentes tamaños de muestra (100, 1000, 10000, 100000, 1000000), definiendo 50 repeticiones en grupos de 10 para poder observar un valor representativo en los resultados ya que los valores son pseudo

aleatorios. El resultado obtenido en cada iteración finalmente es comparado con el resultado obtenido de Wolfram Alpha para poder observar su precisión en base a los decimales de la diferencia.

```

1 for (resultado in cant){
2   for (cantidad in repeticiones) {
3     for (repetir in 1:10) {
4       pedazo <- resultado
5       cuantos <- cantidad
6       montecarlo <- foreach(i = 1:cuantos, .combine=c) %dopar% parte()
7       stopImplicitCluster()
8       integral <- sum(montecarlo) / (cuantos * pedazo)
9       aproximar<-((pi / 2) * integral)
10      diferencia<-rbind(diferencia , c(abs(aproximar-0.0488340), resultado , cantidad))

```

## 4. Resultados

En los resultados representados en la figura 1 y 2 (su acercamiento) podemos observar el área debajo de las líneas coloreadas la precisión en cantidad de dígitos (entendiendo la precisión del primer dígito sobre la línea del segundo dígito)

Entonces, se observa que para esperar una aproximación con una precisión de dos dígitos es necesario usar una muestra de tamaño entre 100 y 1,000, para una de tres dígitos entre 10,000 y 100,000 y para cuatro 1,000,000.

## 5. Conclusiones

Observando los resultados se puede decir que el tamaño de muestra influye directamente en la precisión de la aproximación, siendo que a mayor tamaño de muestra la precisión aumenta. Al ser el método Monte-Carlo una generación de valores pseudo aleatorios dentro de los mismos grupos de variaciones de muestra se presentaron amplios rangos de resultados por lo que es de gran valor realizar repeticiones para poder observar un valor satisfactorio.

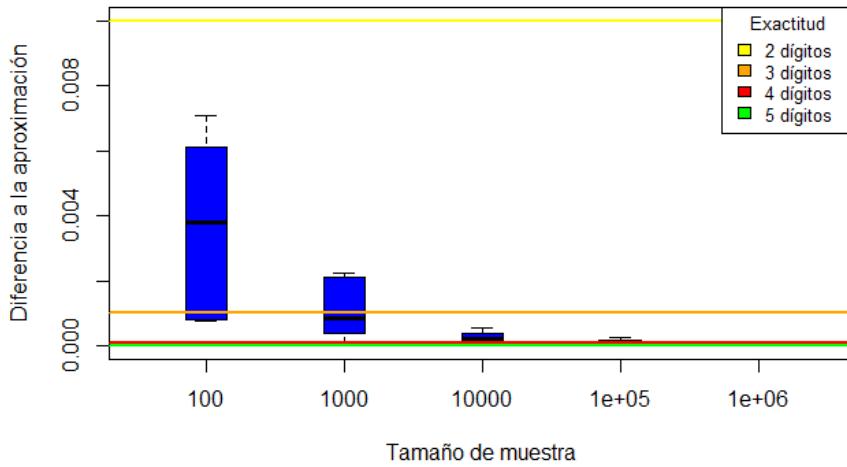


Figura 1: Gráfica con las diferencias entre el valor real y la aproximación de Monte-Carlo

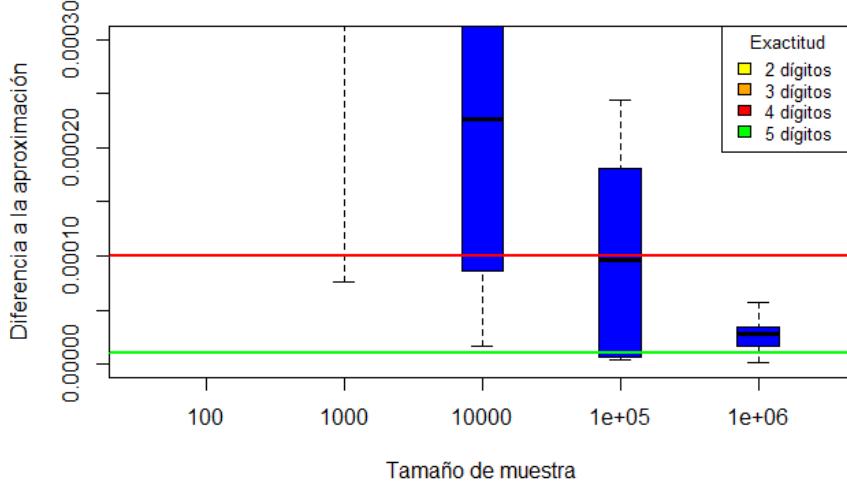


Figura 2: Acercamiento de la gráfica con las diferencias entre el valor real y la aproximación de Monte-Carlo

## 6. Reto 1

El primer reto consiste en implementar la estimación del valor  $\pi$  con el método de Kurt, paralelizando con el tamaño de muestra para encontrar la relación matemática entre éstas y la precisión obtenida en base a cantidad de decimales correctos.

La técnica de Kurt toma en cuenta el área del cuadrado  $= 4r^2$  y el área del círculo  $= \pi(r^2)$  por lo que al efectuar la combinación y obtener un coeficiente se obtiene  $\frac{\pi}{4}$  pudiendo así obtener el valor de  $\pi$  al multiplicarlo por cuatro.

Considerando un cuadrado con  $r = 1$  se realiza la función Monte-Carlo para realizar la aproximación:

```

1 apropi=function() {
2   xs <- runif(replicas , min= -0.5, max= 0.5)
3   ys <- runif(replicas , min= -0.5, max= 0.5)
4   in . circle <- xs^2 + ys^2 <= 0.5^2
5   mc . pi <- (sum(in . circle)/replicas)*4
6   return(mc . pi)

```

Finalmente se paralelizó con las siguientes consideraciones: variaciones de tamaño de muestra fueron 10, 100, 1000, 10000, 100000 y 1000000, se realizaron 500 réplicas en grupos de 20 repeticiones.

Los resultados obtenidos observados en las figuras 3 y 4 (que proceden de la misma gráfica) se pueden observar los rangos de la exactitud de cifras en base a la diferencia de los valores de las aproximaciones comparadas con el valor de  $\pi$  real. Encontrando el tamaño de muestra para la precisión de una cifra con un tamaño de muestra de 10, de dos dígitos con un tamaño de muestra entre 100 y 1000, de tres dígitos con un tamaño de muestra de entre 10,000 y 100,000, de cuatro dígitos con un tamaño de muestra de 1,000,000.

Se puede concluir que el método de Monte-Carlo está basado en que a mayor cantidad de tamaño de muestra o puntos generados, se incrementa el acercamiento con el valor real. Se observó un comportamiento parecido al presentado a la aproximación del área de la integral. Debido a ésto en la práctica el tamaño de muestra está determinado por la aplicación del valor o dato que se busca obtener.

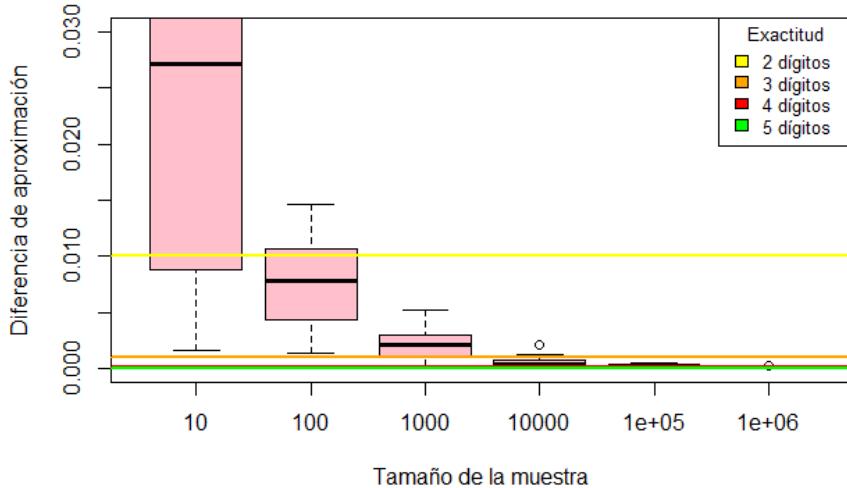


Figura 3: Gráfica de la relación de tamaño de muestra y aproximación de Monte-Carlo al valor  $\frac{\pi}{4}$

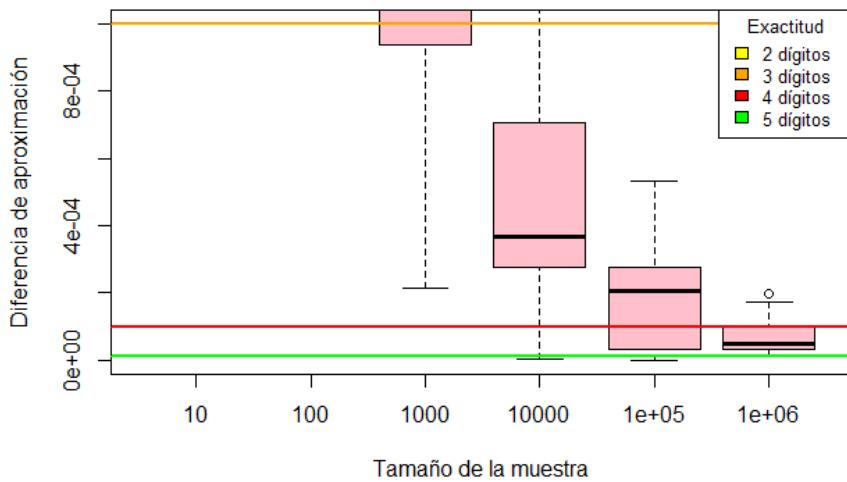


Figura 4: Acercamiento Gráfica de la relación de tamaño de muestra y aproximación de Monte-Carlo al valor  $\frac{\pi}{4}$

## Referencias

- [1] Ángel Franco García. Los métodos montecarlo. Página Web, 2009. URL [http://www.sc.ehu.es/sbweb/fisica\\_numerico/montecarlo/montecarlo.html](http://www.sc.ehu.es/sbweb/fisica_numerico/montecarlo/montecarlo.html)
- [2] Maria Grazia Pia and Georg Weidenspointner. Monte Carlo Simulation for Particle Detectors. Technical Report arXiv:1208.0047, Agosto 2012. URL <http://cds.cern.ch/record/1471335>. CERN Council Open Symposium on European Strategy for Particle Physics, 10–12 September 2012, Krakow, Poland.

- [3] Elisa Schaeffer. Práctica 5: Método monte-carlo. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p5.html>.

# Práctica 6

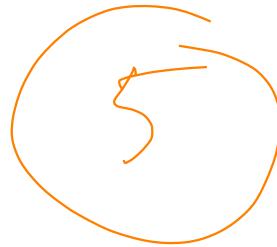
## Sistema Multiagente

### Revisión y Retroalimentación

- Se revisó la ortografía
- Se usaron comandos aritméticos para la fórmula LxL
- Se arreglaron problemas con la bibliografía, se tiene que corroborar que el archivo tenga extensión .bib
- Se incluyó la observación del mínimo de probabilidad de vacunación necesaria para tener una población sana.

# Sistema Multiagente

3175



5 de marzo de 2019

## 1. Introducción

Un sistema multiagente es comúnmente definido como un sistema inmerso en un entorno, que es capaz de percibirlo y actuar sobre él, siguiendo una agenda propia con el fin de modificarlo y que es capaz de comunicarse con otros agentes. Los agentes cuentan con ciertas características: Puede sentir ya sea detectando eventos o por medio de sensores (condiciones). Es reactivo, ya siente mensajes o eventos reacciona en base a éstos, de forma que monitorea activamente el estado del entorno. Es autónomo, funciona sin intervención directa de seres humanos u otros agentes, con control de sus acciones y su estado interno[?].

## 2. Objetivo

Estudiar el efecto estadístico de una probabilidad de vacunación inicial en un sistema multiagente que emula un sistema de contagio donde los agentes pueden ser: susceptibles, infectados o recuperados.

## 3. Metodología

\$ \ell l H' m e s \ell l l \\$

Se comienza con el código proporcionado[?], suponiendo una cantidad de 50 agentes para la simulación dentro de un área en forma de torus la cual es continua de tamaño  $lxl$ , donde los agentes que se mueven en determinada dirección y cierta velocidad tienen una probabilidad de infectarse dada por la distancia euclídea que existe entre dos agentes usando además un umbral de 0.1. Se considera que solamente los susceptibles pueden infectarse y los recuperados con cierta resistencia desarrollada ya no se pueden infectar.

Para el estudio se van a vacunar al principio determinados agentes con cierta probabilidad, la cual va a variar de 0 a 1 en pasos de 0.1, después de ésto se va a infectar aleatoriamente con una probabilidad de infección inicial de 0.05 para finalmente correr las simulaciones con 40 repeticiones y un número máximo de pasos de 100:

```
1 for (pv in PV){ #variando la probabilidad de vacunacion
2   for (rep in 1:40){ #con 40 repeticiones
3     agentes <- data.frame(x = double(), y = double(), dx = double(), dy = double(),
4                           estado = character())
5     for (i in 1:n) {
6       if (runif(1) < pv){
7         e <- "R"
8       } else if (runif(1) < pi){
9         e <- "I"
10      } else{
11        e <- "S"
12      }
```

```

13 agentes <- rbind(agentes , data.frame(x = runif(1, 0, 1), y = runif(1, 0, 1),
14                                         dx = runif(1, -v, v), dy = runif(1, -v, v),
15                                         estado = e))
16
17
18 levels(agentes$estado) <- c("S", "I", "R") #se determinan los posibles estados
19 }
20
21 epidemia <- integer()

```

Se desea conocer el porcentaje máximo de infectados por simulación, para poder observar el fenómeno que tiene el hecho de haber vacunado con cierta probabilidad dentro del sistema multiagente de contagios planteado.

¿ % max?

## 4. Resultados

Los resultados en la figura 1 muestran los porcentajes máximos de infectados presentados en una gráfica de caja bigote, por lo que podemos observar los promedios de las cuarenta repeticiones por cada grupo de probabilidad de ser vacunado.

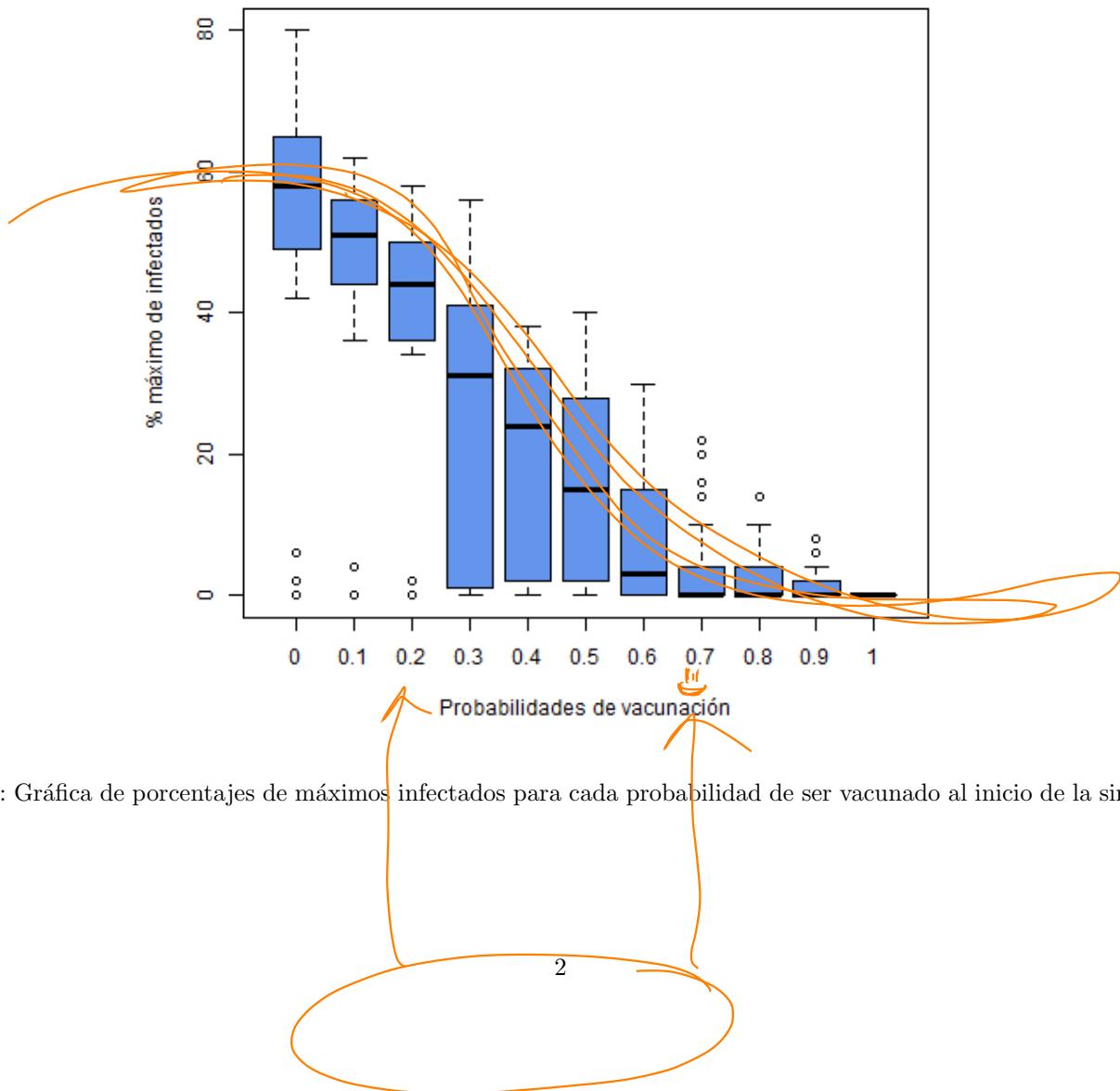
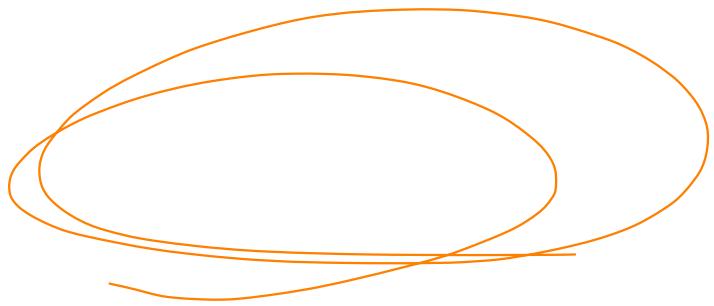


Figura 1: Gráfica de porcentajes de máximos infectados para cada probabilidad de ser vacunado al inicio de la simulación

## 5. Conclusiones

Con base en los resultados se puede concluir que la probabilidad de ser vacunado está relacionada con la cantidad de máximos infectados entre los agentes que se presentarán en cada caso, teniendo una tendencia inversamente proporcional.

## Referencias



# Práctica 6: Sistema Multiagente

3175

5 de Marzo de 2019

## 1. Introducción

Un sistema multiagente es comunmente definido como un sistema inmerso en un entorno, que es capaz de percibirlo y actuar sobre él, siguiendo una agenda propia con el fin de modificarlo y que es capaz de comunicarse con otros agentes [1].

## 2. Objetivo

Estudiar el efecto estadístico de una probabilidad de vacunación inicial en un sistema multiagente que emula un sistema de contagio donde los agentes pueden ser: susceptibles, infectados o recuperados.

## 3. Metodología

Se comienza con el código proporcionado [2], suponiendo una cantidad de 50 agentes para la simulación dentro de un área en forma de torus la cual es continua de tamaño  $\mathcal{L} \times \mathcal{L}$ , donde los agentes que se mueven en determinada dirección y cierta velocidad tienen una probabilidad de infectarse dada por la distancia euclíadiana que existe entre dos agentes usando además un umbral de 0.1. Se considera que solamente los susceptibles pueden infectarse y los recuperados con cierta resistencia desarrollada ya no se pueden infectar.

Para el estudio se van a vacunar al principio determinados agentes con cierta probabilidad  $PV$ , la cual va a variar de 0 a 1 en pasos de 0.1, después de esto se va a infectar aleatoriamente con una probabilidad de infección inicial de 0.05 para finalmente correr las simulaciones con 40 repeticiones y un número máximo de pasos de 100:

```
1 for(pv in PV){ #variando la probabilidad de vacunacion
2   for(rep in 1:40){ #con 40 repeticiones
3     agentes <- data.frame(x = double() , y = double() , dx = double() , dy = double() ,
4                           estado = character())
5     for (i in 1:n) {
6       if(runif(1) < pv){
7         e <- "R"
8       } else if(runif(1) < pi){
9         e <- "I"
10      } else{
11        e <- "S"
12      }
13      agentes <- rbind(agentes , data.frame(x = runif(1 , 0 , 1) , y = runif(1 , 0 , 1) ,
14                                         dx = runif(1 , -v , v) , dy = runif(1 , -v , v) ,
15                                         estado = e))
16      levels(agentes$estado) <- c("S" , "I" , "R") #se determinan los posibles estados
```

```

17 }
18     epidemia <- integer()
19     for (tiempo in 1:tmax){
20         infectados <- dim(agentes[agentes$estado == "I",])[1]
21         epidemia <- c(epidemia, infectados)
22         if (infectados == 0) {
23             break

```

Se desea conocer el porcentaje máximo de infectados por simulación, para poder observar el fenómeno que tiene el hecho de haber vacunado con cierta posibilidad dentro del sistema multiagente de contagios planteado.

## 4. Resultados

Los resultados en la figura 1 muestran los porcentajes máximos de infectados presentados en una gráfica de caja bigote, por lo que podemos observar los promedios de las cuarenta repeticiones por cada grupo de probabilidad de ser vacunado. Se puede observar como se colapsan los promedios del porcentaje de infectados a partir de tener una probabilidad de vacunación del 70 %.

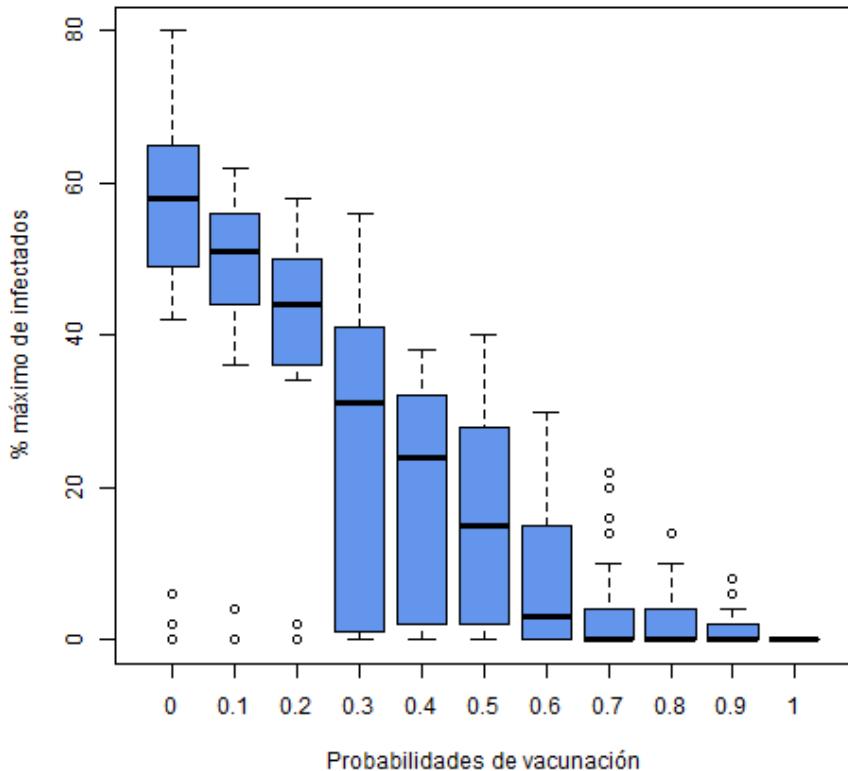


Figura 1: Gráfica de porcentajes de máximos infectados para cada probabilidad de ser vacunado al inicio de la simulación

## 5. Conclusiones

Con base en los resultados se puede concluir que la probabilidad de ser vacunado está relacionada con la cantidad de máximos infectados entre los agentes que se presentarán en cada caso, teniendo una tendencia inversamente proporcional. Se puede concluir en base a los resultados de la gráfica que para mantener una población sana al menos se tiene que vacunar el 70 % de la población.

## Referencias

- [1] Universidad de Sevilla Departamento de Ciencias de la Computación e Inteligencia Artificial. Sistemas multiagente. Página Web, 2018. URL <http://www.cs.us.es/~fsancho/ficheros/ICSR/320Sistemas20Multiagente.pdf>.
- [2] Elisa Schaeffer. Práctica 5: Sistema multiagente. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p6.html>.

# Práctica 7

## Búsqueda Local

### **Revisión y Retroalimentación**

- Se revisó la ortografía y se corrigieron detalles de espaciado y del título.
- Se empleó el código “\times” para reemplazar los asteriscos en las ecuaciones.
- La redacción se debe de hacer de manera impersonal.

# Búsqueda Local

3175

19 de marzo de 2019



## 1. Introducción

Una búsqueda local consiste en buscar determinados valores (ya sean máximos o mínimos) a lo largo de una función tomando en cuenta vecinos en un punto para posteriormente determinar hacia que valor nos queremos desplazar sucesivamente. En la práctica se usaron ejemplos basados en Womersley[2].

Los pasos

## 2. Objetivo

Observar el comportamiento de una búsqueda local de máximos en una función determinada.

## 3. Metodología

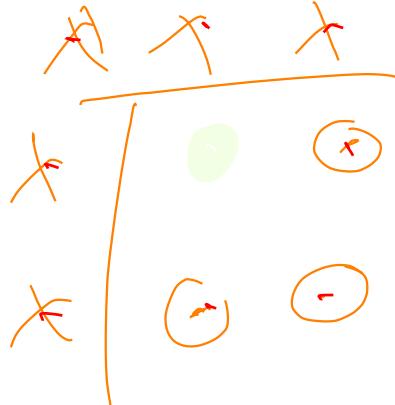
Usando de base el código proporcionado[1], se busca maximizar es decir, buscar el máximo la función bidimensional  $(x + 0,5)^4 - 30x^2 - 20x + (y + 0,5)^4 - 30y^2 - 20y)/100$  dentro de un plano de tamaño delimitado por los valores de -3 a 3 donde tomamos en cuenta el valor máximo de  $z$  que está en función de las coordenadas de  $x$  y  $y$ .

Se realizaron 100 pasos de un tamaño de 0.25, esto quiere decir que se tomaron en cuenta márgenes de 0.25 arriba y abajo de cada punto para moverse a una nueva posición con mayor valor de  $z$ . Además se realizaron 15 repeticiones de manera simultánea y se fijaron las coordenadas dentro de un **data.frame** para su posterior uso y visualización.

```
1 low <- -3
2 high <- 3
3 step <- 0.25
4 replicas <- 15
5 coordinates <- data.frame("X"=0, "Y"=0, "T"=0, "R" = 0)
```

Para poder buscar un máximo en una función bidimensional se necesita buscar en lugares o vecinos, mediante el cual se utilizó el siguiente criterio:

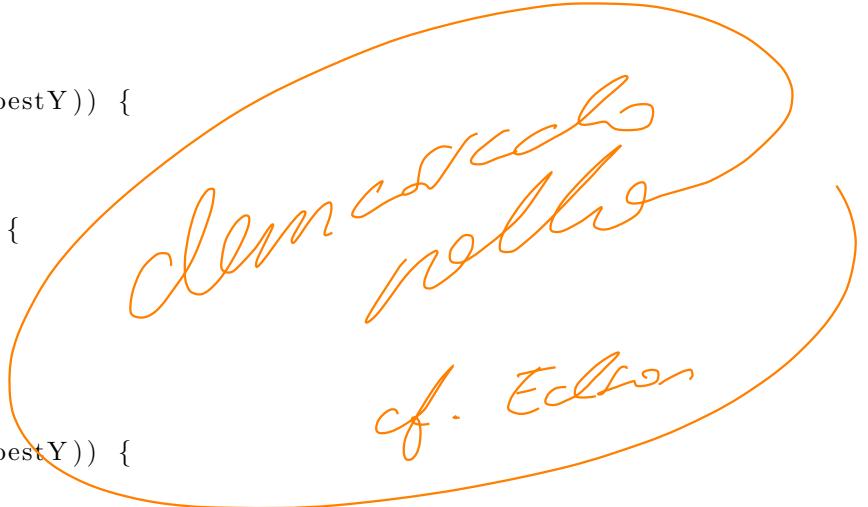
```
1 if (g(left , currY) > g(right , currY)) {
2   currX <- left
3 } else {
4   currX <- right
5 }
6 if (g(currX , currY) > g(bestX , bestY)) {
7   bestX <- currX
8   bestY <- currY
9 }
10 if (g( currX , down) > g( currX , up)) {
```



```

11     currY <- down
12 } else {
13     currY <- up
14 }
15 if (g(currX, currY) > g(bestX, bestY)) {
16     bestX <- currX
17     bestY <- currY
18 }
19 if (g(left, up) > g(right, up)) {
20     currX <- left
21     currY <- up
22 } else {
23     currX <- right
24     currY <- up
25 }
26 if (g(currX, currY) > g(bestX, bestY)) {
27     bestX <- currX
28     bestY <- currY
29 }
30 if (g(left, down) > g(right, down)) {
31     currX <- left
32     currY <- down
33 } else {
34     currX <- right
35     currY <- up
36 }
37 if (g(currX, currY) > g(bestX, bestY)) {
38     bestX <- currX
39     bestY <- currY

```



Finalmente se realizaron gráficas de los resultados obtenidos, marcando con puntos de colores las nuevas posiciones de las 15 repeticiones buscando el valor máximo de  $z$ .

## 4. Resultados

Los resultados mostrados en las figuras 1 a 3 muestran una vista superior de la  $f(x, y)$  durante el inicio, a la mitad y al final de la prueba respectivamente y se puede ver un comportamiento de movimiento inclinado a aproximarse a la zona más oscura de la gráfica, que representa los valores más altos de la función. Mediante una imagen .gif que se encuentra en la carpeta del repositorio de la práctica se puede apreciar mejor los resultados.

## 5. Conclusiones

Se puede concluir que la búsqueda local es una muy importante y efectiva herramienta para buscar valores en una función comparable con un conjunto de datos que pueden ser en una o varias dimensiones.

## Referencias

- [1] Elisa Schaeffer. Práctica 7: Búsqueda local. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p7.html>.

Paso 1

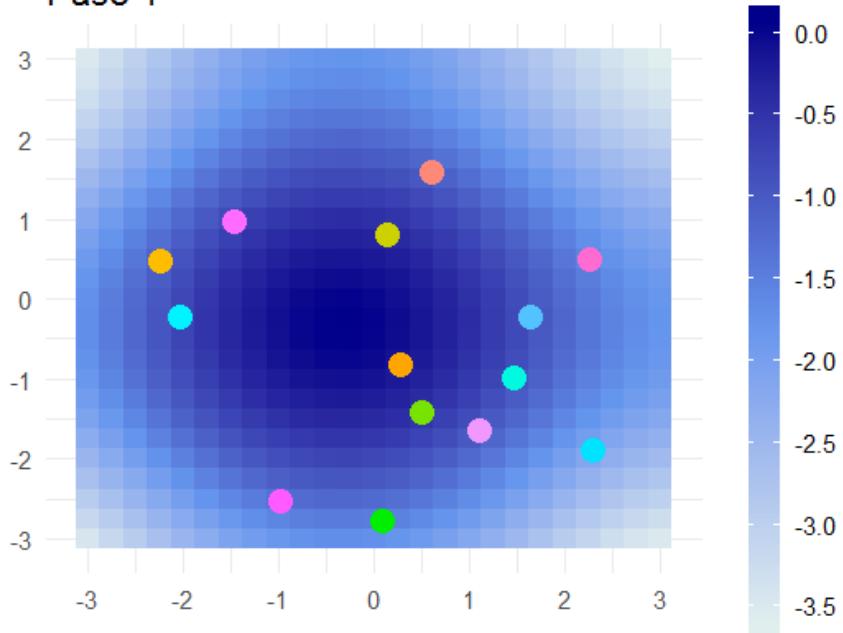


Figura 1: Posición al inicio

Paso 50

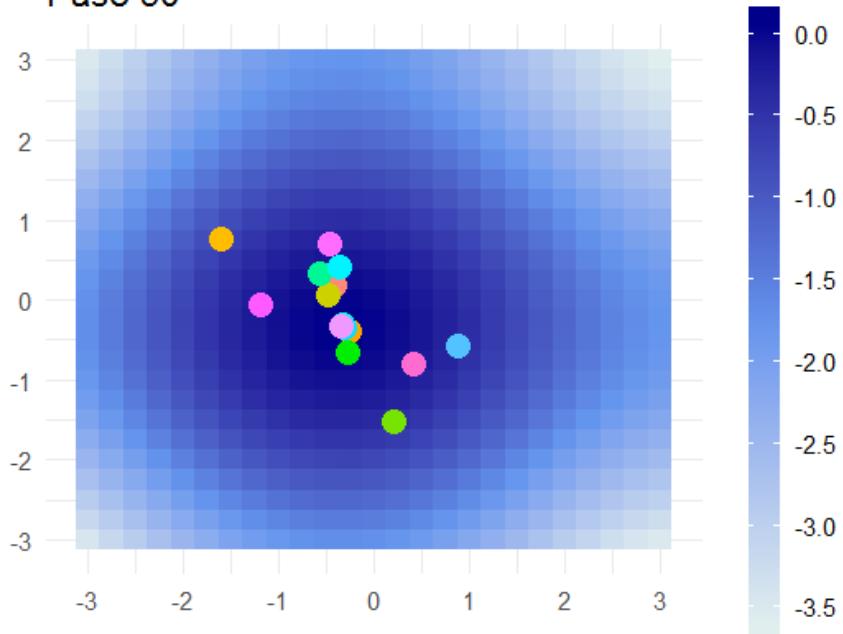


Figura 2: Posición a la mitad, en el paso 50

- [2] Rob Womersley. Local and global optimization. Página Web, 2018. URL <https://web.maths.unsw.edu.au/~rsw/lgopt.pdf>.

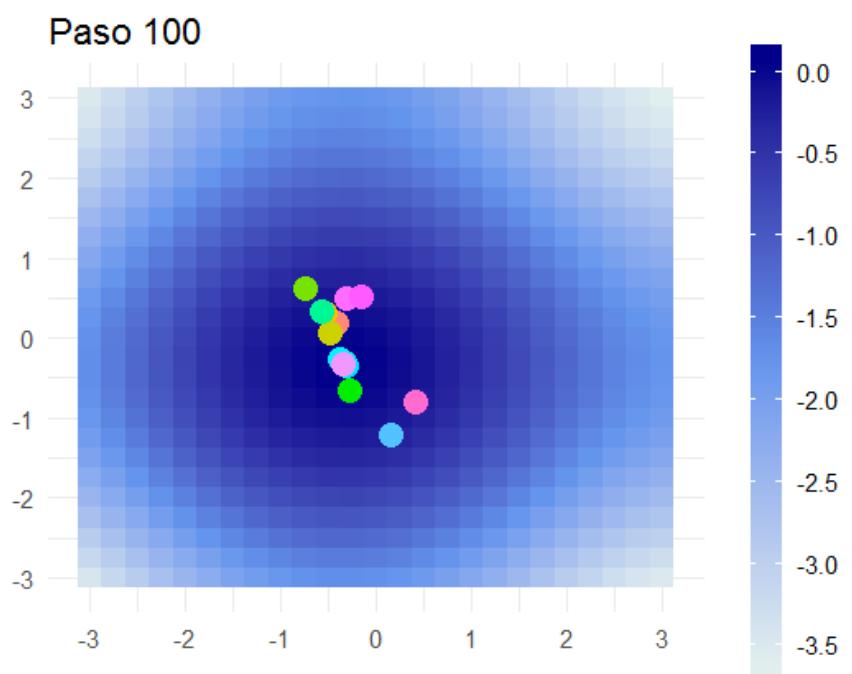


Figura 3: Posición final

# Práctica 7: Búsqueda Local

3175

19 de Marzo de 2019

## 1. Introducción

Una búsqueda local consiste en buscar determinados valores (ya sean máximos o mínimos) a lo largo de una función tomando en cuenta vecinos en un punto para posteriormente determinar hacia que valor se quiere desplazar sucesivamente. En la práctica se usaron ejemplos basados en Womersley [2].

## 2. Objetivo

Observar el comportamiento de una búsqueda local de máximos en una función determinada.

## 3. Metodología

Usando de base el código proporcionado [1], se busca maximizar es decir, buscar el máximo la función bidimensional  $(x + 0,5)^4 - 30 \times x^2 - 20 \times x + (y + 0,5)^4 - 30 \times y^2 - 20 \times y)/100$  dentro de un plano de tamaño delimitado por los valores de  $-3$  a  $3$  donde tomamos en cuenta el valor máximo de  $z$  que está en función de las coordenadas de  $x$  y  $y$ .

Se realizaron 100 pasos de un tamaño de  $0.25$ , esto quiere decir que se tomaron en cuenta márgenes de  $0.25$  arriba y abajo de cada punto para moverse a una nueva posición con mayor valor de  $z$ . Además se realizaron 15 repeticiones de manera simultánea y se fijaron las coordenadas dentro de un *dataframe* para su posterior uso y visualización.

```
1 low <- -3
2 high <- 3
3 paso <- 0.25
4 replicas <- 15
5 coordenadas <- data.frame("X"=0, "Y"=0, "T"=0, "R" = 0)
```

Para poder buscar un máximo en una función bidimensional se necesita buscar en ocho lugares o vecinos, mediante el cual se utilizó el siguiente criterio:

```
1 if (g(left , currY) > g(right , currY)) {
2   currX <- left
3 } else {
4   currX <- right
5 }
6 if (g(currX , currY) > g(bestX , bestY)) {
7   bestX <- currX
8   bestY <- currY
9 }
10 if (g(currX , down) > g(currX , up)) {
```

```

11     currY <- down
12 } else {
13     currY <- up
14 }
15 if (g(currX, currY) > g(bestX, bestY)) {
16     bestX <- currX
17     bestY <- currY
18 }
19 if (g(left, up) > g(right, up)) {
20     currX <- left
21     currY <- up
22 } else {
23     currX <- right
24     currY <- up
25 }
26 if (g(currX, currY) > g(bestX, bestY)) {
27     bestX <- currX
28     bestY <- currY
29 }
30 if (g(left, down) > g(right, down)) {
31     currX <- left
32     currY <- down
33 } else {
34     currX <- right
35     currY <- up
36 }
37 if (g(currX, currY) > g(bestX, bestY)) {
38     bestX <- currX
39     bestY <- currY

```

Finalmente se realizaron gráficas de los resultados obtenidos, marcando con puntos de colores las nuevas posiciones de las 15 repeticiones buscando el valor máximo de  $z$ .

## 4. Resultados

Los resultados mostrados en las figuras 1 a 3 muestran una vista superior de la  $f(x, y)$  durante el inicio, a la mitad y al final de la prueba respectivamente y se puede ver un comportamiento de movimiento inclinado a aproximarse a la zona más oscura de la gráfica, que representa los valores más altos de la función. Mediante una imagen .gif que se encuentra en la carpeta del repositorio de la práctica se puede apreciar mejor los resultados.

## 5. Conclusiones

Se puede concluir que la búsqueda local es una muy importante y efectivo método para buscar valores en una función comparable con un conjunto de datos que pueden ser en una o varias dimensiones.

## Referencias

- [1] Elisa Schaeffer. Práctica 7: Búsqueda local. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p7.html>.

Paso 1

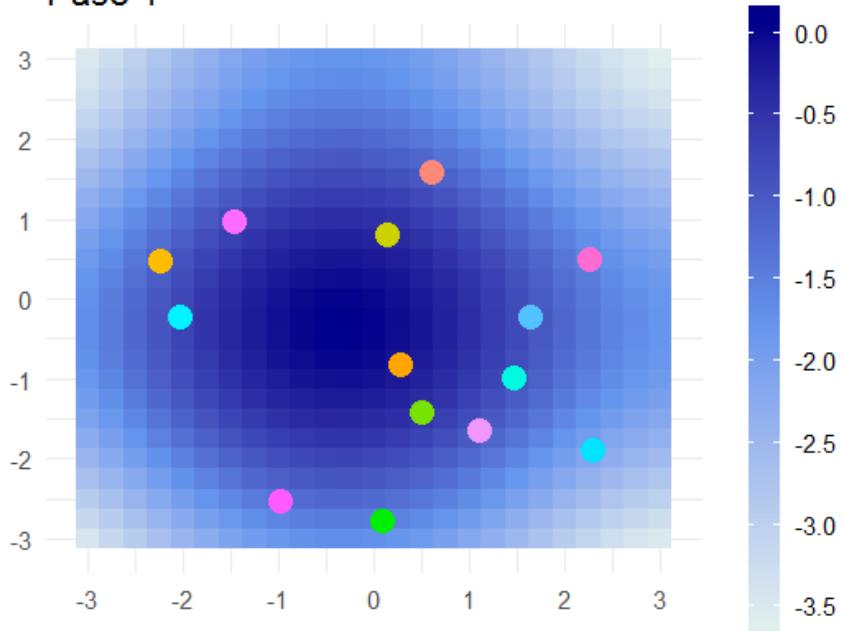


Figura 1: Posición al inicio

Paso 50

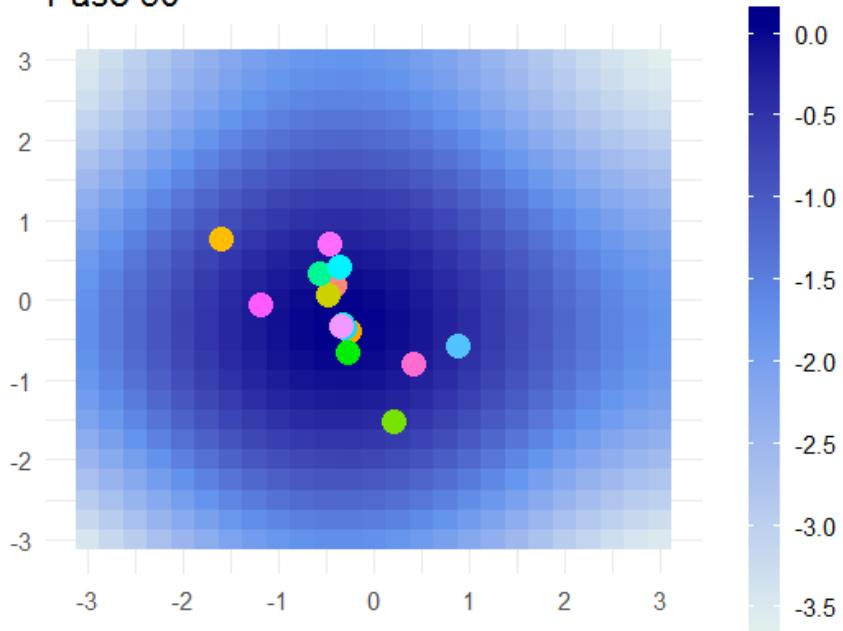


Figura 2: Posición a la mitad, en el paso 50

- [2] Rob Womersley. Local an global optimization. Página Web, 2018. URL <https://web.maths.unsw.edu.au/~rsw/lgopt.pdf>.

Paso 100

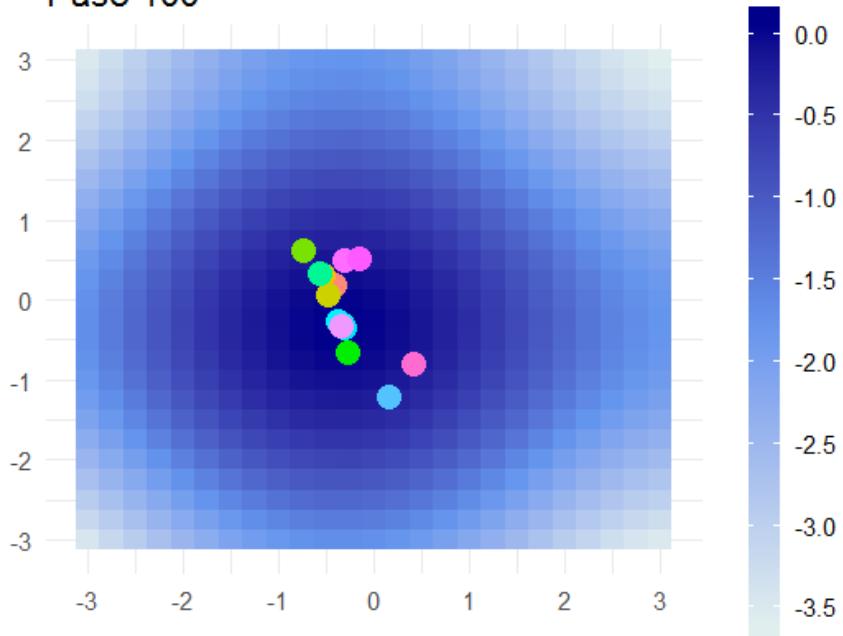


Figura 3: Posición final

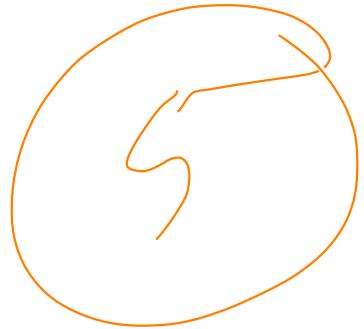
# Práctica 8

## Modelo de Urnas

### **Revisión y Retroalimentación**

- Se revisó la ortografía.
- Se arreglaron los problemas con la bibliografía, en ocasiones el TeXMaker no agrega las bibliografías aunque éstas estén bien colocadas la primera vez que se compila, debo poner más atención de guardar siempre con la extensión “.bib” el archivo de la bibliografía.
- Se borró parte del código que resultaba irrelevante
- Se incluyó la explicación de la prueba estadística en la conclusión, no debo incluirla pues si no se especifica no se ve fundamentada en la prueba.

# Práctica 8: Modelo de Urnas



3175

26 de marzo de 2019

## 1. Introducción

La práctica de urnas tratará de simular efectos de coalescencia y fragmentación de cúmulos, parecido a el fenómeno de flocculación, usado en tratamientos fisicoquímicos de sustancias [? ].

## 2. Objetivo

En base al modelo de urnas, simular un modelo de coalescencia y fragmentación de partículas y comparar el tiempo entre llevar un proceso paralelizado o uno secuencial.

## 3. Metodología

Usando de base el código proporcionado [? ], que usa las funciones de union y separación de partículas  $n$ , partiendo de una distribución normal de frecuencias de tamaño de cúmulo  $k$ .

```
1 k <- cumu
2 n <- part
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)
8 if (diferencia > 0) {
9   for (i in 1:diferencia) {
10     p <- sample(1:k, 1)
11     cumulos[p] <- cumulos[p] + 1
12   }
13 } else if (diferencia < 0) {
14   for (i in 1:-diferencia) {
15     p <- sample(1:k, 1)
16     if (cumulos[p] > 1) {
17       cumulos[p] <- cumulos[p] - 1
18     }
19   }
20 }
```

Posteriormente se realiza la función de unión o rompimiento dependiendo del tamaño del cúmulo, donde se toma como criterio que el tamaño crítico el cual se establece en la mediana de los cúmulos, cúmulos mayores al tamaño crítico tienden a separarse en ~~2~~ partes de tamaños aleatorios diferentes de cero y menores solo pueden unirse

```
1 c <- median(cumulos) # tamaño critico de cumulos
```

```

2 d <- sd(cumulos) / 4 # factor arbitrario para suavizar la curva
3 rotura <- function(x) {
4   return (1 / (1 + exp((c - x) / d)))
5 }
6 union <- function(x) {
7   return (exp(-x / c))
8 }
9 romperse <- function(tam, cuantos) {
10   romper <- round(rotura(tam) * cuantos) # independientes
11   resultado <- rep(tam, cuantos - romper) # los demás
12   if (romper > 0) {
13     for (cumulo in 1:romper) { # agregar las rotas
14       t <- 1
15       if (tam > 2) { # sample no jala con un solo valor
16         t <- sample(1:(tam-1), 1)
17       }
18       resultado <- c(resultado, t, tam - t)
19     }
20   }
21   assert(sum(resultado) == tam * cuantos) # no hubo perdidas
22   return(resultado)
23 }
24 unirse <- function(tam, cuantos) {
25   unir <- round(union(tam) * cuantos) # independientes
26   if (unir > 0) {
27     division <- c(rep(-tam, unir), rep(tam, cuantos - unir))
28     assert(sum(abs(division)) == tam * cuantos)
29     return(division)
30   } else {
31     return(rep(tam, cuantos))

```

Finalmente se determina la duración de este proceso en 50 pasos, se realizan 5 repeticiones y se varían los valores de  $k$  y  $n$  para correr el proceso de manera secuencial y paralelizada tomando los tiempos que toma, de esta manera comparar si se ahorra tiempo en un tratamiento u otro mediante una prueba estadística.

```

1 replicas <- 5
2 tiemposSP <- c()
3 N <- seq(30000,45000,5000)
4 K <- seq(300,450,50)
5 for(part in N) {
6   for(cumu in K) {
7     for(rep in 1:replicas) {
8       tsec <- data.matrix(system.time(urnas(cumu, part))[1])[1]
9       tiemposSEC <- c(tiemposSEC, tsec)
10    }
11  }
12 }
13 suppressMessages(library(doParallel))
14 cluster <- makeCluster(detectCores() - 3)
15 registerDoParallel(cluster)
16 tiemposCP <- c()
17 for(part in N) {
18   for(cumu in K) {
19     tcp <- foreach(r= 1:replicas , .combine = c) %dopar% data.matrix(system.time(urnas(cumu,

```

```

20     tiemposCP <- c(tiemposP , tp) #se guarda tiempo
21     stopImplicitCluster()

```

## 4. Resultados

Los resultados mostrados en las figuras 1 y 2 son los resultados de los tiempos a diferentes valores de  $k$  y  $n$  secuencial y paralelizadamente respectivamente. Se puede observar valores cercanos entre la manera paralelizada y la secuencial, sin embargo es notorio que de manera secuencial los valores presentaron menos variaciones entre repeticiones que los de manera secuencial. La prueba estadística da como resultado que los resultados a éstas repeticiones no son muy diferentes entre uno y otro tratamiento.

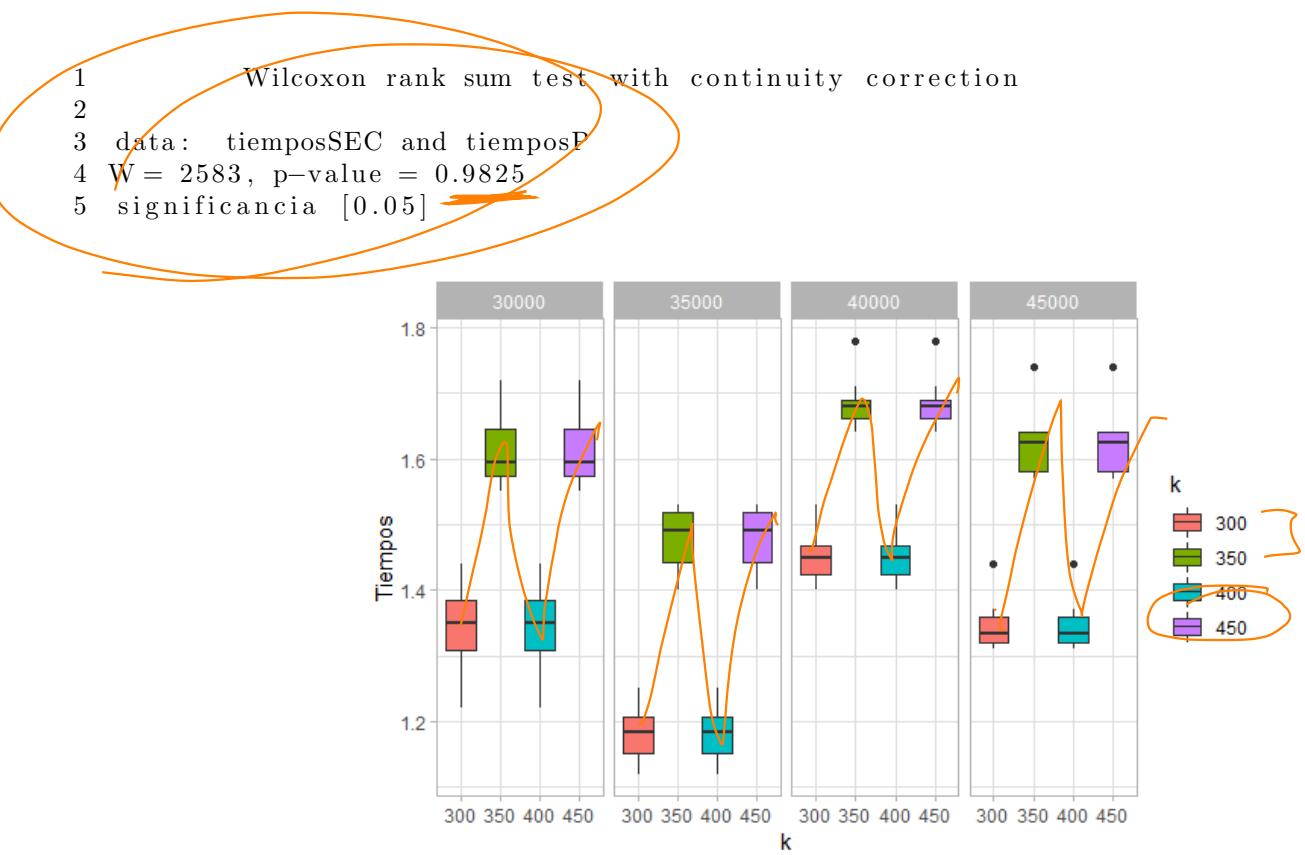


Figura 1: Tiempos en el proceso de manera secuencial

## 5. Conclusiones

Se concluye en base a los resultados que la diferencia de tratamientos no influye de manera significativa en los tiempos de procesamiento a este número de repeticiones. Sería importante verificar posteriormente si esto cambia a diferente número de repeticiones.

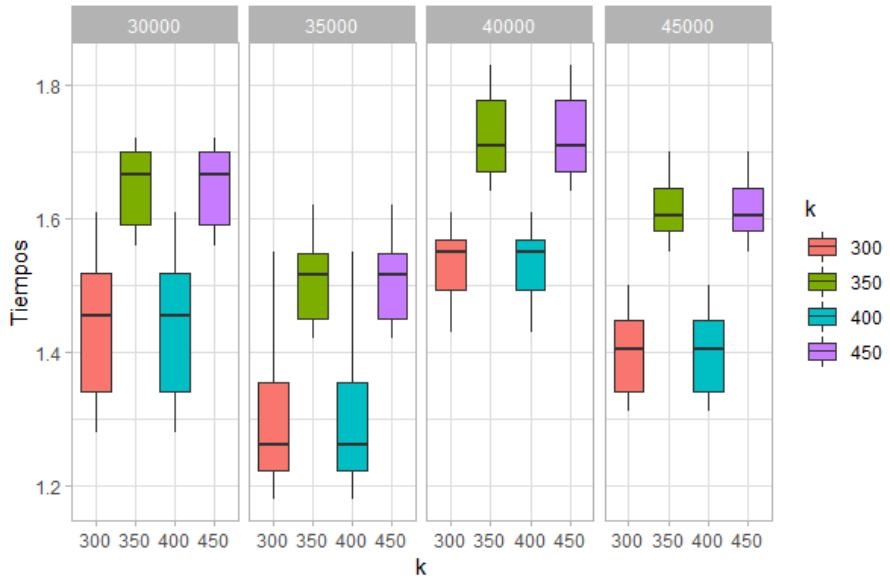


Figura 2: Tiempos en el proceso de manera paralelizada

## Referencias

- María Gonzales Gonzalo. Aplicación de procesos de coagulación floculación en la regeneración de aguas depuradas. Universidad de Zaragoza, 2010. URL <https://zaguan.unizar.es/record/4935?ln=es>.
- Elisa Schaeffer. Práctica 8: Modelo de urnas. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p8.html>.

# Práctica 8: Modelo de Urnas

3175

26 de marzo de 2019

## 1. Introducción

La práctica de urnas trata de simular efectos de coalescencia y fragmentación de cúmulos, parecido a el fenómeno de flocculación, usado en tratamientos fisicoquímicos de sustancias [1].

## 2. Objetivo

En base al modelo de urnas, simular un modelo de coalescencia y fragmentación de partículas y comparar el tiempo entre llevar un proceso paralelizado o uno secuencial.

## 3. Metodología

Usando de base el código proporcionado [2], que usa las funciones de union y separación de partículas  $n$ , partiendo de una distribución normal de frecuencias de tamaño de cúmulo  $k$ .

```
1 k <- cumu
2 n <- part
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)
8 if (diferencia > 0) {
9   for (i in 1:diferencia) {
10     p <- sample(1:k, 1)
11     cumulos[p] <- cumulos[p] + 1
12   }
13 } else if (diferencia < 0) {
14   for (i in 1:-diferencia) {
15     p <- sample(1:k, 1)
16     if (cumulos[p] > 1) {
17       cumulos[p] <- cumulos[p] - 1
```

Posteriormente se realiza la función de unión o rompimiento dependiendo del tamaño del cúmulo, donde se toma como criterio que el tamaño crítico el cual se establece en la mediana de los cúmulos, cumulos mayores al tamaño critico tienden a separarse en dos partes de tamaños aleatorios diferentes de cero y menores solo pueden unirse.

Finalmente se determina la duración de este proceso en 50 pasos, se realizan 5 repeticiones y se varían los valores de  $k$  y  $n$  para correr el proceso de manera secuencial y paralelizada tomando los tiempos que toma, de esta manera comparar si se ahorra tiempo en un tratamiento u otro mediante una prueba estadística.

```
1 suppressMessages(library(doParallel))
2 cluster <- makeCluster(detectCores() - 3)
3 registerDoParallel(cluster)
4 tiemposCP <- c()
5 for(part in N) {
6   for(cumu in K) {
7     tp <- foreach(r= 1:replicas , .combine = c) %dopar%
8       data.matrix(system.time(urnas(cumu, part))[1])[1] #Paralelismo
9     tiemposP <- c(tiemposP, tp)
10   stopImplicitCluster()
11 }
```

## 4. Resultados

Los resultados mostrados en las figuras 1 y 2 son los resultados de los tiempos a diferentes valores de  $k$  y  $n$  secuencial y paralelizadamente respectivamente. Se puede observar valores cercanos entre la manera paralelizada y la secuencial, sin embargo, es notorio que de manera secuencial los valores presentaron menos variaciones entre repeticiones que los de manera secuencial. La prueba estadística da como resultado que los resultados a estas repeticiones no son muy diferentes entre uno y otro tratamiento.

```
1 Wilcoxon rank sum test with continuity correction
2
3 data: tiemposSEC and tiemposP
4 W = 2583, p-value = 0.9825
5 significancia [0.05]
```

## 5. Conclusiones

Se concluye en base a los resultados que la diferencia de tratamientos no influye de manera significativa en los tiempos de procesamiento a este número de repeticiones, dado que en la prueba estadística el valor-p es 0.9825 aceptándose así la hipótesis nula. Sería importante verificar posteriormente si esto cambia a diferente número de repeticiones.

## Referencias

- [1] María Gonzales Gonzalo. Aplicación de procesos de coagulación floculación en la regeneración de aguas depuradas. Universidad de Zaragoza, 2010. URL <https://zaguan.unizar.es/record/4935?ln=es>.
- [2] Elisa Schaeffer. Práctica 8: Modelo de urnas. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p8.html>.

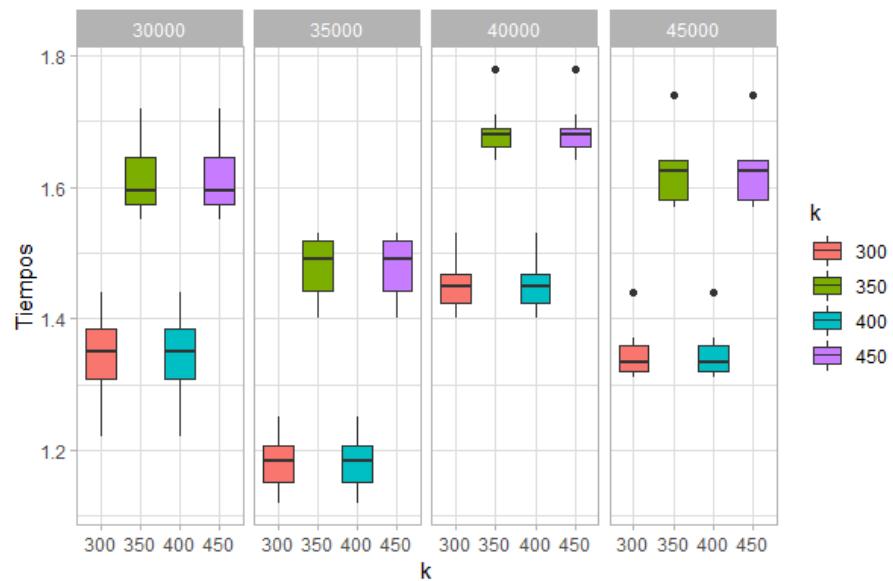


Figura 1: Tiempos en el proceso de manera secuencial

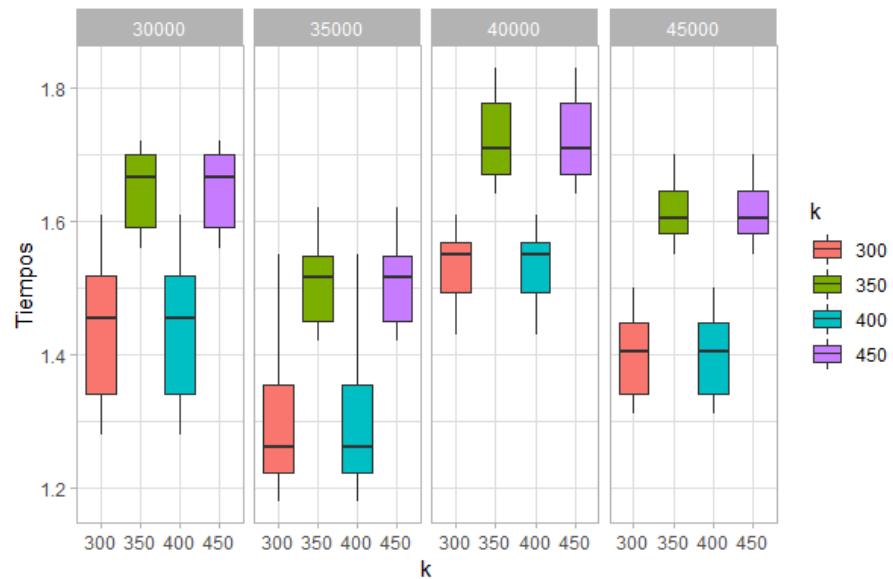


Figura 2: Tiempos en el proceso de manera paralelizada

# Práctica 9

## Interacciones entre partículas

### **Revisión y Retroalimentación**

- Se revisó ortografía, no llevan acento las palabras “éstos, éstas” si tienen el sujeto especificado y se añadieron acentos.
- Se añadió una cita bibliográfica faltante.

# Práctica 9: Interacciones entre partículas

3175



2 de abril de 2019

## 1. Introducción

Las fuerzas entre partículas son muchas y variadas, entre las mas importantes están las fuerzas electroestáticas proporcionadas por las cargas que éstas poseen la cual determina si se atraen o se repelen y la fuerza de gravedad que afecta a todo aquello que tiene una masa o inclusive una masa aparente como lo es en el caso de algunas partículas como los electrones[1].

## 2. Objetivo

En base a un modelo de interacción entre elementos determinado por una simulación simple de cargas, agregar un factor adicional de un fenómeno como es la gravedad también de manera simple para comparar como estos dos fenómenos influyen en el comportamiento de los elementos dentro del sistema tomando como referencia la velocidad de éstos.

## 3. Metodología

Usando de base el código proporcionado[2], se definen cincuenta elementos a los que consideramos partículas  $n$  cada una con una posición  $x$  y  $y$ , carga  $c$  (figura 1) y posteriormente se agrega un valor de masa  $m$  definido en base a valores dentro de una distribución normal en valores positivos de cero a uno, con excepción de las cargas donde éstos valores pueden ser negativos entre menos uno y uno, el signo del valor de la carga describe su carga, la cual es necesaria para determinar si se atraen o se repelen dos partículas.

Dentro de la función de fuerza se deben incluir los criterios que definen el comportamiento por derivado de la masa, el cual es la fuerza de gravedad tomando en cuenta cual partícula tiene mas masa que es la que va a atraer a la otra definiendo la dirección del movimiento de la fuerza, para finalmente aplicarse a las fuerzas totales que inciden. Se realiza esto para las fuerzas totales, para solo las cargas (como en el código base) y solamente tomando en cuenta la masa.

```
1 dirm <- (-1)^(1 + 1 * (mi < mj))  
1 factorm <- dirm * mi*mj / (dx^2 + dy^2) * 6.5  
1 fxm <- (fxm - dx * factorm)
```

Finalmente se realizan las iteraciones para las cincuenta partículas en un lapso de tiempo de cien pasos tomando en cuenta los tres criterios antes mencionados. Definendo los cambios *delta* de posición diferentes para cada caso, en cada paso que es la unidad de tiempo.

```
1 suppressMessages(library(doParallel))  
2 registerDoParallel(makeCluster(detectCores() - 1))  
3 tmax <- 100
```

```

4  deltat <- c()
5  deltam <- c()
6  deltac <- c()
7  for (iter in 1:tmax) {
8    fm <- foreach(i = 1:n, .combine=c) %dopar % fuerzam(i)
9    fc <- foreach(i = 1:n, .combine=c) %dopar % fuerzac(i)
10   f <- foreach(i = 1:n, .combine=c) %dopar % fuerza(i)
11   delta <- 0.02 / max(abs(f))
12   deltam <- 0.02 / max(abs(fm))
13   deltac <- 0.02 / max(abs(fc))
14   deltat <- c(deltat, delta)
15   deltamc <- c(deltamc, deltam)
16   deltacc <- c(deltacc, deltac)

```

## 4. Resultados

Los resultados en la figura 2 muestran el comportamiento del sistema de partículas representado por líneas que describen el comportamiento de la velocidad, en el cual se aprecia la velocidad con cargas en color azul, la velocidad con masas en color negro y la combinación de ambas en línea amarilla.

## 5. Conclusiones

Se puede determinar en base a la gráfica que el comportamiento de la velocidad de las partículas con ambos fenómenos se ve mas fuertemente influenciado por la fuerza de gravedad que por las cargas, que sin embargo, sí modifica visiblemente el comportamiento de la velocidad pero en menor medida

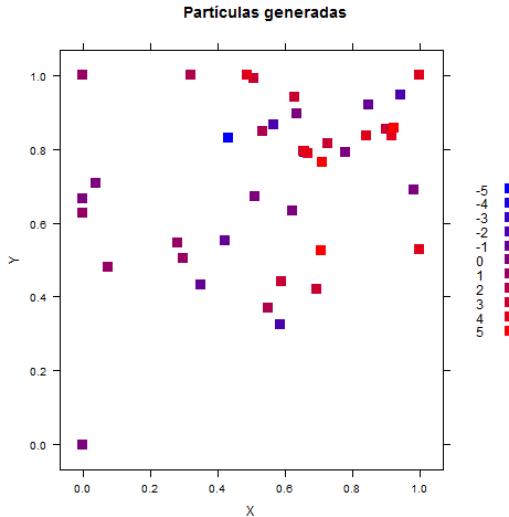


Figura 1: Posiciones iniciales de las partículas

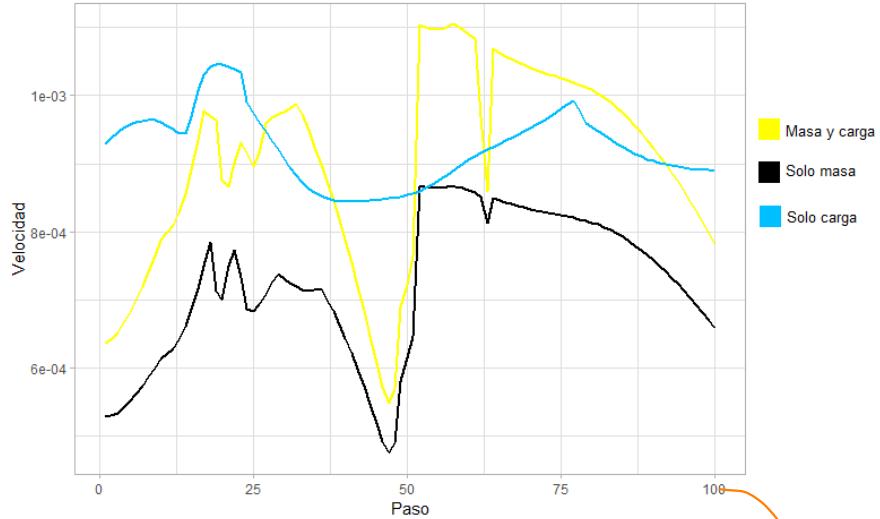


Figura 2: Gráfica de velocidades

## Referencias

- [1] Teresa Martín Blas. Fuerzas de la naturaleza. Universidad Politécnica de Madrid, 2010. URL [http://www2.montes.upm.es/dptos/digfa/cfisica/dinam1p/dinam1p\\_2.html](http://www2.montes.upm.es/dptos/digfa/cfisica/dinam1p/dinam1p_2.html).
- [2] Elisa Schaeffer. Práctica 9: Interacciones entre partículas. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.



# Práctica 9: Interacciones entre partículas

3175

2 de Abril de 2019

## 1. Introducción

Las fuerzas entre partículas son muchas y variadas, entre las más importantes están las fuerzas electroestáticas proporcionadas por las cargas que éstas poseen la cual determina si se atraen o se repelen y la fuerza de gravedad que afecta a todo aquello que tiene una masa o inclusive una masa aparente como lo es en el caso de algunas partículas como los electrones [III](#).

## 2. Objetivo

En base a un modelo de interacción entre elementos determinado por una simulación simple de cargas, agregar un factor adicional de un fenómeno como es la gravedad también de manera simple para comparar como estos dos fenómenos influyen en el comportamiento de los elementos dentro del sistema tomando como referencia la velocidad de estos.

## 3. Metodología

Usando de base el código proporcionado [2](#), se definen cincuenta elementos a los que consideramos partículas  $n$  cada una con una posición  $x$  y  $y$ , carga  $c$  (figura 1) y posteriormente se agrega un valor de masa  $m$  definido en base a valores dentro de una distribución normal en valores positivos de cero a uno, con excepción de las cargas donde estos valores pueden ser negativos entre menos uno y uno, el signo del valor de la carga describe su carga, la cual es necesaria para determinar si se atraen o se repelen dos partículas.

Dentro de la función de fuerza se deben incluir los criterios que definen el comportamiento por derivado de la masa, el cual es la fuerza de gravedad tomando en cuenta cual partícula tiene más masa que es la que va a atraer a la otra definiendo la dirección del movimiento de la fuerza, para finalmente aplicarse a las fuerzas totales que inciden. Se realiza esto para las fuerzas totales, para solo las cargas (como en el código base) y solamente tomando en cuenta la masa.

```
1 dirm <- (-1)^(1 + 1 * (mi < mj))  
1 factorm <- dirm * mi * mj / (dx^2 + dy^2)  
1 fxm <- (fxm - dx * factorm)
```

Finalmente se realizan las iteraciones para las cincuenta partículas en un lapso de tiempo de cien pasos tomando en cuenta los tres criterios antes mencionados, tomando como ejemplo el código [2](#). Definendo los cambios *delta* de posición diferentes para cada caso, en cada paso que es la unidad de tiempo.

```
1 suppressMessages(library(doParallel))  
2 registerDoParallel(makeCluster(detectCores() - 1))  
3 tmax <- 100
```

```

4  deltat <- c()
5  deltam <- c()
6  deltac <- c()
7  for (iter in 1:tmax) {
8    fm <- foreach(i = 1:n, .combine=c) %dopar % fuerzam(i)
9    fc <- foreach(i = 1:n, .combine=c) %dopar % fuerzac(i)
10   f <- foreach(i = 1:n, .combine=c) %dopar % fuerza(i)
11   delta <- 0.02 / max(abs(f))
12   deltam <- 0.02 / max(abs(fm))
13   deltac <- 0.02 / max(abs(fc))
14   deltat <- c(deltat, delta)
15   deltamc <- c(deltamc, deltam)
16   deltacc <- c(deltacc, deltac)

```

## 4. Resultados

Los resultados en la figura 2 muestran el comportamiento del sistema de partículas representado por líneas que describen el comportamiento de la velocidad, en el cual se aprecia la velocidad con cargas en color azul, la velocidad con masas en color negro y la combinación de ambas en línea amarilla.

## 5. Conclusiones

Se puede determinar en base a la gráfica que el comportamiento de la velocidad de las partículas con ambos fenómenos se ve mas fuertemente influenciado por la fuerza de gravedad que por las cargas, que sin embargo, sí modifica visiblemente el comportamiento de la velocidad pero en menor medida

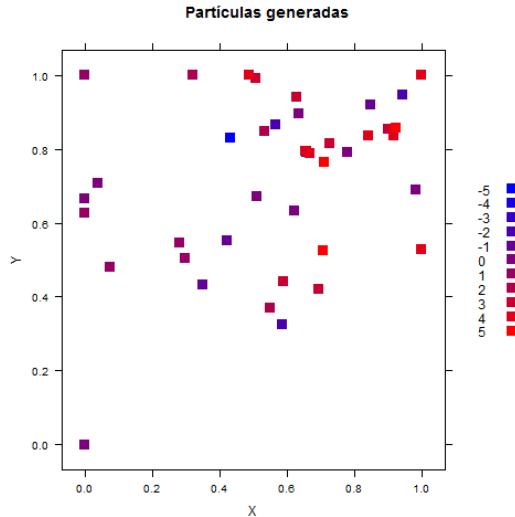


Figura 1: Posiciones iniciales de las partículas

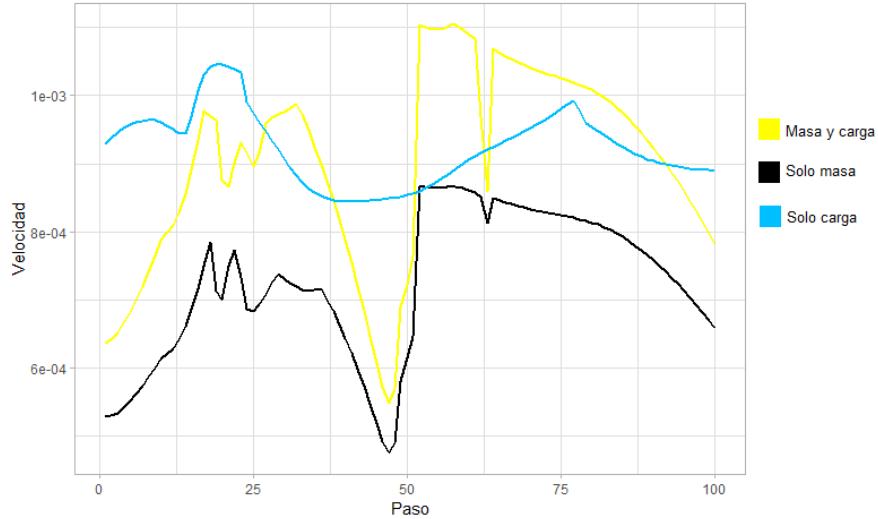


Figura 2: Gráfica de velocidades

## Referencias

- [1] Teresa Martín Blas. Fuerzas de la naturaleza. Universidad Politécnica de Madrid, 2010. URL [http://www2.montes.upm.es/dptos/digfa/cfisica/dinam1p/dinam1p\\_2.html](http://www2.montes.upm.es/dptos/digfa/cfisica/dinam1p/dinam1p_2.html).
- [2] Edson Raúl Cepeda Marquez. Interacciones entre partículas. R Paralelo: Simulación y análisis de datos, 2019. URL <https://sourceforge.net/projects/systemssimulation/files/P9/>.
- [3] Elisa Schaeffer. Práctica 9: Interacciones entre partículas. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p9.html>.

# Práctica 10

## Algoritmo Genético

### Revisión y Retroalimentación

- Se revisó la ortografía
- Se acomodaron figuras mal acomodadas
- Se corrigió la sección de introducción, pues tenía una que no estaba relacionada con la práctica, se pueden usar como base el código latex de prácticas anteriores pero hay que revisar muy bien que no se revuelvan.

# Práctica 10: Algoritmo Genético

3175

9 de abril de 2019



## 1. Introducción

Las fuerzas entre partículas son muchas y variadas, entre las más importantes están las fuerzas electroestáticas proporcionadas por las cargas que éstas poseen la cual determina si se atraen o se repelen y la fuerza de gravedad que afecta a todo aquello que tiene una masa o inclusive una masa aparente como lo es en el caso de algunas partículas como los electrones [?].

## 2. Objetivo

Determinar usando distintas reglas de asignación de valores y pesos la manera más eficiente de reslover, el algoritmo genético con la mejor selección de valores dentro de un rango de pesos permitido.

## 3. Metodología

Usando de base el código proporcionado[?], se obtiene la selección con la cantidad mas alta de la suma de valores que tengan peso permitido, por medio de la función *knapsack*, posteriormente se modifican las funciones para generar los valores de los pesos y valores: la primera con los valores asignados independientemente uno del otro, la segunda asignado independientemente pero correlacionado con el peso del objeto, esto es, entre mas pesado mas valioso, finalmente la ultima instancia es con los valores inversamente relacionados a los pesos, se verifica la distribución de los valores mediante gráficas de la figura 1.

Mediante el código:

```
1 #no correlacionado
2 generador.pesos <- function(cuantos, min, max) {
3   return(sort(round(normalizar(rnorm(cuantos)) * (max - min) + min)))
4 }
5
6 generador.valores <- function(pesos, min, max) {
7   n <- length(pesos)
8   valores <- double()
9   for (i in 1:n) {
10     media <- pesos[n]
11     desv <- runif(1)
12     valores <- c(valores, rnorm(1))
13   }
14   valores <- normalizar(valores) * (max - min) + min
15   return(valores)
```

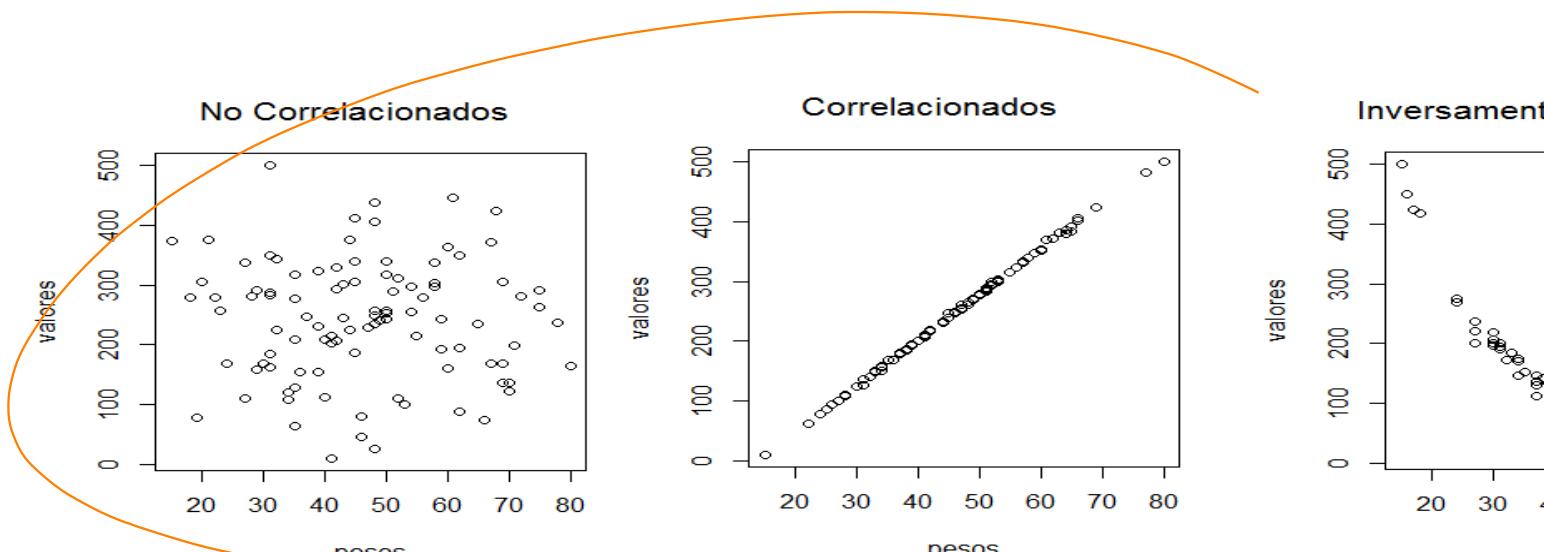


Figura 1: Distribución de valores y pesos de los objetos

```

16 }
17 #CORRELACIONADO
18 generador.pesos <- function(cuantos, min, max) {
19   return(sort(round(normalizar(rnorm(cuantos)) * (max - min) + min)))
20 }
21
22 generador.valores <- function(pesos, min, max) {
23   n <- length(pesos)
24   valores <- double()
25   for (i in 1:n) {
26     media <- pesos[n]
27     desv <- runif(1)
28     valores <- c(valores, (rnorm(1, media, desv) * (pesos[i])) + (rnorm(1)/10))
29   }
30   valores <- normalizar(valores) * (max - min) + min
31   return(valores)
32 }
33 #Inversamente CORRELACIONADO
34 generador.pesos <- function(cuantos, min, max) {
35   return(sort(round(normalizar(rnorm(cuantos)) * (max - min) + min)))
36 }
37
38 generador.valores <- function(pesos, min, max) {
39   n <- length(pesos)
40   valores <- double()
41   for (i in 1:n) {
42     media <- pesos[n]
43     desv <- runif(1)
44     valores <- c(valores, (rnorm(1, media, desv) / (pesos[i])) + (rnorm(1)/10))
45   }
46   valores <- normalizar(valores) * (max - min) + min
47   return(valores)
48 }
```

Finalmente se realiza la ejecución en paralelo del algoritmo genético y de la función *knapsack*

```

1 library(parallel)
2 cluster <- makeCluster(detectCores() - 1)
3 clusterExport(cluster, c("mutar", "mutacion", "reproduccion", "normalizar", "reproducete", "obj")
4 tamao <- seq(50, 100, by= 25)
5 for (n in tamao){
6   inicio1 <- Sys.time()
7   pesos <- generador.pesos(tamao, 15, 80)
8   valores <- generador.valores(pesos, 10, 500)
9   capacidad <- round(sum(pesos) * 0.65)
10  print(n)
11  optimo <- knapsack(capacidad, pesos, valores)
12  final1 <- Sys.time()
13 #termina optimo
14  inicial1 <- Sys.time()
15  init <- 200
16  p <- poblacion.inicial(n, init)
17  tam <- dim(p)[1]
18  assert(tam == init)
19  pm <- 0.05
20  rep <- 50
21  tmax <- 50
22  mejores <- double()
23  clusterExport(cluster, c("n", "pm", "valores", "pesos", "capacidad", "objetivo"))
24  for (iter in 1:tmax) {
25    p$obj <- NULL
26    p$fact <- NULL
27    clusterExport(cluster, "p")
28    mute <- parSapply(cluster, 1:tam, mutar)
29    for(i in 1:length(mute)){
30      if(!is.null(mute[[i]])){
31        p <- rbind(p, mute[[i]])
32      }
33    }
34    clusterExport(cluster, c("tam", "p"))
35    S <- parSapply(cluster, 1:rep, reproducete) # una cantidad fija de reproducciones
36
37    for (agrego in 1:length(S)) {
38      p <- rbind(p, S[[agrego]])
39    }
40    tam <- dim(p)[1]
41    obj <- double()
42    fact <- integer()
43    clusterExport(cluster, c("p", "tam", "factibilidad", "objet"))
44    obj <- parSapply(cluster, 1:tam, objet)
45    fact <- parSapply(cluster, 1:tam, factibilidad)
46    p <- cbind(p, obj)
47    p <- cbind(p, fact)
48    mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
49    p <- p[mantener,]
50    tam <- dim(p)[1]
51    assert(tam == init)
52    factibles <- p[p$fact == TRUE,]
```

```

53     mejor <- max(factibles$obj)
54     mejores <- c(mejores, mejor)
55   }
56 termino1<- Sys.time()

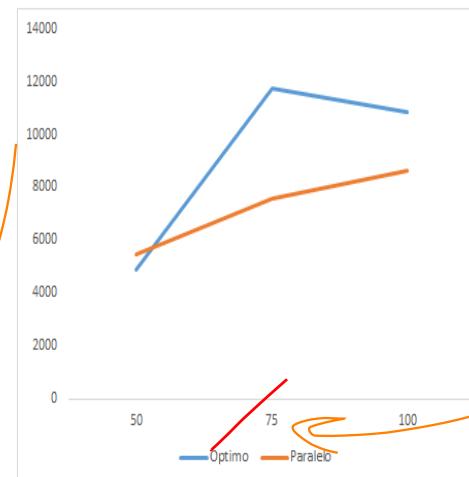
```

## 4. Resultados

Los resultados en las figuras 2,3 y 4 muestran que las asignaciones de los valores si influyen en la cantidad de tiempo que le toma resolver el algoritmo genético .

## 5. Conclusiones

Se puede determinar que los valores determinan en gran medida el tiempo que toma resolver el algoritmo genético y éste mejora contra la función *knapsack* conforme aumentan los valores.



excel

Gráfica de valores no correlacionados

## Referencias

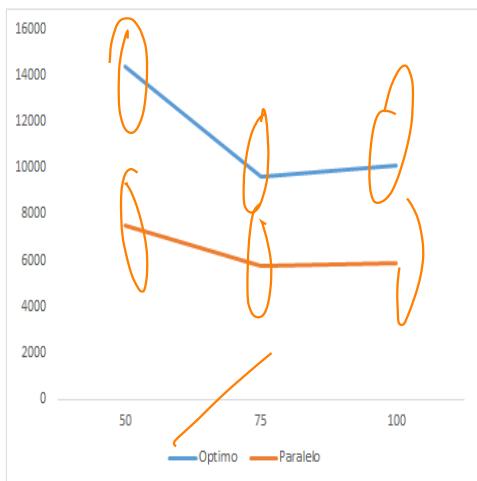


Figura 2: Gráfica valores correlacionados

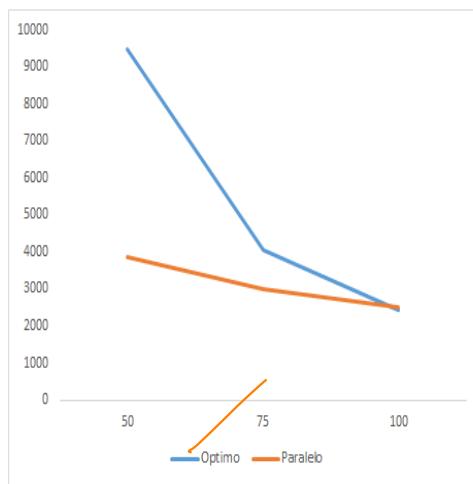


Figura 3: Gráfica de valores inversamente correlacionados

# Práctica 10: Algoritmo Genético

3175

9 de Abril de 2019

## 1. Introducción

El algoritmo genético surgió en los años 70's de la mano de *John Henry Holland* como una de las líneas mas prometedoras de la inteligencia artificial. Son llamados así por que se inspiran en la evolución biológica y su base genético-molecular [1].

## 2. Objetivo

Determinar usando distintas reglas de asignación de valores y pesos la manera mas eficiente de reslover, el algoritmo genético con la mejor selección de valores dentro de un rango de pesos permitido.

## 3. Metodología

Usando de base el código proporcionado [3], se obtiene la selección con la cantidad más alta de la suma de valores que tengan peso permitido, por medio de la función *knapsack*, posteriormente se modifican las funciones para generar los valores de los pesos y valores: la primera con los valores asignados independientemente uno del otro, la segunda asignado independientemente pero correlacionado con el peso del objeto, esto es, entre más pesado más valioso, finalmente la última instancia es con los valores inversamente relacionados a los pesos, se verifica la distribución de los valores mediante gráficas de la figura 1.

Mediante el código:

```
1 #no correlacionado
2 generador.valores <- function(pesos, min, max) {
3   n <- length(pesos)
4   valores <- double()
5   for (i in 1:n) {
6     media <- pesos[n]
7     desv <- rnorm(1)
8     valores <- c(valores, rnorm(1))
9   }
10  valores <- normalizar(valores) * (max - min) + min
11  return(valores)
12 }
```

```
1 #correlacionado
2 generador.valores <- function(pesos, min, max) {
3   n <- length(pesos)
4   valores <- double()
```

```

5   for (i in 1:n) {
6     media <- pesos[n]
7     desv <- rnorm(1)
8     valores <- c(valores , (rnorm(1, media , desv)* (pesos [ i ])) + (rnorm(1)/10))
9   }
10    valores <- normalizar(valores) * (max - min) + min
11  return(valores)
12 }

1 #Inversamente correlacionado
2 generador.valores <- function(pesos , min, max) {
3   n <- length(pesos)
4   valores <- doublefor (i in 1:n) {
6     media <- pesos[n]
7     desv <- rnorm(1)
8     valores <- c(valores , (rnorm(1, media , desv) / (pesos [ i ])) + (rnorm(1)/10))
9   }
10  valores <- normalizar(valores) * (max - min) + min
11  return(valores)
12 }
```

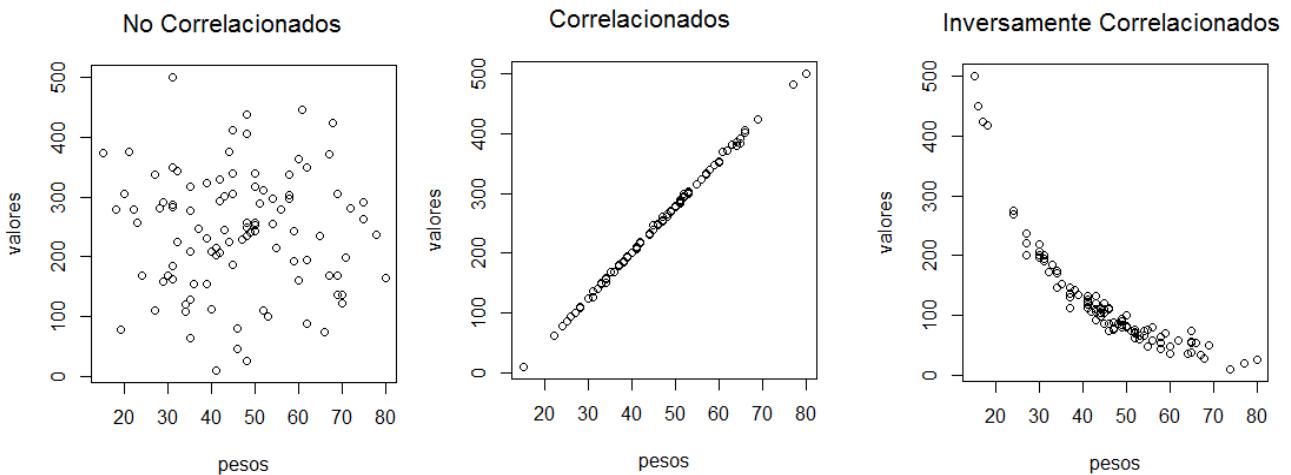


Figura 1: Distribución de valores y pesos de los objetos no correlacionados, correlacionados e inversamente correlacionados respectivamente

Finalmente se realiza la ejecución en paralelo tomando como referencia el código [2] del algoritmo genético y de la función *knapsack*:

```

1 library(parallel)
2 cluster <- makeCluster(detectCores() - 1)
3 clusterExport(cluster , c("mutar" , "mutacion" , "reproduccion",
4                           "normalizar" , "reproducete" , "objetivo" , "factible"))
5 tam <- seq(50, 100, by= 25)
6 for (n in tam){
7   inicio1 <- Sys.time()
8   pesos <- generador.pesos(tam, 15, 80)
```

```

9    valores <- generador.valores(pesos, 10, 500)
10   capacidad <- round(sum(pesos) * 0.65)
11   print(n)
12   optimo <- knapsack(capacidad, pesos, valores)
13   final1 <- Sys.time()
14   #termina optimo
15   inicial1 <- Sys.time()
16   init <- 200
17   p <- poblacion.inicial(n, init)
18   tam <- dim(p)[1]
19   assert(tam == init)
20   pm <- 0.05
21   rep <- 50
22   tmax <- 50
23   mejores <- double()
24   clusterExport(cluster, c("n", "pm", "valores", "pesos", "capacidad", "objetivo"))
25   for (iter in 1:tmax) {
26     p$obj <- NULL
27     p$fact <- NULL
28     clusterExport(cluster, "p")
29     mute <- parSapply(cluster, 1:tam, mutar)
30     for(i in 1:length(mute)){
31       if(!is.null(mute[[i]])){
32         p <- rbind(p, mute[[i]])
33       }
34     }
35     clusterExport(cluster, c("tam", "p"))
36     S <- parSapply(cluster, 1:rep, reproducete) # una cantidad fija de reproducciones
37
38     for (agrego in 1:length(S)) {
39       p <- rbind(p, S[[agrego]])
40     }
41     tam <- dim(p)[1]
42     obj <- double()
43     fact <- integer()
44     clusterExport(cluster, c("p", "tam", "factibilidad", "objet"))
45     obj <- parSapply(cluster, 1:tam, objet)
46     fact <- parSapply(cluster, 1:tam, factibilidad)
47     p <- cbind(p, obj)
48     p <- cbind(p, fact)
49     mantener <- order(-p[, (n + 2)], -p[, (n + 1)])[1:init]
50     p <- p[maintener,]
51     tam <- dim(p)[1]
52     assert(tam == init)
53     factibles <- p[p$fact == TRUE,]
54     mejor <- max(factibles$obj)
55     mejores <- c(mejores, mejor)
56   }
57   termino1<- Sys.time()

```

## 4. Resultados

Los resultados en las figuras 2,3 y 4 muestran que las asignaciones de los valores sí influyen en la cantidad de tiempo que le toma resolver el algoritmo genético.

## 5. Conclusiones

Se puede determinar que los valores determinan en gran medida el tiempo que toma resolver el algoritmo genético y éste mejora contra la función *knapsack* conforme aumentan los valores.

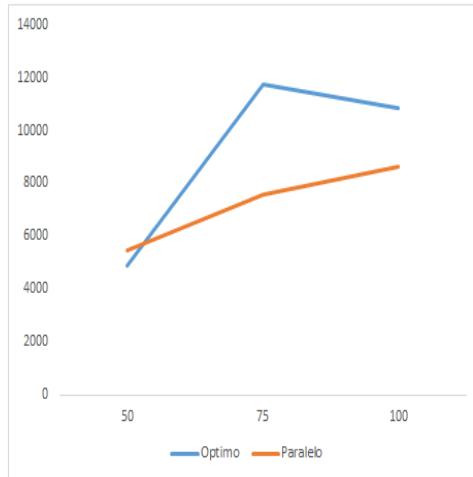


Figura 2: Gráfica de valores no correlacionados

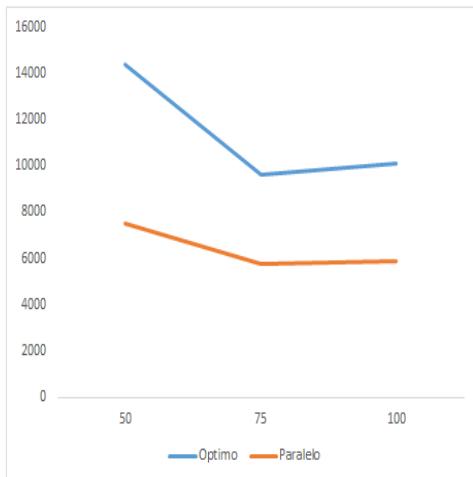


Figura 3: Gráfica valores correlacionados

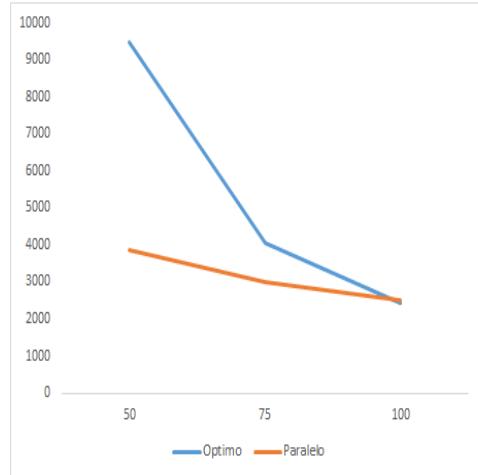


Figura 4: Gráfica de valores inversamente correlacionados

## Referencias

- [1] D. E. Goldberg. Addison-Wesley. Genetic algorithms in search, optimization and machine learning. Longman Publishing Co., Inc., Boston, MA, USA, 1989.
- [2] Yessica Reina Fernández. Práctica 10: Algoritmo genético. R Paralelo: Simulación y análisis de datos, 2018. URL <https://sourceforge.net/projects/simulacion-de-sistemas/files/Practica%2010/>.
- [3] Elisa Schaeffer. Práctica 10: Algoritmo genético. Página Web, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p10.html>.

# Práctica 11

## Frentes de Pareto

### **Revisión y Retroalimentación**

- Se revisó la ortografía y se usó la notación científica pues ésta resulta más sencilla de entender.
- Se realizaron los retos.



# Práctica 11: Frentes de Pareto

3175

30 de abril de 2019

## 1. Introducción

Las frentes de Pareto reciben el nombre en honor a Vilfredo Pareto, quien fué quien los enunció por primera vez ya que empíricamente observó que las poblaciones se distribuían las pertenencias entre muy pocos, por ejemplo observó que el ochenta porciento de las tierras en Italia le pertenecían al veinte porciento de las personas[1].



## 2. Objetivo

Analizar por medio de la simulación de polinomios y las soluciones de estos, la distribución de soluciones de pareto utilizando diferente cantidad de funciones objetivo.

## 3. Metodología

Usando de base el código proporcionado[3], se genera polinomios al azar, los cuales se van a utilizar como restricciones y funciones objetivo, además como parámetros se definieron a contenerlos cinco términos, cuatro variables y un grado máximo de tres. Posteriormente se generan las soluciones iniciales que van a ser evaluadas entre sí para construir así la frente de pareto. Para la práctica se realizaron simulaciones con funciones objetivo diferentes, comenzando con dos hasta llegar a diez, realizando 60 repeticiones para obtener una muestra significativa y generándose al principio 200 y 500 soluciones iniciales. El proceso se paralelizó mediante el uso de la librería *Parallel*:

```
1 library(parallel)
2 cl <- makeCluster(detectCores()-1)
3 clusterExport(cl, funciones)
4 clusterExport(cl, parametros)
5 n <- 500 #soluciones iniciales
6 resultados <- data.frame(Funciones = integer(), Replica = integer(),
7                               SolNoDom = integer(), Porcentaje = numeric())
8 for (k in 2:K) {
9   clusterExport(cl, "k")
10  for (r in 1:repeticion) {
11
12    obj <- parSapply(cl, 1:k, function(i) {
13      return(list(poli(md, vc, tc)))
14    })
15    minim <- (runif(k) > 0.5)
16    sign <- (1 + -2 * minim)
17    clusterExport(cl, c("obj", "sign", "n"))
```

```

18     sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
19     val <- matrix(parRapply(cl, sol, function(i) {
20       evaluacion <- double()
21       for (j in 1:k) { # para todos los objetivos
22         evaluacion <- c(evaluacion, eval(obj[[j]], i, tc))
23       }
24       return(evaluacion)
25     }), nrow=n, ncol=k, byrow = TRUE)
26
27   datos <- t(parSapply(cl, 1:n, function(i) {
28     d <- logical()
29     for (j in 1:n) {
30       d <- c(d, domin.by(sign * val[i,], sign * val[j,], k))
31     }
32     dominadores <- sum(d)
33     no.dom <- dominadores == 0 # nadie le domina
34     return(c(dominadores, no.dom))
35   }))

```

Finalmente se realizaron ~~2~~<sup>dos</sup> gráficas, con 200 y 500 soluciones iniciales, utilizando gráficas de violín y de caja bigote para mayor facilidad visualización<sup>[2]</sup>.

## 4. Resultados

Los resultados en las figuras 1 y 2 muestran una tendencia marcada al aumento del porcentaje de soluciones no dominadas conforme se aumenta la cantidad de funciones objetivo. Se encontró que los resultados en ambos casos presentan diferencias significativas entre los resultados obtenidos en las diferente cantidad de funciones objetivo, debido a que al realizar la prueba ANOVA el factor de Fisher fue menor a 0.05 en ambos casos.

```

1 Con 200 resultados
2
3           Df Sum Sq Mean Sq F value Pr(>F)
4 Funciones      8  56.40    7.051   177.6 <2e-16 ***
5 Residuals    531  21.08    0.040
6
7 Con 500 resultados
8           Df Sum Sq Mean Sq F value Pr(>F)
9 Funciones      8  59.79    7.473   172.1 <2e-16 ***
10 Residuals    531  23.06    0.043

```

$2 \times 10^{-16}$

## 5. Conclusiones

Con base en los resultados obtenidos se pueden razonar que al aumentar el número de funciones objetivo aumenta el porcentaje de soluciones dominantes debido a que al buscar más objetivos las soluciones se distribuyen entre todas éstas disminuyendo la cantidad de competencia directa entre éstas. Evidencia de esta conclusión se puede observar ya que al aumentar el número de soluciones a 500 en los lugares donde casi se alcanza el cien porciento se distribuye mayormente, aun que con la misma tendencia.

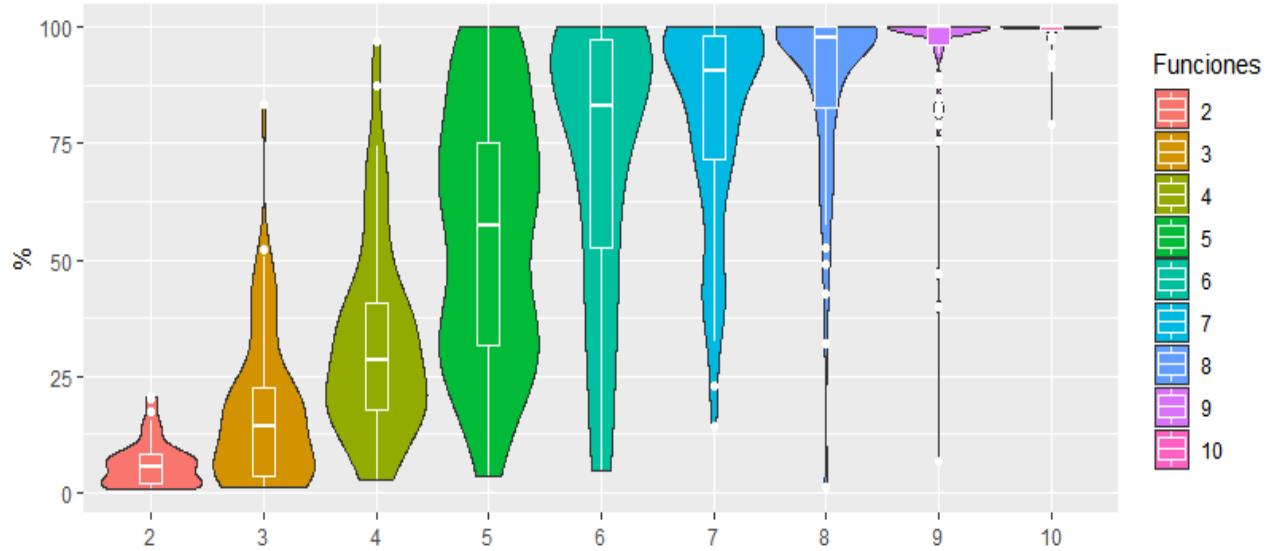


Figura 1: Porcentajes de resultados dominantes encontrados con diferentes funciones objetivo generando 200 soluciones al principio

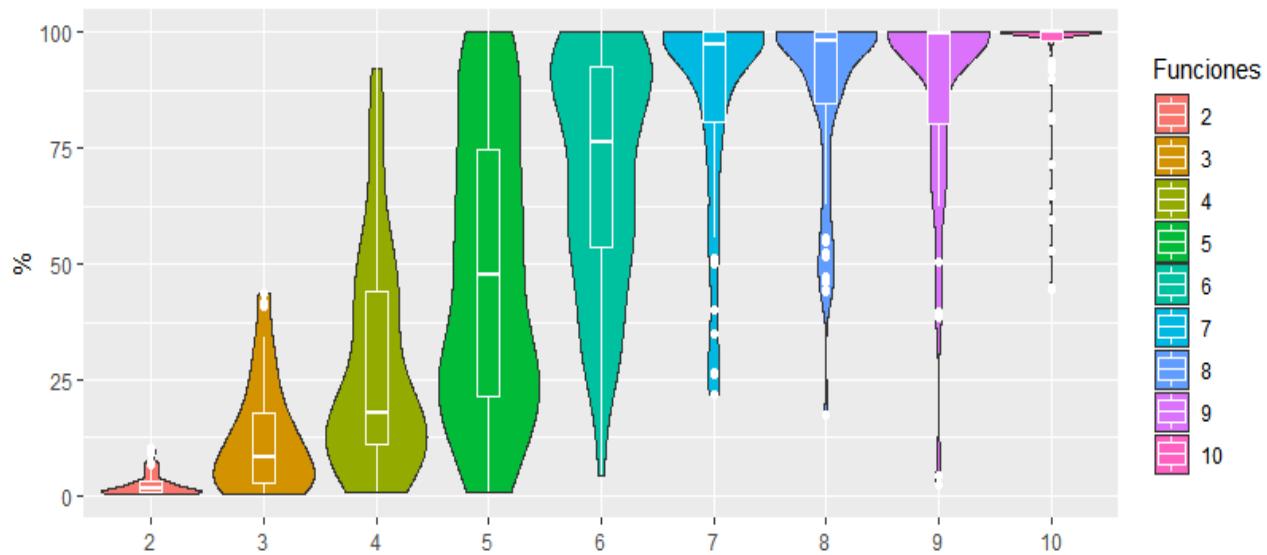


Figura 2: Porcentajes de resultados dominantes encontrados con diferentes funciones objetivo generando 500 soluciones al principio

## Referencias

- [1] Gehisy. El diagrama de pareto. Página Web: Calidad y ADR, 2017. URL <https://aprendiendocalidadyadr.com/diagrama-de-pareto/>.
- [2] Angel Moreno. Frentes de pareto. Repositorio GitHub, 2018. URL <https://github.com/angisabel44/Simulation/tree/master/Homework11>.

{Pareto}

- [3] Elisa Schaeffer. Práctica 11: Frentes de pareto. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.

# Práctica 11: Frentes de Pareto

3175

30 de Abril de 2019

## 1. Introducción

Las frentes de Pareto reciben el nombre en honor a Vilfredo Pareto, quien fué quien los enunció por primera vez ya que empíricamente observó que las poblaciones se distribuían las pertenencias entre muy pocos, por ejemplo observó que el ochenta porciento de las tierras en Italia le pertenecían al veinte porciento de las personas [1].

## 2. Objetivo

Analizar por medio de la simulación de polinomios y las soluciones de estos, la distribución de soluciones de Pareto utilizando diferente cantidad de funciones objetivo.

## 3. Metodología

Usando de base el código proporcionado [2], se genera polinomios al azar, los cuales se van a utilizar como restricciones y funciones objetivo, además como parámetros se definieron a contenerlos cinco términos, cuatro variables y un grado máximo de 3. Posteriormente se generan las soluciones iniciales que van a ser evaluadas entre sí para construir así la frente de Pareto. Para la práctica se realizaron simulaciones con funciones objetivo diferentes, comenzando con dos hasta llegar a diez, realizando 60 repeticiones para obtener una muestra significativa y generándose al principio 200 y 500 soluciones iniciales. El proceso se parallelizó mediante el uso de la librería *Parallel*:

```
1 library(parallel)
2 cl <- makeCluster(detectCores()-1)
3 clusterExport(cl, funciones)
4 clusterExport(cl, parametros)
5 n <- 500 #soluciones iniciales
6 resultados <- data.frame(Funciones = integer(), Replica = integer(),
7                               SolNoDom = integer(), Porcentaje = numeric())
8 for (k in 2:K) {
9   clusterExport(cl, "k")
10  for (r in 1:repeticion) {
11    obj <- parSapply(cl, 1:k, function(i) {
12      return(list(poli(md, vc, tc)))
13    })
14    minim <- (runif(k) > 0.5)
15    sign <- (1 + -2 * minim)
16    clusterExport(cl, c("obj", "sign", "n"))}
```

```

18     sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
19     val <- matrix(parRapply(cl, sol, function(i) {
20       evaluacion <- double()
21       for (j in 1:k) { # para todos los objetivos
22         evaluacion <- c(evaluacion, eval(obj[[j]], i, tc))
23       }
24       return(evaluacion)
25     }), nrow=n, ncol=k, byrow = TRUE)
26
27     datos <- t(parSapply(cl, 1:n, function(i) {
28       d <- logical()
29       for (j in 1:n) {
30         d <- c(d, domin.by(sign * val[i,], sign * val[j,], k))
31       }
32       dominadores <- sum(d)
33       no.dom <- dominadores == 0 # nadie le domina
34       return(c(dominadores, no.dom))
35     }))

```

Finalmente se realizaron dos gráficas, con 200 y 500 soluciones iniciales, utilizando gráficas de violín y de caja bigote para mayor facilidad visualización [2].

## 4. Resultados

Los resultados en las figuras 1 y 2 muestran una tendencia marcada al aumento del porcentaje de soluciones no dominadas conforme se aumenta la cantidad de funciones objetivo. Se encontró que los resultados en ambos casos presentan diferencias significativas entre los resultados obtenidos en las diferente cantidad de funciones objetivo, debido a que al realizar la prueba ANOVA el factor de Fisher fue menor a 0.05 en ambos casos.

```

1 Con 200 resultados
2
3             Df Sum Sq Mean Sq F value Pr(>F)
4 Funciones      8   56.40    7.051   177.6 <2 × 10⁻¹⁶ ***
5 Residuals    531   21.08    0.040
6
7 Con 500 resultados
8             Df Sum Sq Mean Sq F value Pr(>F)
9 Funciones      8   59.79    7.473   172.1 <2 × 10⁻¹⁶ ***
10 Residuals    531   23.06    0.043

```

## 5. Conclusiones

Con base en los resultados obtenidos se pueden razonar que al aumentar el número de funciones objetivo aumenta el porcentaje de soluciones dominantes debido a que al buscar más objetivos las soluciones se distribuyen entre todas éstas disminuyendo la cantidad de competencia directa entre éstas. Evidencia de esta conclusión se puede observar ya que al aumentar el número de soluciones a 500 en los lugares donde casi se alcanza el cien porciento se distribuye mayormente, aun que con la misma tendencia.

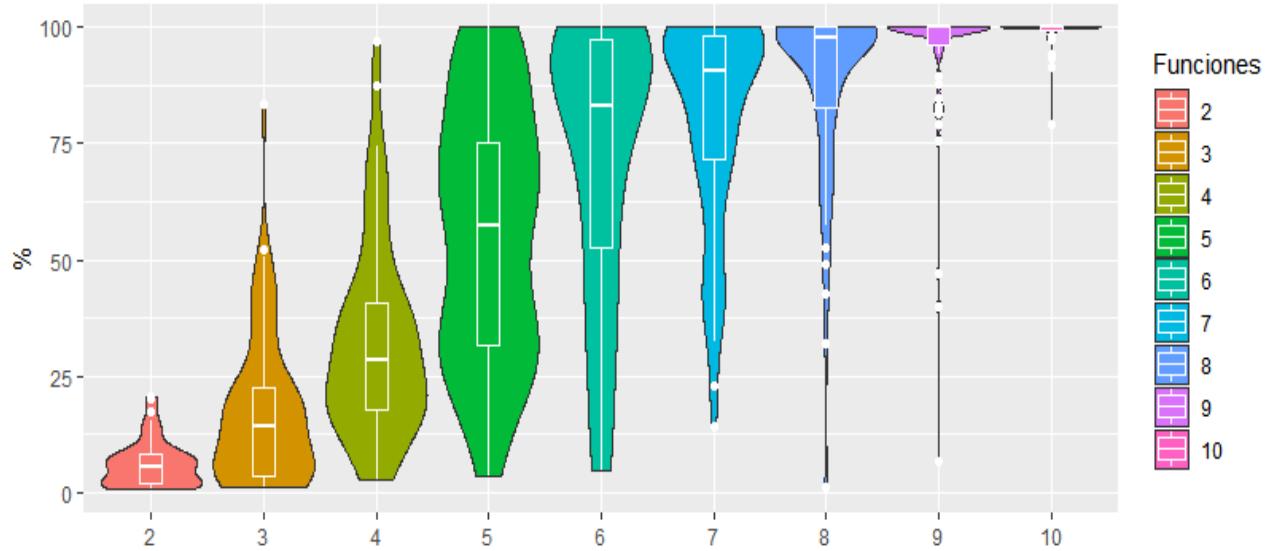


Figura 1: Porcentajes de resultados dominantes encontrados con diferentes funciones objetivo generando 200 soluciones al principio

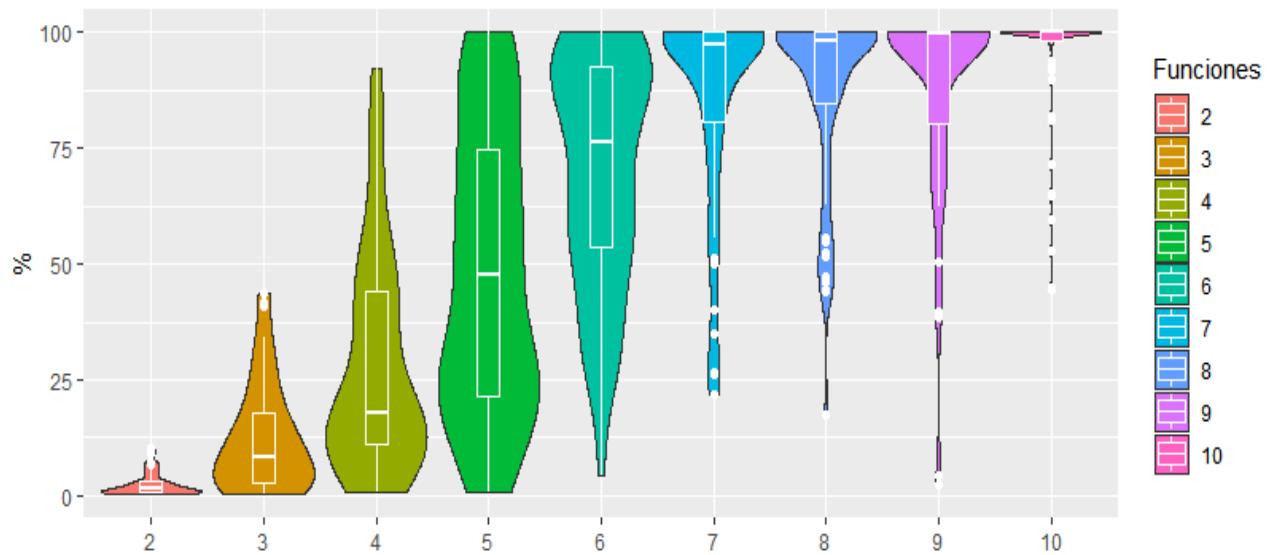


Figura 2: Porcentajes de resultados dominantes encontrados con diferentes funciones objetivo generando 500 soluciones al principio

## 6. Reto 1

El primer reto consiste en seleccionar un subconjunto del frente de Pareto de tal forma que la solución esté diversificada, es decir, no formen parte del total del frente de Pareto. Se realizó para un total de trescientas soluciones aleatorias.

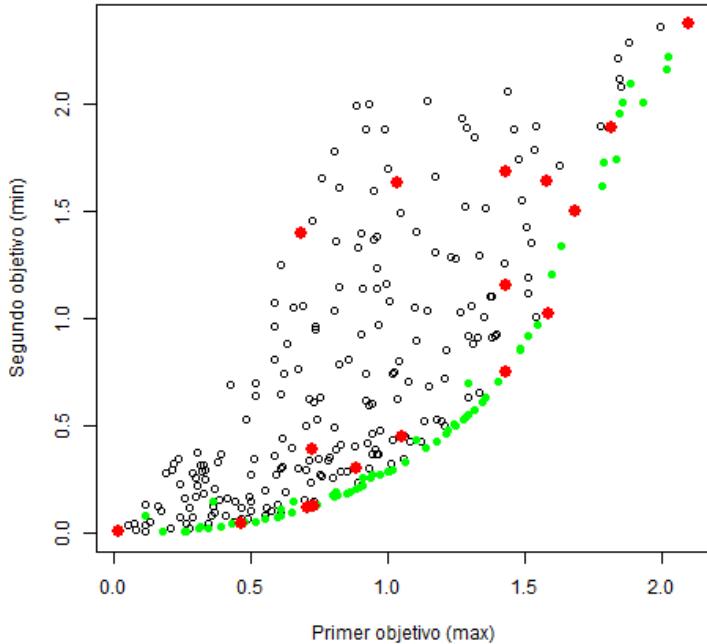


Figura 3: Frente de Pareto total en verde y el 35 % de soluciones diversificadas en rojo

## 7. Reto 2

El segundo reto consiste en adaptar el algoritmo genético para que vaya mejorando buscando al frente de Pareto, creando un gif animado con las imágenes para mejor visualización. La prueba se hizo con una probabilidad de mutación de 0,03, treinta repeticiones y treinta pasos. El gif se encuentra en el repositorio.

## Referencias

- [1] Gehisy. El diagrama de Pareto. Página Web: Calidad y ADR, 2017. URL <https://aprendiendocalidadyadr.com/diagrama-de-pareto/>.
- [2] Angel Moreno. Frentes de Pareto. Repositorio GitHub, 2018. URL <https://github.com/angisabel44/Simulation/tree/master/Homework11>.
- [3] Elisa Schaeffer. Práctica 11: Frentes de Pareto. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p11.html>.

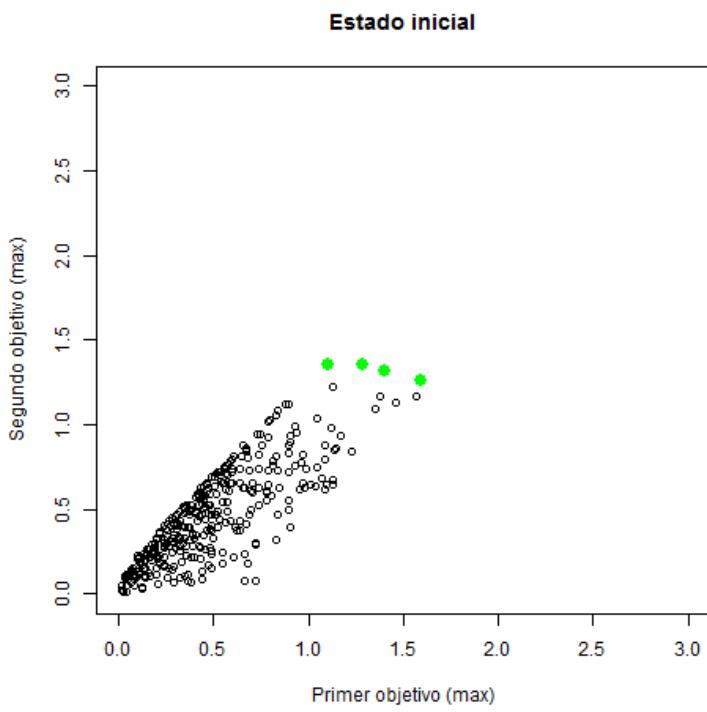


Figura 4: Imagen inicial

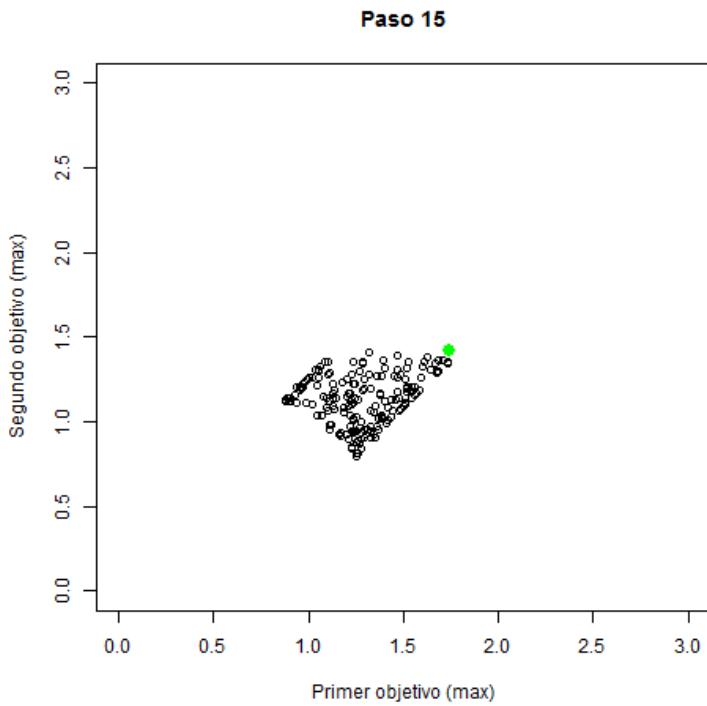


Figura 5: Posiciones a mitad del proceso

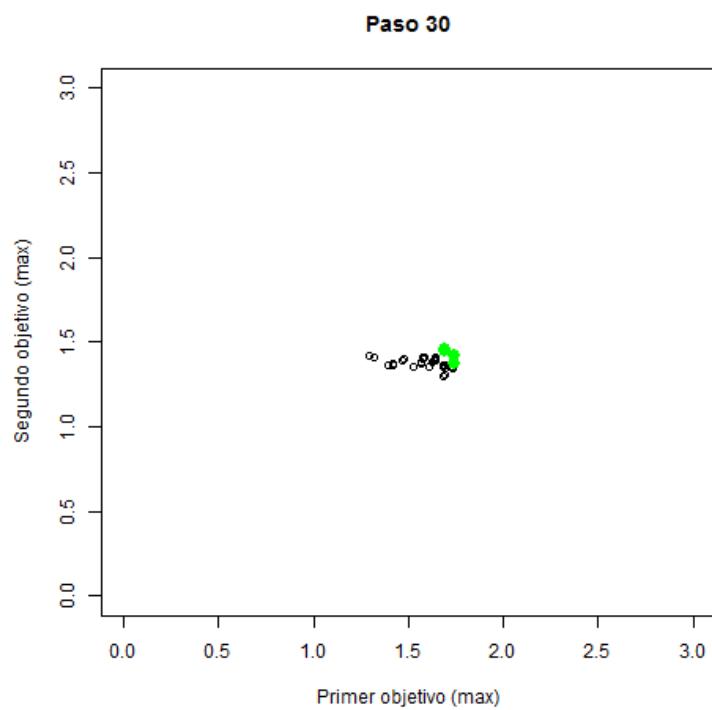


Figura 6: Posiciones al finalizar

# Práctica 12

## Red Neuronal

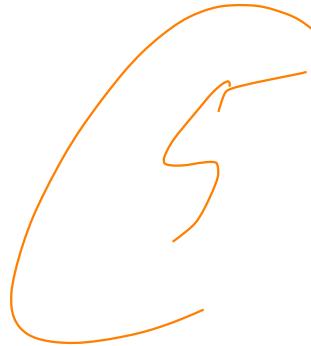
### Revisión y Retroalimentación

- Se revisó la ortografía
- Se aprendió el uso de “\times” en vez de poner la letra “X” y “\%”
- Se corrigieron nombres en bibliografía
- Se realizó el Reto 1
- Se deben guardar las gráficas con mejor resolución para que al colocarlas en el trabajo no se distorsionen al hacer zoom.

# Práctica 12: Red Neuronal

3175

7 de mayo de 2019



## 1. Introducción

Las Redes Neuronales son un campo muy importante dentro de la Inteligencia Artificial. Inspirándose en el comportamiento conocido del cerebro humano, trata de crear modelos artificiales que solucionen problemas difíciles de resolver mediante técnicas algorítmicas convencionales. Las primeros ejemplos de redes neuronales artificiales aparecen al final de la década de los cincuenta y la referencia histórica más corriente es la que alude al trabajo realizado por *Frank Rosenblatt* en un dispositivo denominado perceptrón [1].

## 2. Objetivo

Mediante una red neuronal que es una demostración básica de aprendizaje de máquina se desea analizar el desempeño de la misma, tomando como objetivo reconocer dígitos de imágenes pequeñas en blanco y negro.

## 3. Metodología

Usando de base el código proporcionado [3], Se usan perceptrones para determinar si el producto de  $X$  y  $w$ (peso) es positiva en cuyo caso se determina como VERDADERO o si no FALSO. Para convertir un entero en una secuencia de bits que indican cuáles potencias están presentes y cuáles están ausentes de la suma en donde los posibles residuos en división entre dos son solamente cero y uno), simplemente se prueban cuáles potencias le caben. Posteriormente se de usan 15 pixeles por dígito en un cuadro de  $3 \times 5$ . Los cuales son determinados de manera probabilística para los 10 dígitos usando colores negro, gris, blanco modelados por un archivo CSV.

1 modelos <- **read.csv**("modelos2.csv", sep=" ", header=FALSE, stringsAsFactors=F)

Las probabilidades para el color del pixeles negros de 0,995 y 0,8, los blancos 0,001 y 0,1, finalmente para los grises de 0,75 y 0,65. Finalmente con referencia en el código[2], se paralelizó la etapa de la prueba llamando a las funciones a los núcleos por medio del paquete *parSapply* y se realizaron 30 réplicas:

```
1 clusterExport(cluster, c("neuronas", "binario", "decimal", "modelos", "tope", "k",
2                         "dim", "n"))
3 contadores <- parSapply(cluster, 1:prueba, function(x){
4   d <- sample(0:tope, 1)
5   pixeles <- runif(dim) < modelos[d + 1,]
6   correcto <- binario(d, n)
7   salida <- rep(FALSE, n)
8   for (i in 1:n) {
9     w <- neuronas[i,]
10    deseada <- correcto[i]
```

```

11     resultado <- sum(w * pixeles) >= 0
12     salida [ i ] <- resultado
13   }
14   r <- min(decimal(salida , n) , k)
15   return(r==d))
16 datos<-rbind(datos , data.frame(Replica= replicas , PNegras=PN, PGrises=PG, PBlancas=PB,

```

## 4. Resultados

Los resultados de la efectividad de la red neuronal se midieron en base al porcentaje de contadores entre el número de pruebas para los distintos casos de probabilidad de los píxeles, para así notar que tan precisa es la red neuronal en base al modelo proporcionado (los dígitos).

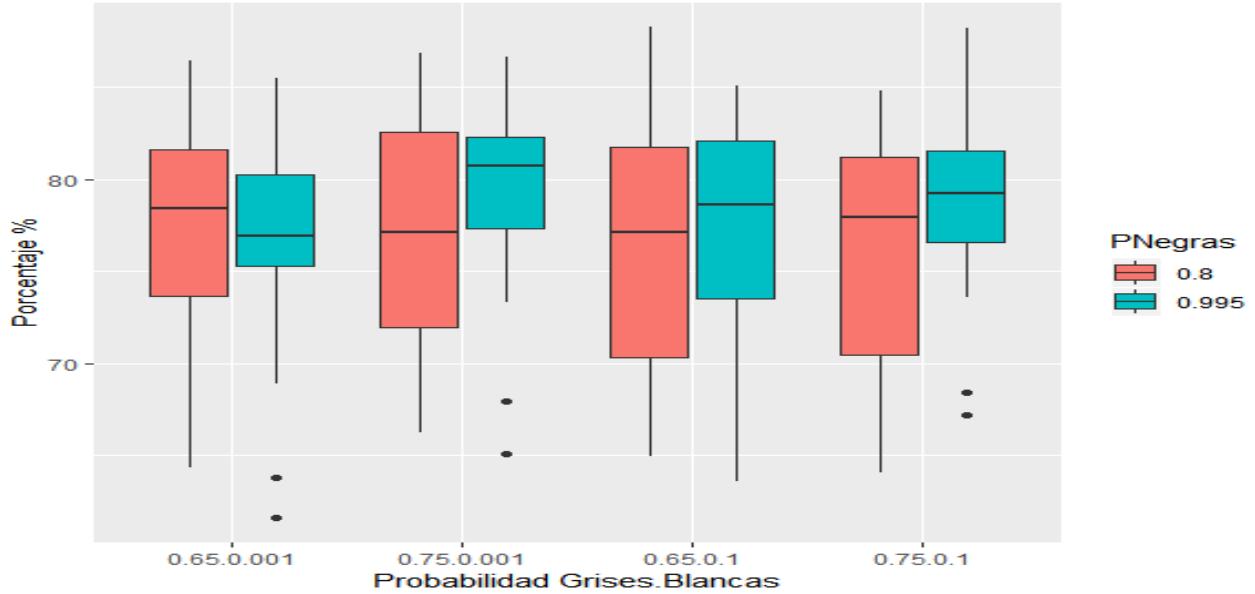


Figura 1: Porcentaje de precisión de la red neuronal con distintas probabilidades

## 5. Conclusiones

Se puede concluir que la mayoría de aciertos se presentó para las probabilidades de 0,995 hablando de los píxeles negros, 0,75 al referirnos a los píxeles grises y 0,001 en el caso de los píxeles blancos. Probablemente debido a que la mayoría de los píxeles de los modelos eran negros o grises y en menor medida blancos, por eso se encontraron favorecidas las probabilidades altas en los blancos y grises, y la menor en los blancos. Se pueden hacer pruebas posteriores con distintas iteraciones y modelos para corroborar éstas afirmaciones.

## Referencias

- [1] Universidad de Salamanca. Redes neuronales. Página Web: Redes Neuronales, 2016. URL <http://avellano.fis.usal.es/~lalonso/RNA/index.htm>.

- [2] Liliana Saus. Redes neuronales. Repositorio GitHub, 2018. URL <https://github.com/pejli/simulacion/tree/master/P12>.
- [3] Elisa Schaeffer. Práctica 12: Redes neuronales. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p12.html>.

# Práctica 12: Red Neuronal

3175

7 de Mayo de 2019

## 1. Introducción

Las Redes Neuronales son un campo muy importante dentro de la Inteligencia Artificial. Inspirándose en el comportamiento conocido del cerebro humano, trata de crear modelos artificiales que solucionen problemas difíciles de resolver mediante técnicas algorítmicas convencionales. Las primeros ejemplos de redes neuronales artificiales aparecen al final de la década de los cincuenta y la referencia histórica más corriente es la que alude al trabajo realizado por *Frank Rosenblatt* en un dispositivo denominado perceptrón [1].

## 2. Objetivo

Mediante una red neuronal que es una demostración básica de aprendizaje de máquina se desea analizar el desempeño de la misma, tomando como objetivo reconocer dígitos de imágenes pequeñas en blanco y negro.

## 3. Metodología

Usando de base el código proporcionado [2], se usan perceptrones para determinar si el producto de X y W(peso) es positiva en cuyo caso se determina como VERDADERO o si no FALSO. Para convertir un entero en una secuencia de bits que indican cuáles potencias están presentes y cuáles están ausentes de la suma en donde los posibles residuos en división entre dos son solamente cero y uno), simplemente se prueban cuáles potencias le caben. Posteriormente se de usan 15 pixeles por dígito en un cuadro de  $3 \times 5$ . Los cuales son determinados de manera probabilística para los 10 dígitos usando colores negro, gris, blanco modelados por un archivo CSV.

```
1 modelos <- read.csv("modelos2.csv", sep=" ", header=FALSE, stringsAsFactors=F)
```

Las probabilidades para el color del pixeles negros de 0,995 y 0,8, los blancos 0,001 y 0,1, finalmente para los grises de 0,75 y 0,65. Finalmente con referencia en el código [2], se paralelizó la etapa de la prueba llamando a las funciones a los núcleos por medio del paquete *parSapply* y se realizaron 30 réplicas:

```
1 clusterExport(cluster, c( "neuronas" , "binario" , "decimal" , "modelos" , "tope" , "k" ,
2                                "dim" , "n" ))
3 contadores <- parSapply(cluster, 1:prueba , function(x){
4   d <- sample(0:tope, 1)
5   pixeles <- runif(dim) < modelos[d + 1,]
6   correcto <- binario(d, n)
7   salida <- rep(FALSE, n)
8   for (i in 1:n) {
9     w <- neuronas[i ,]
10    deseada <- correcto [i]
```

```

11     resultado <- sum(w * pixeles) >= 0
12     salida [ i ] <- resultado
13   }
14   r <- min(decimal(salida , n) , k)
15   return(r==d))
16 datos<-rbind(datos , data.frame(Replica= replicas , PNegras=PN, PGrises=PG, PBlancas=PB,

```

## 4. Resultados

Los resultados de la efectividad de la red neuronal se midieron en base al % de contadores entre el número de pruebas para los distintos casos de probabilidad de los píxeles, para así notar que tan precisa es la red neuronal en base al modelo proporcionado (los dígitos).

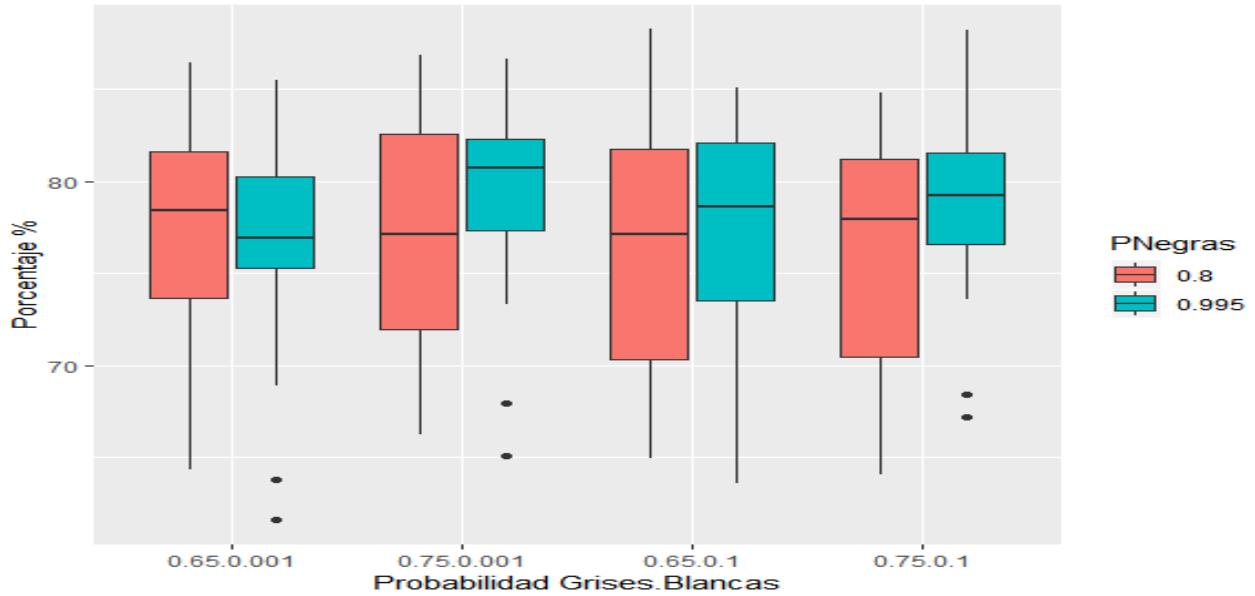


Figura 1: Porcentaje de precisión de la red neuronal con distintas probabilidades

## 5. Conclusiones

Se puede concluir que la mayoría de aciertos se presentó para las probabilidades de 0,995 hablando de los píxeles negros, 0,75 al referirnos a los píxeles grises y 0,001 en el caso de los píxeles blancos. Probablemente debido a que la mayoría de los píxeles de los modelos eran negros o grises y en menor medida blancos, por eso se encontraron favorecidas las probabilidades altas en los blancos y grises, y la menor en los blancos. Se pueden hacer pruebas posteriores con distintas iteraciones y modelos para corroborar estas afirmaciones.

## 6. Reto 1

Se aumentan el número de en el mapa de píxeles en el mapa a uno de  $5 \times 7$ , además se agregan 12 nuevos caracteres ASCII con el código:

```

1 modelos <- read.csv("reto1.csv", sep="~", header=FALSE, stringsAsFactors=F)
2 modelos[modelos=='n'] <- 0.995 # pixeles negros en plantillas
3 modelos[modelos=='g'] <- 0.92 # pixeles grises en plantillas
4 modelos[modelos=='b'] <- 0.002 # pixeles blancos en plantillas
5
6 r <- 7
7 c <- 5
8 dim <- r * c
9
10 n <- 49
11 w <- ceiling(sqrt(n))
12 h <- ceiling(n / w)
13
14 letras <- c(0:9, "L", "I", "H", "C", "T", "E", "F", "U", "P", "A", "W", "M")

```

La cual obtuvo la plantilla de la figura 2, con un tiempo de mil repeticiones en el entrenamiento aún aún se observan discrepancias con las figuras.

## Referencias

- [1] Alonso Avellano. Redes neuronales. Página Web: Redes Neuronales, 2016. URL <http://avellano.fis.usal.es/~lalonso/RNA/index.htm>.
- [2] Liliana Saus. Redes neuronales. Repositorio GitHub, 2018. URL <https://github.com/pejli/simulacion/tree/master/P12>.
- [3] Elisa Schaeffer. Práctica 12: Redes neuronales. Página Web: R Paralelo, 2019. URL <https://elisa.dyndns-web.com/teaching/comp/par/p12.html>.

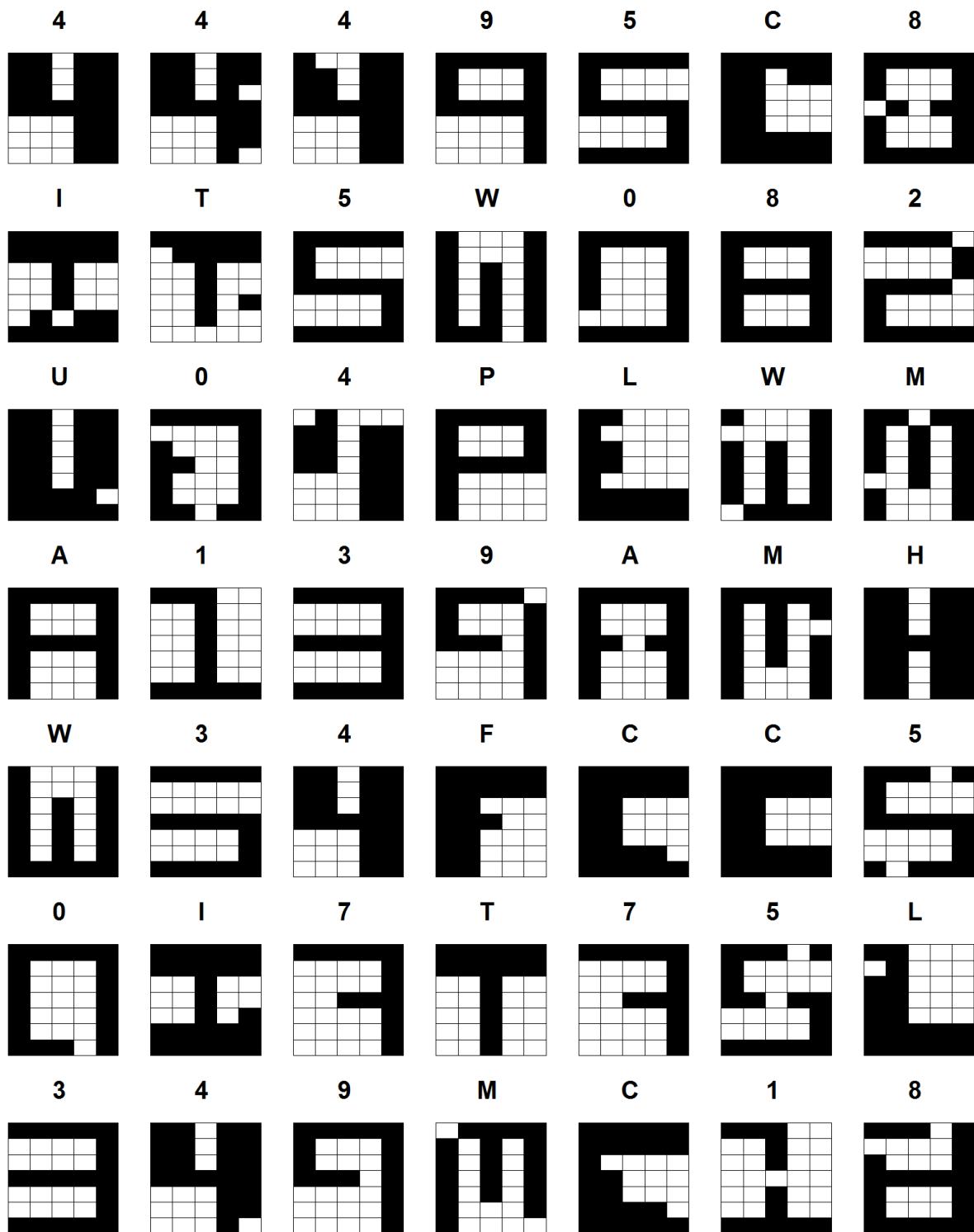


Figura 2: Plantilla resultante

# Proyecto Integrador

## Revisión y Retroalimentación

- Se corrigieron los formatos de las bibliografías.
- Se sustituyeron textos plagiados, es importante basarse en trabajos anteriores, sin embargo, evitar copiarlos.
- Se agregaron resultados y conclusiones.

3/35

# Propuesta de Simulación de Secuencia de ARN e Identificación de Codones Responsables de la Síntesis de la Cisteína

José Ángel García Cedillo

4 de junio de 2019

## Resumen

Las enfermedades neurodegenerativas como la de Huntington están directamente relacionadas con la expresión o mutación de los codones que dan origen a los aminoácidos cisteína y glutamina. Diversos factores ocasionan daños en la estructura del ARN que promueven las malas traducciones o **Algunos Debido** a ésto es necesario implementar una herramienta que ayude a procesar los datos de secuencia genética para determinar de manera estadística

Palabras clave: Simulación, ARN, Codones, Farmacogenética, Código *R*.

## 1. Introducción

Los genes como una base de datos contiene toda la información que nos forma y describe. Los genes están formados por el ARN (Ácido Ribonucleico) y el ADN (Ácido desoxirribonucleico) el cual está encargado de las labores de síntesis de proteínas [9]. El mapa genético incluso nos puede decir lo proclives que somos a padecer ciertas afecciones, físicas o mentales [6]; se conoce una estrecha relación entre las enfermedades físicas y mentales, se conocen variedad de enfermedades médicas que dan como resultado trastornos mentales claramente identificados entre éstas predominan las endocrinológicas, las cardiopatías, las **inmunológicas** y las **neuroológicas** [5]. Actualmente México, ~~una~~ una de cada cinco personas que enfrenta un problema de salud mental degenerativa o emocional tiene acceso a atención especializada [7] por lo que el país tiene un rezago del 80 por ciento ~~en este ámbito~~ ~~en~~ ~~que~~ ~~lo~~ ~~convierte~~ ~~en~~ ~~un~~ ~~problema~~ ~~real~~ ~~que~~ ~~salta~~ ~~a~~ ~~la~~

~~vista y que si se deja sin atender puede empeorar.~~  
En México se estima que 9.2 % de la población ha sufrido enfermedad mentales degenerativas, y que una de cada cinco personas las sufrirá antes de los 75 años [2]. Como antes se mencionó, las desordenes mentales muchas veces son precedidas de anomalías físicas las cuales las desarrollan, ésto es por el desequilibrio neuroquímico o que presentan los pacientes con **esquizofrenia** y **depresión** por decir algunos ejemplos, estas dos enfermedades están relacionadas con la producción anormal de neurotransmisores catecolaminas: dopamina, norepinefrina y epinefrina [4]. Otra de las importantes causas del desarrollo de enfermedades neuronales son las mutaciones al código genético como lo es la enfermedad de Huntington, un trastorno hereditario, que provoca daño debido a la codificación anormal de ADN para el aminoácido glutamina. Las personas sanas tienen entre 15 y 20 "repeticiones" de ADN en esa parte de su código genético, mientras que los portadores del gen de la enfermedad de Hun-

tington poseen más de 36 o incluso hasta 100 [8]. Además se ha ligado a el desarrollo de la enfermedad de Hungtinton, la deficiencia del aminoácido cisteína ya que se han encontrado muy pocos niveles de la enzima para la biosíntesis para la cisteína en tejidos de pacientes con la enfermedad [3].

## 2. Antecedentes

El ARN fue descubierto junto al ADN en 1867, por *Friedrich Miescher*, quien los llamó *nucleína* y los aisló del núcleo celular. El modo de síntesis del ARN en la célula fue descubierto posteriormente por el español *Severo Ochoa Albornoz*, ganador del Premio Nobel en Medicina [9] [9]. El ADN es una macromolécula formada por unidades denominadas nucleótidos o nucleótidos que forma el ADN sólo pueden ser cuatro: A (adenina), T (timina), C (citosina) y G (guanina). Para que esta información pueda ser utilizada por las células se transcribirá a una molécula de ARN (ácido ribonucleico). La molécula de ARN se copia fielmente a partir de la molécula de ADN en un proceso denominado transcripción. Existen diferencias químicas entre las moléculas que componen el ADN y el ARN, además el código que difiere ya que la T del ADN es reemplazada por uracilo ("U") en el ARN. Se han reportado trabajos en donde se expone el daño a código genético por mutaciones, entre las que destacan los cambios en la leucina, la contaminación, la oxidación y las drogas farmacológicas para tratar enfermedades [10] llegando a provocar problemas secundarios como obesidad provocados debido a los mismos tratamientos farmacológicos. [1].

## 3. Solución Propuesta

Debido a que ciertas enfermedades neurodegenerativas son causadas por mutaciones en el código genético que llevan a trastornos de sobre expresión o deficiencia de aminoácidos por parte de los codones, es importante simular el RNA de cierta población para crear una herramienta que pueda proce-

sar buscando los codones o la deficiencia de estos.

## 4. Metodología

Mediante el código se definen los bloques básicos, los nucleótidos como A,U,G,C; uracilo en vez de timina ya que se usa el RNA pues el codón que codifica para cistina es UGC. Despues se crea la secuencia con mil bases nitrogenadas alazar y finalmente se crea una función para buscar cuantas veces se presenta el codón.

```
library(parallel)
bases = c("A", "U", "G", "C")
codonCist = c("UGC")
repeticiones = 30
poblacion = 10
nchar = 17

cluster <- makeCluster(detectCores())
cluster[1]
clusterExport(cluster, "bases")
clusterExport(cluster, "codonesCist")
clusterExport(cluster, "poblacion")
Data <- data.frame()
clusterExport(cluster, "minimo")
for (minimo in 1:min) {
  cuantosDist <- 0
  Resultados <- parSapply(cluster, 1:poblacion, function(r) {
    secuencia <- paste(sample(bases, 1000,
      replace = TRUE), collapse = ""))
    enfermos <- 0 #mayor <- 0
    cuantosCist <- 0
    for (minimo in 1:min) {
      for (i in 1:(nchar(secuencia) - 2)) {
        (codon = substring(secuencia, i, i +
```

```

26
27         2))
28     (esUGC = codon %in %
29         codonesCist)
30     if(esUGC){
31         cuantosCist = (
32             cuantosCist + 1)
33         }
34         return((
35             cuantosCist
36             ))
37     if(cuantosCist > 34){
38         enfermos = (enfermos +
39             1)}
40     }
41
42 Datos<- rbind (Datos , Resultados)
43
44 print(Datos)
45 plot(Datos)

```

## 5. Resultados

No se encontró el modelo para contar con la herramienta, las gráficas arrojan datos confusos.

## 6. Conclusiones

Se debe seguir trabajando para desarrollar el contador de codones de manera eficaz.

## 7. Trabajo a Futuro

Utilizar de manera inversa para identificar el codón para glutamina por encima de 36 repeticiones, indicaría un síntoma de padecer Huntington. Además existen muchos paquetes para procesar este tipo de datos de manera más elegante, se buscará aprender a usarlos.

## Referencias

- [1] Carlos Prieto Jesus Sainz Benedicto Crespo-Facorro. "Altered gene expression in antipsychotic-induced weight gain". Journal: npj Schizophrenia, Abril 2019. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656420/>. *Corina Benjet. "día mundial para la prevención del suicidio. sin salud mental no hay salud". Artículo web: Agencia Informativa Conacyt, 2015. URL <http://conacytprens.mx/index.php/ciencia/salud/2913-dia-internacional-para-la-prevencion-del-suicidio>*
- [2] Risheng Xu M. Scott Vander Jagyong Y. Cha Adele M. Snowman Solomon H. Snyder Bindu D. Paul, Juan I. Soodio. "cystathione -lyase deficiency mediates neurodegeneration in huntington's disease". Nature N509, 96-100 p, March 2014. URL <https://www.genome.gov/For-Patients-and-Families/Genetic-Disorders>.
- [3] Marco Castellani "el desequilibrio neuroquímico de la esquizofrenia". Artículo en la web: Infosalus, 2014. URL <https://www.infosalus.com/salud-investigacion/noticia-desequilibrio-neuroquimico-esquizofrenia.html>.
- [4] Lic. María Paz Richard Mañoz Dr. Juan Martín Sandoval De Escorcia. "la salud mental en México". Reporte del Servicio de Investigación y Análisis, Cámara de Diputados, 2005. URL <http://www.salud.gob.mx/unidades/cdi/documentos/SaludMentalMexico.pdf>.
- [5] National Human Genome Research Institute. "genetic disorders". Página Web: Genetic Disorders, 2018. URL <https://www.genome.gov/For-Patients-and-Families/Genetic-Disorders>.
- [6] Arturo Sánchez Jiménez. "méxico, con rezago en atención de la salud mental del

- 80 %: Inprf'. Artículo en la Web: Periódico La Jornada, Febrero 2019. URL <https://www.jornada.com.mx/ultimas/2019/02/26/en-mexico-1-de-cada-5-con-problemas-mentales-recibe-atencion-inprf-670.html>.
- [8] Daniela Montesarchio Antonio Giordano Concetta Giancola Mariarosa A.B.Melone Marco Caterino, Tiziana Squillaro. "huntingtin protein: A new option for fixing the huntington's disease countdown clock". Journal of Neuropharmacology, Volume 135, Junio 2018. URL <https://doi.org/10.1016/j.neuropharm.2018.03.009>.
- [9] María Estela Raffino. ".arn". Página Web: Concepto.de, 2018. URL <https://concepto.de/arn/>.
- [10] Elisabeth J. Wurtmann and Sandra L. Wolin. rna under attack: Cellular handling of rna damage". Critical Reviews in Biochemistry and Molecular Biology, 2009, 44(1), 34–49pp. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656420/>.

# Propuesta de Simulación: secuencia de ARN e Identificación de codones para el diagnóstico de enfermedad de Huntington y su relación con la calidad del aire.

José Ángel García Cedillo

10 de junio de 2019

---

## Resumen

El ADN comprende toda la información que nos determina muchas de las características que nos describen, incluso, nuestra salud. Por eso muchas enfermedades tienen trasfondo genético, es decir, suceden por afectaciones a nuestro código genético. Los daños oxidativos que sufren los ácidos nucleicos son causados principalmente por la contaminación ambiental a la que estamos sometidos. En éste trabajo se simula poblaciones de secuencias genéticas en las cuales se buscan codones específicos para determinar un posible diagnóstico de enfermedad, en el caso específico: “enfermedad de Huntington”. A la simulación se le agregó un factor discreto de contaminación ambiental en base a los valores reales de algunos municipios de la zona metropolitana de Monterrey para reflejar el daño oxidativo que ésta le provoca.

Palabras clave: Simulación, ARN, Codones, Calidad del aire, Código *R*.

---

## 1. Introducción

La información genética en los seres vivos está contenida en las moléculas de ácido desoxirribonucleico *ADN* y ácido ribonucleico *ARN* por sus siglas, las cuales están formadas por unidades denominadas nucleótidos. Los nucleótidos que forman estas macromoléculas sólo pueden ser: A (adenina), T (timina), C (citosina) o G (guanina) y en el caso del *ARN* la timina está sustituida por el U (uracilo). La molécula de *ARN* tiene la función de copiar al *ADN* en un proceso llamado transcripción [1]. Durante este proceso el *ARN* mensajero lleva la información para la síntesis de proteínas, es decir, determina el orden en que se unirán los aminoáci-

dos. Ésta información está codificada en forma de tres bases nitrogenadas, llamados también codones y son los encargados de que determinar la síntesis de un aminoácidos que posteriormente serán proteínas [6]. La contaminación del aire puede es actualmente el mayor riesgo ambiental para la salud del mundo, de acuerdo a la Asociación Mundial de la Salud *WHO* [7]. Es especialmente dañino someterse a la exposición al aire con partículas finas con diámetro mayor a 2.5 micras y de hecho es el quinto factor mayor de riesgo de muerte en el mundo, debido a que causa 4.2 millones de muertes [3] pues está comprobado que daña el *ADN*. Los gases más comunes que presenta la atmósfera contaminada incluyen dióxido de sulfuro, óxido nítrico, dióxido

de nitrógeno, hidrocarburos reactivos (compuestos orgánicos volátiles) y monóxido de carbono. [4]. La contaminación, es pues uno de los factores mas importantes que dañan y causan mutación en el código genético.

## 2. Antecedentes

Se han reportado trabajos en donde se expone el daño al código genético por diversas razones, entre las que destacan los rayos ultravioleta, la contaminación, la oxidación [2]. Concretamente la contaminación ambiental afecta directamente al genoma del ser humano y al animal por medio de un mecanismo de oxidación [2] y se ha podido demostrar la cercana relación que guarda el respirar partículas suspendidas en el aire menores a 25 micras con las muertes provocadas por enfermedades respiratorias, diabetes, desórdenes cardiovasculares y cáncer [10]. El mecanismo oxidativo que rompe al ADN, ha sido estudiado en personas que están muy expuestas al ambiente, como los policías controladores de tráfico vehicular y se ha comprobado que crea alteraciones en su composición, el daño resulta ser cumulativo y puede ser uno de los primeros pasos hacia desarrollar cáncer [9]. El índice de Calidad del Aire *ICA* es un indicador que muestra que tan contaminado está el aire de una zona en concreto y sus posibles afecciones a la salud. Se fundamenta en la norma Ambiental (NADF-009-AIRE-2017) [5] y en ella se detallan los requisitos para calcularlo y la difusión de *ICA*. Éste toma en cuenta cinco de los contaminantes del aire: dióxido de azufre, monóxido de carbono, dióxido de nitrógeno, ozono y partículas suspendidas, maneja una escala de 0 a 500 en donde el 100 es el que dicte la norma oficial mexicana.

A continuación se muestra un mapa detallando los valores *ICA*, separados por región presentados el día 10 de Junio a las 8 am (figura1).

La enfermedad de Huntington es un desorden autosomal dominante causado por la expansión de una cadena de poliglutamina de la proteína huntingtina. Otras enfermedades neuronales son provocadas

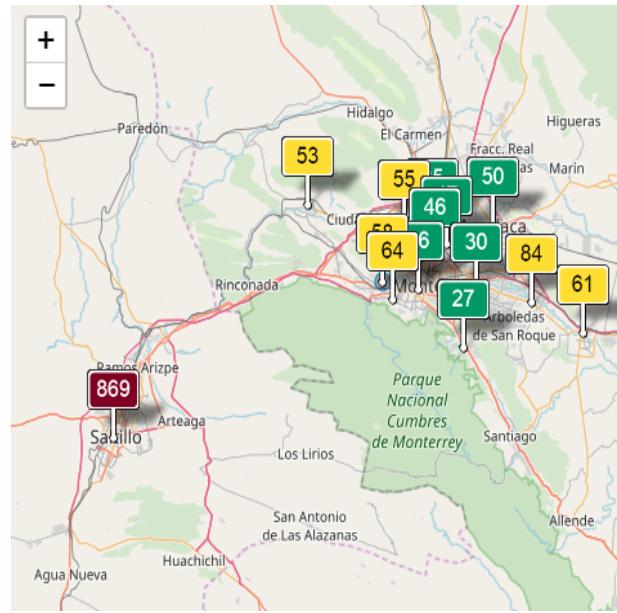


Figura 1: Mapa de emisiones tomado de World Air Quality Index: <https://waqi.info/es>

también por el alargamiento de las regiones de poliglutamina [11]. El codón de transcripción para glutamina es el de “CAG” y está identificado que se repite al menos 36 veces dentro de el gen de la huntingtina, 10% de los que la padecen heredaron de un parente que no se vio afectado por la expresión de las 36 “CAG” repeticiones [8].

## 3. Solución Propuesta

Debido a la situación precaria de la calidad del aire actual de la zona metropolitana de Monterrey y que ciertas enfermedades son causadas por mutaciones en el código genético y llevan a trastornos de sobre expresión o deficiencia de aminoácidos por parte de traducción de codones, se propone que se simulen poblaciones locales tomando como ejemplo la enfermedad de Huntington y mediante una herramienta de identificación de triadas de bases nitrogenadas (codones) se relacionen las probables repercusiones que tiene la contaminación en la salud.

## 4. Metodología

Mediante el código se definen los bloques básicos, los nucleótidos como **A,U,G,C**; uracilo en vez de timina ya que el RNA se encarga de sintetizar los aminoácido y el codón que codifica para glutamina relacionado con la sobre expresión de glutamina es el **CAG**. Después se crea la secuencia de bases nitrogenadas al azar y finalmente se crea un substring para buscar cuantas veces se presenta el codón. En caso de presentarse más de 36 veces [8] el sujeto es considerado como enfermo y entra a un contador.

```

1 resultadoP <- parSapply(cluster , 1:
2   repeticiones ,
3     function(r) {
4       cuantosglut = 0
5       enfermos = 0
6       for (p in 1:poblacion) { #mod
7         secuencia <- paste0(sample(bases ,
8           Locacion , replace = TRUE),
9           collapse = ""))
10      for(i in 1:(nchar(secuencia) - 2))
11        {
12          codon = substring(secuencia , i , i
13            + 2))
14          esUGU = codon %n% codonGlut)
15          if(esUGU){
16            cuantosGlut = (cuantosGlut + 1)
17          }
18        }
19        return(cuantosGlut)
20      if (cuantosGlut > 36){
21        enfermos= enfermos + 1}
22      return (enfermos)
23    }
24  )
25  }
26  
```

Finalmente se paraleliza llamando a los clusters usando la paquetería parallel y simulando 5000 sujetos como población, efectuando 30 repeticiones para que sea una muestra significativa. Se tomaron en cuenta los índices de calidad de aire de los municipios de la zona metropolitana de Monterrey en una proporción de  $ICA \times 0,1$  al agregar a la base de mil bases nitrogenadas de la secuencia para aumentar la probabilidad de presentarse enfermo simulando así el daño oxidativo de la contaminación en el código genético.

```

1 library(parallel)
2
3 bases = c("A" , "U" , "G" , "C")
4 codonGlut = c("CAA" , "CAG")
5 locacion= c(1077, 1054, 1068, 1106, 1042,
6           1064,1084, 1058, 1864)
7 repeticiones= 30
8 poblacion= 5000
9
10 cluster <- makeCluster(detectCores() - 1)
11 clusterExport(cluster , "poblacion")
12 clusterExport(cluster , "repeticiones")
13 clusterExport(cluster , "bases")
14 clusterExport(cluster , "codonGlut")
15 clusterExport(cluster , "cuantosGlut")
16
17 pdatos <- data.frame()
18
19 for (Locacion in locacion ) {
20   clusterExport(cluster , "Locacion")
```

## 5. Resultados

Los resultados muestran la cantidad de enfermos por municipio simulado, así como su porcentaje con respecto al total de la población (figuras 2 y 3).

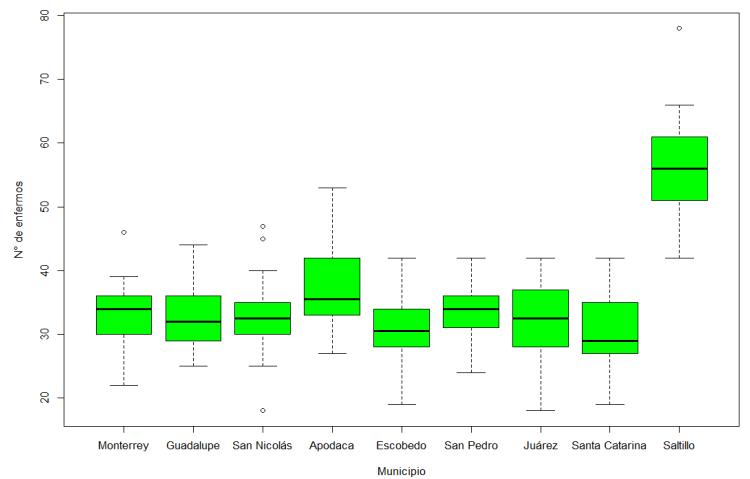


Figura 2: Resultados de simulación de cantidad de enfermos de Huntington por municipio

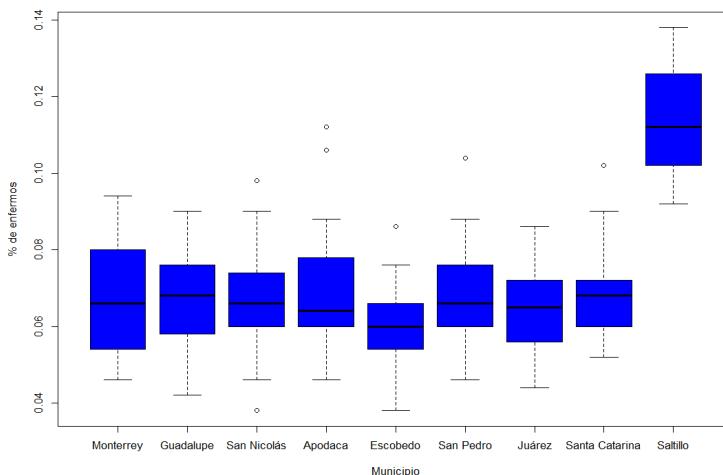


Figura 3: Resultados de simulación del porcentaje de enfermos de Huntington por municipio

Se realiza la prueba ANOVA y la gráfica de los residuales para comprobar que los resultados están normalmente distribuidos (figura 4)

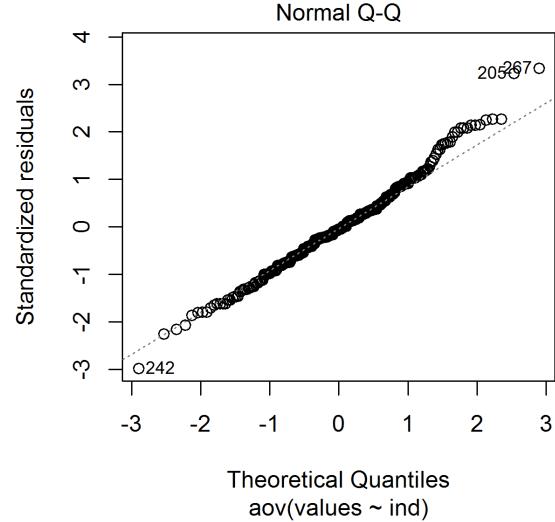


Figura 4: gráfica de normalidad de resultados

Sobre los resultados de los enfermos, especialmente en la excentricidad que mostraba el índice de Saltillo.

## 7. Trabajo a Futuro

El trabajo a desarrollar para empezar a formar una herramienta mas completa incluyendo en la simulación factores y datos reales, como cantidad de habitantes por municipio, cantidad de bases nitrogenadas reales involucradas en la sobre expresión de glutamina, agregar un factor de mutación derivado de estudios mas extensos para que sea mas real la simulación. Además, aprender a usar y utilizar muchos de los paquetes bioinformáticos disponibles para R.

Debido a que el valor de Fisher es por debajo de 0,5 la hipótesis nula, la cual estima que los resultados entre los tratamientos, en éste caso, no tienen nada de relación entre sí se rechaza.

## 6. Conclusiones

En base a los resultados se puede concluir que la hipótesis de que simulación fue congruente con el nivel contaminación del ambiente. Usando como factor la cantidad de bases nitrogenadas a más asignadas se aumenta la probabilidad de presentar más repeticiones de codones para glutamina, que traspolado a los índices de calidad del aire reflejó de manera significativa la influencia que ésta tiene so-

## Referencias

- [1] *Translation is the RNA-directed synthesis of a polypeptide: A closer look*, chapter 1. Pearson, 2011.
- [2] Pavel Rossner Jr. Andrea Rossnerovaa Vlasta Svecovaa Alena Milcovaa Jana Pulkrabovab Jana Hajslovab Milos Veleminsky Jr.c Ivo

- Solanskya Radim J. Srama Antonin Ambroza, Veronika Vlkovaa. Impact of air pollution on oxidative dna damage and lipidperoxidation in mothers and their newborns. *International Journal of Hygiene and Environmental Health*, 2016.
- [3] Burnett R Cohen AJ, Brauer M. Estimates and 25-year trends of the global burden of disease attributable to ambient air pollution. Technical report, The Global Burden of Diseases, 2015.
- [4] Clayton T. Cowl Dean E. Schraufnagel, John R. Balmes. Air pollution and noncommunicable diseases. Technical report, Forum of International Respiratory Societies Environmental, 2019.
- [5] Secretaría del Medio Ambiente. Gaceta oficial de la ciudad de méxico. Technical report, Secretaría del Medio Ambiente, 2018.
- [6] Carlos González. Traducción: Síntesis de proteínas. Página web: Gabinete de botánica de la Universidad Autónoma de Buenos Aires, Página Web consultada el 10 de Junio 2019.  
URL <http://www.botanica.cnba.uba.ar/Pakete/3er/LosCompuestosOrganicos/1111/Traducion.htm>.
- [7] World Health Organization. Ambient air pollution. Technical report, Geneva, Switzerland: World Health Organization, 2016.
- [8] Anna R. Hayden Jeffrey B. Carroll Stefanie L. Butland Simon C. Warby, Alexandre Montpetit. Cag expansion in the huntington disease gene is associated with a specific and targetable predisposing haplogroup. *The American Journal of Human Genetics*, (84):351–366, 2009.
- [9] Wang Yupeng Zhu Yan Shi Ting Lin Mingyue Deng Zhonghua Wang Zhua Tan Chaochao, Lu Shijie a. Long-term exposure to high air pollution induces cumulative dna damages in traffic policemen. *Science of the Total Environment*, 2017.
- [10] Tsang H. Lai H.K. Thomas G.N. Lam K.B. Chan K.P. Wong, C.M. Cancer mortality risks from long-term exposure to ambient fine particle. *Cancer Epidemiology Biomark*, 2016.
- [11] Brusilow W.S. Is huntington's a glutamine storage disease? *Neuroscientist*, 2006.
- [12] Elisabeth J. Wurtmann and Sandra L. Wolin. Rna under attack: Cellular handling of rna damage. *Critical Reviews in Biochemistry and Molecular Biology*, 2009.