UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

MAESTRÍA EN CIENCIAS DE LA INGENIERIA CON ORIENTACIÓN EN NANOTECNOLOGÍA

PORTAFOLIO DE EVIDENCIA

SEMESTRE (FEBRERO-JUNIO 2021)

SIMULACIÓN COMPUTACIONAL

PROFESORA: SATU ELISA SCHAEFFER

ALUMNO: JOSÉ ADRIAN GARCIA FUENTES

MATRICULA: 1551575

CALIFICACIÓN TOTAL: 80

https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES

14/JUNIO/2021

Práctica Nº 1
Movimiento Browniano

Nombre: José Adrián García Fuentes **Profesor:** Satu Elisa Schaeffer
Fecha: 13/Febrero/2021

---

1. Objetivo

- Examinar de manera sistemática los efectos de la dimensión en el tiempo de regreso al origen del movimiento browniano para dimensiones 1 a 5 en incrementos lineales de uno, variando el número de pasos de la caminata como potencias de dos con exponente de 4 a 9 en incrementos lineales de uno, con 30 repeticiones del experimento para cada combinación.[1]
- Graficar los resultados en una sola figura con diagramas de caja bigote.[1]
- Incluir un cuadro indicando el mínimo, promedio y máximo del tiempo de regreso por cada dimensión junto con el porcentaje de caminatas que nunca regresaron.[1]

2. Metodología

La metodología empleada se realizó a través de RStudio[2] llevando a cabo los pasos señalados en la *Práctica 1 Movimiento Browniano*,[1] se compararon los resultados con secuencias paralelas[3] encontradas en el repositorio de github,[4] se tomaron los repositorios caminata.r[5] y distance.r[6] para determinar el tipo de caminata y la distancia recorrida en el experimento, el código completo de la metodología empleada se encuentra en el repositorio[7] del autor.

3. Resultados

Se obtuvo el código secuencia de los efectos de la dimensión en el tiempo de regreso al origen del movimiento browniano para dimensiones 1 a 5 en incrementos lineales de 1, variando el numero de pasos, con 30 repeticiones. A continuación se muestra parte del código del experimento[7] en el cual se señala el número de repeticiones y parte de la función dada solicitando numeros de manera pseudoaleatoria que determinaran la posición final de nuestro punto.

```
repetir<- 30
experimento<- function(dim, dur){
  pos<- rep(0,dim)
  for(t in 1:dur){
    d<- sample(1:dim, 1)
    if(runif(1) < 0.5){
       pos[d]<- pos[d] - 1
    } else {
       pos[d]<- pos[d] + 1
    }
    if (all(pos == 0)){
      return(t)
    }
```

```
  }
  return(-1)
}
```

Con la finalidad de encontrar la distancia de una posición desde el origen. Las dos opciones comunes son la distancia Manhattan que mide la suma de los valores absolutos de las coordenadas y la distancia Euclideana que mide el largo del segmento de línea que conecta el origen al punto en cuestión,[1] para comparar el tipo de distancia se llamaron las rutinas de caminata.r[5] y distance.r[6] en la Fig.1 se muestra el diagrama caja bigote con distancia Manhattan de las 5 dimensiones.

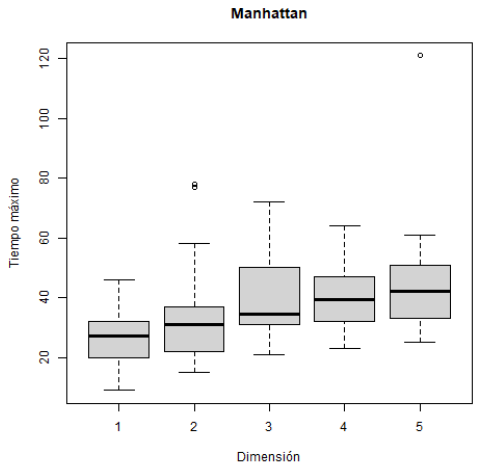


Figura 1: Diagrama caja bigote.

Para un mayor entendimiento se determino el mínimo, promedio y máximo del tiempo de regreso por cada dimensión junto con el porcentaje de caminatas que nunca regresaron al origen los datos obtenidos se encuentran tabulados en el Cuadro 1.

Cuadro 1: Indica el mínimo, promedio y máximo del tiempo de regreso por cada dimensión junto con el porcentaje de caminatas que nunca regresaron.

Dimension	Mínimo	Promedio	Máximo	Porcentaje
1	2.0	8.5	10.0	37.5 %
1	2.0	30.01	32.0	0 %
1	2.0	27.3	44.0	31.25 %
1	2.000	18.9	74.000	42.1875 %
1	2.000	25.03	116.000	54.6875 %
1	2.0	45.02	256.0	50 %
2	2.000	15.12	16.000	0 %
2	2.000	8.3	8.000	75 %
2	2.00	53.12	64.00	0 %
2	2.00	59.135	118.00	7.8125 %
2	2.000	17.3	30.000	88.28125 %
2	2.0	51.15	66.0	87.109375 %
3	2.000	6.365	10.000	37.5 %
3	2.000	7.532	8.000	75 %
3	2.00	39.78	42.00	34.375 %
3	2.000	5.432	8.000	93.75 %
3	2.00	60.465	68.00	73.4375 %
3	2.0	72.03	82.0	83.984375 %
4	2	58.3	6	62.5 %
4	2.000	9.45	10.000	68.75 %
4	2	2.35	2	96.875 %
4	2.0	4.12	4.0	96.875 %
4	2	12.16	10	96.09375 %
4	2.0	31.19	32.0	93.75 %
5	4	32.17	4	75 %
5	2	17.8	2	93.75 %
5	2.0	32.3	4.0	93.75 %
5	2	64.17	8	89.76 %
5	2.000	80	10.000	96.09375 %
5	2	16	2	99.609375 %

4. Conclusión

Se demostró de manera sistemática los efectos de la dimensión en el tiempo de regreso al origen del movimiento de nuestra partícula o punto obteniendo como resultado que estas tardaran más en regresar al punto de origen o regresaran en una menor proporción a medida que la dimensión cambia.

5. Retos

Reto 1: Estudiar de forma sistemática y automatizada el tiempo de ejecución de una caminata (en milisegundos) en términos del largo de la caminata (en pasos) y la dimensión. Para medir el tiempo de una réplica, ejecútala múltiples veces y normaliza con la cantidad de repeticiones para obtener un promedio del tiempo de una réplica individual.[1]

Reto 2: Realizar una comparación entre una implementación paralela y otra versión que no aproveche paralelismo en términos del tiempo de ejecución, aplicando alguna prueba estadística adecuada para determinar si la diferencia es significativa.[1]

Referencias

[1] E. Schaeffer, "Práctica 1: Movimiento browniano," Febrero 2021. https://elisa.dyndns-web.com/teaching/comp/par/p1.html.

[2] J. J. Allaire, "Rstudio," Febrero 2021. https://rstudio.com.

[3] E. Schaeffer, "Parallel.r," Febrero 2021. https://github.com/satuelisa/Simulation/blob/master/BrownianMotion/parallel.R.

[4] E. Schaeffer, "brownian motion," Febrero 2021. https://github.com/satuelisa/Simulation/tree/master/BrownianMotion.

[5] E. Schaeffer, "caminata.r," Febrero 2021. https://github.com/satuelisa/Simulation/blob/master/BrownianMotion/caminata.R.

[6] E. Schaeffer, "distance.r," Febrero 2021. https://github.com/satuelisa/Simulation/blob/master/BrownianMotion/distance.R.

[7] J. A. Garcia Febrero 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%201.

Práctica Nº 2
Autómata Celular

Nombre: José Adrián García Fuentes **Profesor:** Satu Elisa Schaeffer
Fecha: 23/Febrero/2021

---

1. Objetivo

- Diseñar y ejecutar un experimento para determinar el efecto de la regla de supervivencia (por lo menos cinco reglas) en la vida de la colonia en una malla de 12 por 12 celdas hasta que se mueran todas o que se hayan cumplido 30 iteraciones, teniendo cada celda o viva o muerta con la misma probabilidad al inicio [1].
- Graficar y tabular los hallazgos [1].

2. Metodología

La metodología empleada se realizó a través de RStudio [2] llevando a cabo los pasos señalados en la *Práctica 2 Autómata Celular* [1], la secuencia realizada de forma paralela fue basada en el código en R [3] encontrada en el repositorio de GitHub [4] para diseñar y ejecutar el experimento se estableció una colonia a la que llamaremos nido dentro del código con dimensiones de 12 por 12 celdas, se generó un punto de origen con una rutina en donde intervienen los números enteros más cercanos quienes determinara si nuestro origen está vivo o muerto en nuestro caso se acopló esta regla imaginando a un ave fénix por tanto se determinara si está hecho cenizas o se encuentra vivo, sin embargo, para la supervivencia se efectuaron distintas reglas que dependerán del número de vecinos vivos, el código completo de la metodología empleada se encuentra en el repositorio [5] del autor.

3. Reglas de supervivencia

Las 5 reglas de supervivencia aplicadas son las siguientes:

- Nuestro punto se encontrara vivo siempre y cuando 6 de sus vecinos se encuentre vivo.
- Nuestro punto se encontrara vivo siempre y cuando 3 de sus vecinos se encuentre vivo.
- Nuestro punto se encontrara vivo siempre y cuando 1 de sus vecinos se encuentre vivo.
- Nuestro punto se encontrara vivo siempre y cuando 5 de sus vecinos se encuentre vivo.
- Nuestro punto se encontrara vivo siempre y cuando 2 de sus vecinos se encuentre vivo.

- Nuestro punto se encontrara vivo siempre y cuando 4 de sus vecinos se encuentre vivo.

Sin embargo para cada regla se deberá contar con un par de condiciones a cumplir de manera que el porcentaje de supervivencia cambiara.

- El número de vecinos tendrá que ser dado por una condición aleatoria.
- Si el número de sus vecinos es mayor a 5 cambiara la regla de supervivencia en una cantidad de vecinos entre 2 y 5
- Si el número de sus vecinos es menor a 5 cambiara la regla de supervivencia en una cantidad de vecinos entre 1 y 7

Por tanto las cantidades de números de vecinos serán de forma aleatoria y pueden reducirse o incrementar, para realizar una única regla de supervivencia podríamos revisar el código de GitHub [6] en donde se subió el código por separado con números que dicten un solo valor de supervivencia.

4. Resultados

A continuación se muestra parte del código realizado en RStudio [2] para ver el código completo consultar la base de datos de GitHub [3].

```
library(parallel)

terreno <- 12
hectareascuadradas <-  terreno^2
expansion<- terreno + 2
nido <- matrix(round(runif(hectareascuadradas)), nrow=terreno, ncol
    =terreno, byrow=TRUE)
suppressMessages(library("sna"))
png("practica2_t0_r.png")
plot.sociomatrix(nido, diaglab=FALSE, main="Inicio")
graphics.off()

Ano <- function(posicionvector) {
  fila <- floor((posicionvector - 1) / terreno) + 1
  columna <- ((posicionvector - 1) %% terreno) + 1
  vecinos <-  nido[max(fila - 1, 1) : min(fila + 1 , terreno),
                       max(columna - 1, 1): min(columna + 1, terreno
    )]
  return(1 *((sum(vecinos) - nido[fila, columna]) == sample(1:6,1))
    )
  if (((1 *((sum(vecinos) - nido[fila, columna]) == sample(1:6,1)))
    )>5) {(1 *((sum(vecinos) - nido[fila, columna]) == sample
    (1:6,1)))==(2:5)}
  else {(1 *((sum(vecinos) - nido[fila, columna]) == sample(1:6,1))
    )==((1 *((sum(vecinos) - nido[fila, columna]) == sample(1:7,1))
    )+1)}
  }
print(nido)
```

Los resultados obtenidos tras correr el código en RStudio [2] fueron convertidos a archivos .png, cada uno de estos representa la cantidad de vivos que se encuentran en cada interacción, las imágenes obtenidas fueron convertidas en un

archivo .gif [7] el cual se encuentra en el repositorio de GitHub [5], en la Fig. 1 se muestra el inicio de la secuencia experimental así como el paso final dado durante el experimento.

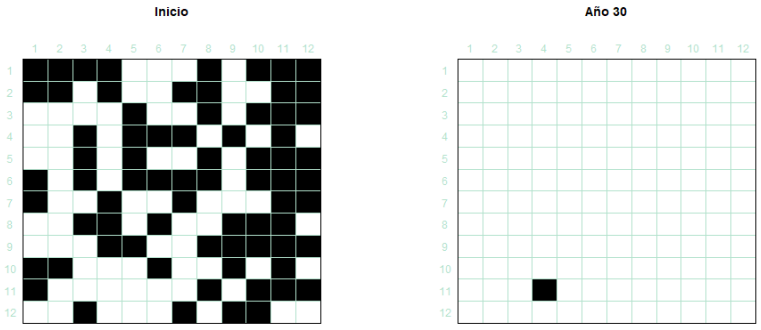


Figura 1:
Secuencia de pasos de autómata celular

De acuerdo a la interacción de pasos dados al cual llamaremos años se realizó un grafico que representa el numero de celdas vivas dependientes del paso de los años ver Fig. 2.

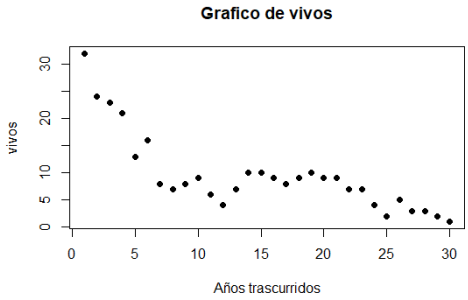


Figura 2:
Grafico representativo de vivos en el trascurso de los 30 pasos

5. Conclusión

Es de suma importancia comprender el diseño y ejecución de un experimento poniendo grados de complejidad, en este caso reglas de supervivencia con la finalidad de modelar un experimento de manera in situ antes de pasar al modo in vitro.

Referencias

[1] E. Schaeffer, "Práctica 2: autómata celular," Febrero 2021. https://elisa.dyndns-web.com/teaching/comp/par/p2.html.

[2] J. J. Allaire, "Rstudio," Febrero 2021. https://rstudio.com.

[3] E. Schaeffer, "gameoflife.r," Febrero 2021. https://github.com/fuentesadrian/Simulation/blob/master/CellularAutomata/gameOfLife.R.

[4] E. Schaeffer, "Cellulara utomata," Febrero 2021. https://github.com/fuentesadrian/Simulation/tree/master/CellularAutomata.

[5] J. A. Garcia Febrero 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%202.

[6] J. A. Garcia Febrero 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/blob/main/Tarea%202/codigoseparado.r.

[7] J. A. Garcia Febrero 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/blob/main/Tarea%202/automatacellular.gif.

Práctica 3: Teoría de colas

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer

Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica
16 de junio de 2021

1. Introducción

La teoría de colas es un área de las matemáticas que estudia el comportamiento de líneas de espera. Los trabajos que están esperando ejecución en un clúster esencialmente forman una línea de espera [1].

2. Objetivo

Examinar cómo las diferencias en los tiempos de ejecución de los diferentes ordenamientos cambian cuando se varía el número de núcleos asignados al clúster, utilizando como datos de entrada un vector que contiene primos grandes, descargados de https://primes.utm.edu/lists/small/millions/ y no primos (creados a partir de ellos). Con por lo menos nueve dígitos, aplicando pruebas estadísticas adecuadas y visualización científica clara e informativa [1].

3. Metodología

La metodología empleada se realizó a través de RStudio[2] llevando a cabo los pasos señalados en la *Práctica 3: teoría de colas* [1].

4. Resultados

Se obtuvo el código secuencia para determinar si un número es primo o no primo, a partir de este experimento se determinó el tiempo que tardaba en detectar si era primo o no primo. El código del experimento se encuentra en el repositorio de Garcia [3] en el cual se señala el número de repeticiones y parte de la función dada solicitando números de manera pseudoaleatoria, dicho código fue modificado del código del Schaeffer [4].

En la figura 1 se muestra un diagrama de violín de los tiempos de ejecución de cada orden (original, invertido y aleatorio) respectivamente, el tiempo en segundos es mayor para el orden original y menor para el invertido.

En el cuadro 1 se muestran los datos estadísticos de cada orden, se muestran valores muy similares en medias de los tiempos de ejecución, como también en el tiempo mínimo y una ligera diferencia en el tiempo máximo.

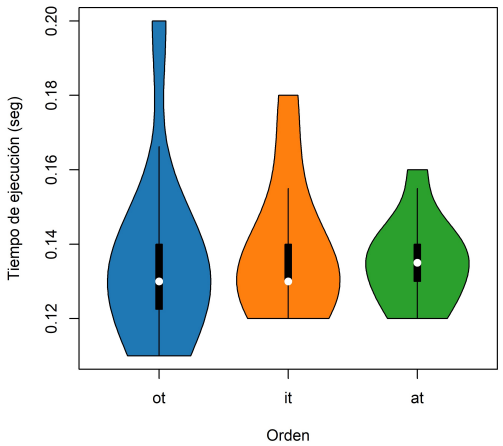


Figura 1: Tiempos de ejecución para cada orden.

5. Conclusión

En conclusión el tiempo en que tarda un experimento en correr una secuencia variando el número de

núcleos de un CPU puede ser más corto dependiendo de que tan largo sea el experimento para el caso de los tiempos de ejecución de la figura 1 el orden con mayor tiempo fue el original con un máximo de veinte segundos para el caso del orden aleatorio un máximo de dieciséis segundos, el código se encuentra en el repositorio de Garcia [3], sin embargo al realizar modificaciones del código se obtuvo un tiempo de ejecución mayor al esperado esto puede deberse a algún error en la asignación del número de núcleos o bien a la cantidad total procesada de números primos.

Cuadro 1: Descripción estadística

Orden	Mín	1er. Q	Mediana	Media	3er Q	Máx
Original	0,11	0,12	0,13	0,13	0,14	0,20
Invertido	0,12	0,13	0,13	0,13	0,14	0,18
Aleatorio	0,12	0,13	0,13	0,13	0,14	0,16

Referencias

[1] E. Schaeffer, "Práctica 3: teoría de colas," junio 2021. https://elisa.dyndns-web.com/teaching/comp/par/p3.html.

[2] J. J. Allaire, "Rstudio," junio 2021. https://rstudio.com.

[3] J. A. Garcia, junio 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%2012.

[4] E. Schaeffer, "Práctica 12: red neuronal," junio 2021. https://github.com/satuelisa/Simulation/tree/master/NeuralNetwork.

Práctica Nº 4
Diagramas de Voronoi

Nombre: José Adrián García Fuentes **Profesor:** Satu Elisa Schaeffer
Fecha: 05/Marzo/2021

---

1. Introducción

Los diagramas de Voronoi son un método comúnmente utilizado para realizar interpolaciones simples. Esta práctica aborda la simulación del crecimiento de un número determinado de semillas sobre una matriz. El material se encuentra dividido en regiones conocidas como celdas de Voronoi que se asemejan a núcleos formados por un proceso de cristalización del material. Si representamos el material con una cuadrícula bidimensional, una celda de Voronoi está formada por las casillas más próximas a cada una de las k semillas inicialmente dadas [1]. Una grieta o fractura comienza en las orillas de la cuadrícula y tiene mayor probabilidad de propagarse por las fronteras de las celdas de Voronoi que por el interior de las mismas. Se desea estudiar el efecto que tiene el número de semillas y el tamaño de la zona [2].

2. Objetivo

- Examinar de manera sistemática el efecto del número de semillas y del tamaño de la zona en la distribución de las grietas que se forman en términos de la mayor distancia manhattan entre la grieta y el exterior de la pieza [2].

3. Metodología

La metodología empleada se realizó a través de RStudio [3] llevando a cabo los pasos señalados en la *Práctica 4: Diagramas de Voronoi* [2], se examina las diferencias del número de semillas y del tamaño de la zona en la distribución de las grietas que se forman en terminos de la mayor distancia manhattan entre la grieta y el exterior de la pieza, el código completo de la metodología empleada se encuentra en el repositorio de GitHub [4].

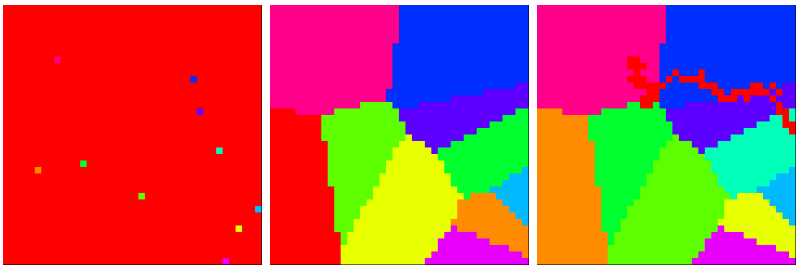
4. Resultados

Se obtuvo el código secuencia de GitHub de Schaeffer E. [5] y se adecuaron algunas modificaciones variando el número de semillas (10, 20, 40, 60, 80 y 100) y el tamaño de la zona (40, 50, 60, 70, 80, 100) con la finalidad de determinar el efecto que se tiene cambiando el numero de semillas en diferentes tamaños de zonas, se determino el largo de la grieta producida en términos de distancia manhattan determinando que tan lejos llega la grieta del borde, el mínimo de los máximos de la distancia medida de la penetración para evaluar que tan drástica es la fracturación. A continuación se muestra parte del código del experimento [4]

en el cual se señala el número de semillas y tamaño de zona, los datos obtenidos de distancia manhattan fueron agrupados en diagramas tipo violín.

```
n <-  c(40, 50, 60, 70, 80, 100)
zona <- matrix(rep(0, n * n), nrow = n, ncol = n)
k <- c(10, 20, 40, 60, 80, 100)
x <- rep(0, k)
y <- rep(0, k)
for (semilla in 1:k) {
  while (TRUE) {
    fila <- sample(1:n, 1)
    columna <- sample(1:n, 1)
    if (zona[fila, columna] == 0) {
      zona[fila, columna] = semilla
      x[semilla] <- columna
      y[semilla] <- fila
      break
    }
  }
}
celda <-  function(pos) {
  fila <- floor((pos - 1) / n) + 1
  columna <- ((pos - 1) %% n) + 1
  if (zona[fila, columna] > 0) { # es una semilla
    return(zona[fila, columna])
  } else {
    cercano <- NULL # sin valor por el momento
    menor <- n * sqrt(2) # mayor posible para comenzar la busqueda
    for (semilla in 1:k) {
      dx <- columna - x[semilla]
      dy <- fila - y[semilla]
      dist <- sqrt(dx^2 + dy^2)
      if (dist < menor) {
        cercano <- semilla
        menor <- dist
      }
    }
    return(cercano)
  }
}
```

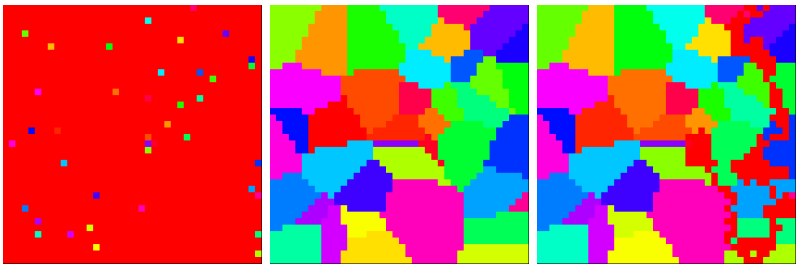
Para cada combinación se obtuvieron las semillas base (figura 1a), las celdas de Voronoi (figura 1b) y replicas de las fracturas (figura 1c), observe la figura 1.



(a) Semillas. (b) Tamaño 40×40. (c) Fractura.

Figura 1: Zona de 40×40 con 10 semillas.

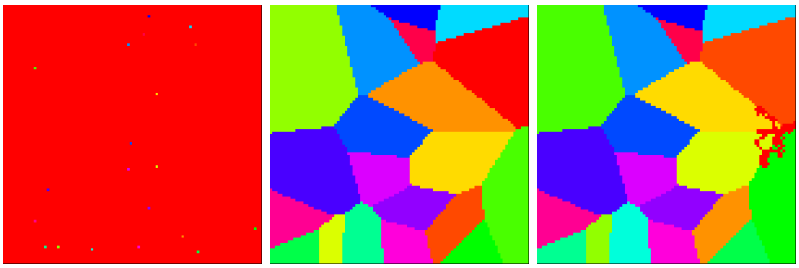
En la figura 2 se muestra un número de semillas (figura 2a) alto en una zona de 40×40 (figura 2b) se observa que la propagación de la fractura (figura 2c) es amplia, con el fin de cumplir los propósitos de esta práctica se tomará en cuenta el largo de la grieta en función de que tan lejos llegó desde el borde, en otras palabras tomando en cuenta la medida de penetración de la grieta y el borde para saber que tan drástica es la fracturación.



(a) Semillas. (b) Tamaño 40×40. (c) Fractura.

Figura 2: Zona de 40×40 con 40 semillas.

Al variar el número de semillas (figura 3a) en un determinado tamaño de zonas (figura 3b) se determinó el efecto de la penetración de la fractura (figura 3c) en la figura 3 se muestra una zona de 100×100 con un número de semillas bajo, debido aque la probabilidad de que la fractura propague sobre la frontera y no entre los diagramas de Voronoi se obtiene una frecuencia menor de fractura en una zona más grande para un número de semillas más limitado.

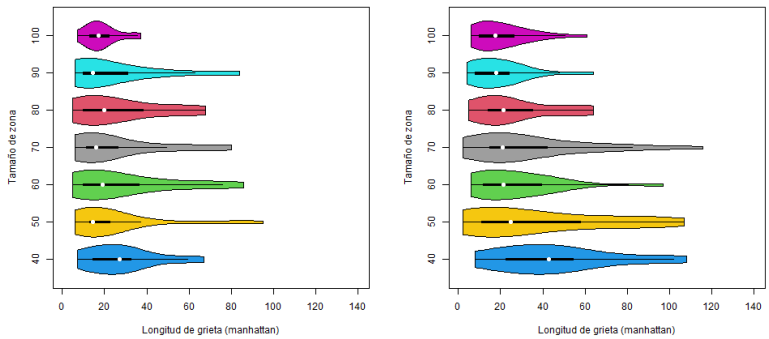


(a) Semillas. (b) Tamaño 100×100. (c) Fractura.

Figura 3: Zona de 100×100 con 20 semillas

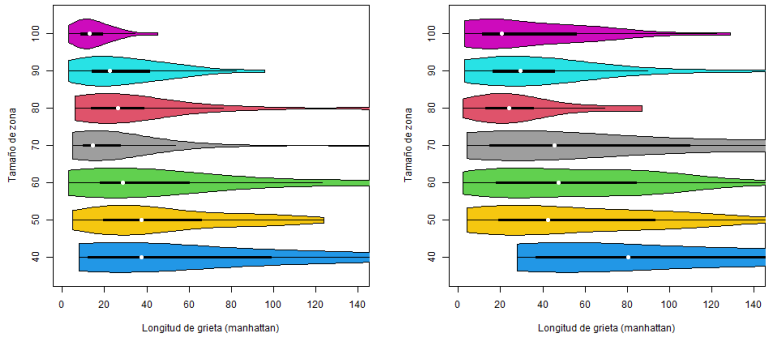
En la fig 4 y fig 5 se muestra un diagrama de tipo violín que marca la longitud de distancia tipo manhattan variando el tamaño de zona (n) en el eje Y, mientras que en el eje X la distancia que penetró la grieta desde el borde tomando en cuenta lo visto en la práctica 1.

Al observar los diagramas se deduce que la densidad de semillas afecta en el comportamiento de la fractura y la distancia que se recorrió como máximo desde el borde al punto de mayor penetración. Para mayor comprensión de lo



(a) 20 Semillas. (b) 40 Semillas.

Figura 4: Diagrama de violín con variación en el número de semillas.



(a) 60 Semillas. (b) 80 Semillas.

Figura 5: Diagrama de violín con variación en el número de semillas.

que pasa en cada experimento se obtuvo archivos .gif donde se muestra distintos recorridos de la fractura, los gráficos de las distancias manhatan para 10 y 100 semillas se encuentran en el repositorio de GitHub [4].

En el cuadro 1 se muestra un fragmento [4] de las distancias máximas alcanzadas desde el borde hasta lo máximo que penetro del eje la fractura.

Distancia Manhattan		
Zona.	Semillas.	Distancia desde borde
Tamaño 40×40.	10	18
Tamaño 50×50.	20	26
Tamaño 60×60.	40	50
Tamaño 70×70.	60	48
Tamaño 80×80.	80	66
Tamaño 100×100.	100	83

5. Conclusión

Al examinar el efecto del número de semillas y del tamaño de la zona en la distribución de las grietas que se forman en términos de la mayor distancia manhattan entre la grieta y el exterior de la pieza se observa que a un mayor número de semillas y una dimensión reducida la grieta propagara un mayor número de veces, aunque la grieta del mismo modo en una dimensión más grande puede propagar con mayor facilidad el número de veces que propaga es menor debido a la frontera entre semillas, es importante mencionar que si el número de semillas es bajo en una zona muy grande la posibilidad de generar una fractura sera demasiado baja tomando como ejemplo la metodología de 10 semillas en una zona de 100 x 100.

Referencias

[1] O. Cheong, "Computational geometry: Algorithms and applications," Marzo 2008. Recuperado el 05 de marzo del 2021.

[2] E. Schaeffer, "Práctica 4: Diagramas de voronoi," Marzo 2021. https://elisa.dyndns-web.com/teaching/comp/par/p4.html.

[3] J. J. Allaire, "Rstudio," Marzo 2021. https://rstudio.com.

[4] J. A. Garcia Marzo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%204.

[5] E. Schaeffer, "Práctica 4 diagramas de voronoi," Marzo 2021. https://github.com/fuentesadrian/Simulation/tree/master/VoronoiDiagrams.

Práctica Nº 5
Método Monte-Carlo

Nombre: José Adrián García Fuentes **Profesor:** Satu Elisa Schaeffer
Fecha: 16/Marzo/2021

---

1. Introducción

El método Monte-Carlo es idóneo para situaciones en las cuales algún valor o alguna distribución no se conoce y resulta complicado de determinar de manera analítica [1], es un método utilizado para conseguir aproximaciones de expresiones matemáticas y de alto grado para ser evaluadas con exactitud. Consiste en usar un espacio delimitado donde se encuentra la función a evaluar, la idea principal del método es generar números aleatorios dentro de ese espacio y calcular la proporción de puntos que están dentro o fuera, por arriba o por abajo de la función y obtener así una aproximación del área cubierta [2].

2. Objetivo

- Determinar el tamaño de muestra requerido por lugar decimal de precisión para el integral, comparando con Wolfram Alpha para por lo menos desde dos hasta cinco decimales [1].
- Representar el resultado como una sola gráfica con el número de decimales correctos contra el tamaño de muestra [1].

3. Metodología

La metodología empleada se realizó a través de RStudio [3] llevando a cabo los pasos señalados en la *Práctica 5: Método Monte-Carlo* [1], se implementa el método Monte-Carlo para aproximar el valor de la integral (1)

$$\int_a^b f(x)dx \tag{1}$$

para la función (2)

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

esta aproximación es posible porque la función (3)

$$2f(x)/\pi \tag{3}$$

es una distribución de probabilidad válida igual a 1, observe la ecuación (4).

$$\int_{-\infty}^{\infty} \frac{2}{\pi} f(x)dx = 1 \tag{4}$$

Este hecho permite generar números pseudoaleatorios con la distribución $g(x) = \frac{2f(x)}{\pi}$, así estimar $\int_a^b g(x)dx$, y de ahí normalizar el estimado para que sea $\int_a^b f(x)dx$.

A partir del código en el repositorio de Schaeffer [4] se puede realizar el gráfico de distribución de probabilidad como el de la figura 1, el código completo de la metodología empleada se encuentra en el repositorio de GitHub [5].

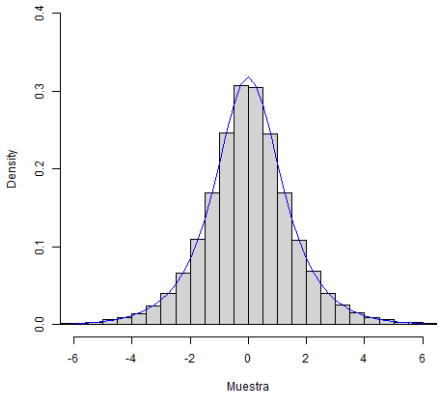


Figura 1:
Histograma de g(x) comparado con g(x)

4. Resultados

Se obtuvo el código secuencia de GitHub de Schaeffer [4] se adecuaron modificaciones con el fin de determinar el tamaño de muestra requerido por lugar decimal de precisión para el integral, se representaron los resultados en una sola gráfica con el número de decimales correctos contra el tamaño de la muestra (ver figura 2).

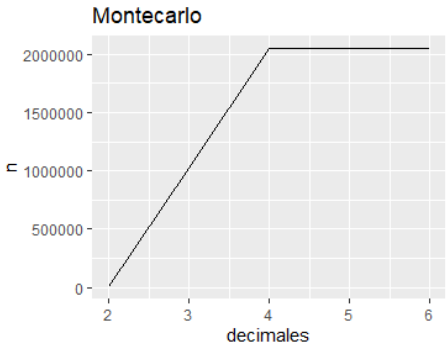


Figura 2:
Gráfico precisión de decimales variando tamaño de muestra [6].

A continuación se muestra parte del código del experimento [5].

```
library(distr)
library(ggplot2)
wolf=0.048834
rate=0.80
exitos=as.data.frame(matrix(nrow = 5, ncol = 1))
wolfram=cbind(round(wolf,digits=2),round(wolf,digits=3),round(wolf,
    digits=4),round(wolf,digits=5),wolf)
montecarlosf<-function(n) {
  desde <- 3
  hasta <- 7
  f <- function(x) { return(1 / (exp(x) + exp(-x))) }
  g <- function(x) { return((2 / pi) * f(x)) }
  suppressMessages(library(distr))
  generador  <- r(AbscontDistribution(d = g))
  valores <- generador(n)
  mc=sum(valores >= desde & valores <= hasta)
  integral <- sum(mc) / n
  resultado=(pi / 2) * integral
  return(resultado)
}
replica <- function() { replicate(reps, montecarlosf(nsample))  }
for (c in 2:6) {
  reps=20
  nsample=5000
  paso=1000
for (i in 1:10) {
resultado<-round(replica(),digits=c)
if (as.numeric(length(resultado[resultado == wolfram[1,c-1] ])/reps
    ) <= rate){
  paso=paso+paso
  nsample=nsample+paso
} else {
  break
}
}
  exitos[c-1,1]=nsample
}
ggplot()+geom_line(data=exitos,aes(x=c(2:6),y=exitos$V1))+labs(
    title="Montecarlo",y="n",x="decimales")
```

A medida que aumenta la cantidad de la muestra, el resultado se aproxima mayormente al valor obtenido en Wolfram Alpha (0,048834). Por tanto la precisión de los decimales se ve afectada por el tamaño de la muestra, observe como en la figura 2 se logra una mayor presición en el número de decimales mientras más grande es la muestra.

5. Conclusión

Si se aumenta el tamaño de n para la estimación de la integral, se obtendrá una mayor precisión con respecto al valor de Wolfram Alpha, este método es una opción viable para replicar el cálculo una cantidad de veces, ya que se trabaja con valores aleatorios, para el caso de muestras grandes se pueden tener aproximaciones en períodos cortos de tiempo con una mayor exactitud.

6. Reto 1

Implementar la estimación del valor de π de Kurt y determinar la relación matemática entre el número de muestras obtenidas y la precisión obtenida en términos del error absoluto [1].

Para la comprensión es necesario saber como determinar el área de un círculo (5)

$$A = \pi r^2 \tag{5}$$

y el área de un cuadrado que contiene ese círculo (6)

$$A = 4r^2 \tag{6}$$

La relación del área del círculo con el área de la plaza es (7)

$$\frac{\pi r^2}{4r^2} \tag{7}$$

que se puede reducir a (8).

$$\frac{\pi}{4} \tag{8}$$

Dado este hecho, si se puede determinar la relación del area del círculo con el área de la plaza [6].

```
runs <- 100000
xs <- runif(runs,min=-0.5,max=0.5)
ys <- runif(runs,min=-0.5,max=0.5)
in.circle <- xs^2 + ys^2 <= 0.5^2
mc.pi <- (sum(in.circle)/runs)*4
plot(xs,ys,pch='.',col=ifelse(in.circle,"blue","grey")
     ,xlab='',ylab='',asp=1,
     main=paste("MC Aproximacion de Pi =",mc.pi))
```

En la figura 3 se muestra la estimación del valor de π de kurt

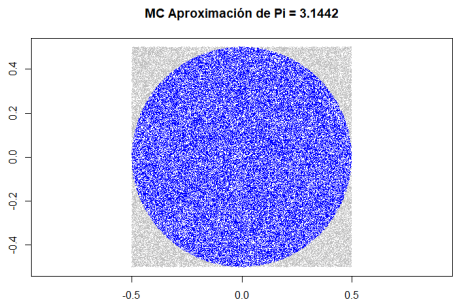


Figura 3:
Estimación del valor de π de Kurt [7].

De igual manera que la primera parte de esta práctica, se logra visualizar el mismo patrón a medida que n cambia [8]

7. Reto 2

Aplicar un método Monte Carlo para estimar la cantidad de pintura necesaria en un mural, comparando conteos exactos de píxeles de distintos colores con conteos estimados con muestreo aleatorio [1].

```
install.packages("jpeg")
library(jpeg)
img <- readJPEG("C:/Users/ADRIAN GARCIA/Desktop/img.jpg")
img <- as.raster(img)
tab <- table(img)
tab <- data.frame(Color = names(tab), Count = as.integer(tab))
RGB <- t(col2rgb(tab$Color))
tab <- cbind(tab, RGB)
tot=sum(tab$Count)
ratios=(tab$Count/tot)*100
tab$'cantidadpintura%'<-round(ratios,2)
```

En la figura 4 se muestra una pintura de mural en la que se comparo un total de 288000 pixeles de los cuales se obtuvo un total de variables de 6 colores distintos sin embargo se determino la cantidad de pintura en porcentaje para utilizar solo entre dos tipos de colores los datos obtenidos se encuentran dentro del repositorio [5]

Figura 4:
Pintura Mural [9]

Referencias

[1] E. Schaeffer, "Práctica 5: Método monte-carlo," Marzo 2021. https://elisa.dyndns-web.com/teaching/comp/par/p5.html.

[2] D. Peña Sanchez, "Deducción de distribuciones: el método de monte carlo," Fundamentos de estadistica Madrid. Recuperado el 15 de marzo del 2021.

[3] J. J. Allaire, "Rstudio," Marzo 2021. https://rstudio.com.

[4] E. Schaeffer, "Práctica 5: Método monte-carlo," Marzo 2021. https://github.com/fuentesadrian/Simulation/tree/master/MonteCarlo.

[5] J. A. Garcia Marzo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%205.

[6] O. Cheong, "Computational geometry: Algorithms and applications," Marzo 2008. Recuperado el 05 de marzo del 2021.

[7] kurt Will, "Monte carlo simulations in r," 2015. https://www.countbayesie.com.

[8] P. Pérez, "Aplicación de la técnica de simulación monte carlo," 2018. https://www.famaf.unc.edu.ar/~pperez1/manuales/cim/cap6.html.

[9] H. Weller, "Introduction to countcolors package," 2019. https://cran.r-project.org/web/packages/countcolors/vignettes/Introduction.html.

Práctica Nº 6
Sistema multiagente

Nombre: José Adrián García Fuentes **Profesor:** Satu Elisa Schaeffer
Fecha: 21/Marzo/2021

---

1. Introducción

Un sistema multiagente es un poco como un autómata celular: hay un conjunto de entidades con estados internos que pueden observar estados de los otros y reaccionar cambiando su propio estado. La diferencia es que un sistema multiagente es un concepto más general y permite que estos agentes se muevan y varíen su vecindad, entre otras cosas. En la sexta práctica se implementará un sistema multiagente con una aplicación en epidemiología. Los agentes podrán estar en uno de tres estados: **s**usceptibles, **i**nfectados o **r**ecuperados (esto se conoce como el modelo SIR) [1].

Los parámetros serán el número de agentes n y la probabilidad de infección al inicio P_i. Vamos a suponer, por simplicidad, que la infección produce inmunidad en los recuperados, por lo cual solamente los susceptibles podrán ser infectados. La probabilidad de contagio será en nuestro caso proporcional a la distancia euclidiana entre dos agentes $d(i, j)$ de la siguiente manera (1):

$$\mathrm{P}_c = \begin{cases} 0, & \text{si} \quad d(i,j) \geq \mathrm{r}, \\ \dfrac{r-d}{r}, & \text{en otro caso}, \end{cases} \tag{1}$$

donde r es un umbral [1].

Nuestros agentes tendrán coordenadas x y y , una dirección y una velocidad (expresadas las dos últimas simplemente en términos de $\triangle x$ y $\triangle y$). Vamos a posicionar los agentes, por ahora, uniformemente al azar en un torus formado por doblar un rectángulo de l x l, visualizando en todo momento el rectángulo en dos dimensiones [1].

2. Objetivo

- Vacunar con probabilidad P_v a los agentes al momento de crearlos de tal forma que están desde el inicio en el estado R y ya no podrán contagiarse ni propagar la infección [1].
- Estudiar el efecto estadístico del valor de P_v en el porcentaje máximo de infectados durante la simulación y el momento en el cual se alcanza ese máximo [1].

3. Metodología

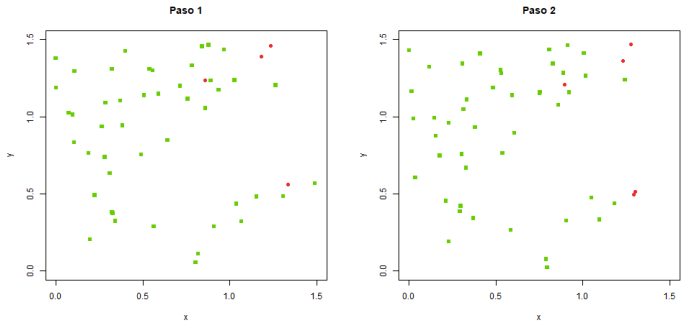
La metodología empleada se realizó a través de RStudio [2] llevando a cabo los pasos señalados en la *Práctica 6: Sistema multiagente* [1], a partir del código en el repositorio de Schaeffer [3], se hicieron modificaciones para tener un sistema multiagente en el que nuestro agente no viviera en una posición fija en la celda sino que pudiera moverse de posición en el cuadro, cada agente estará en uno de 3 posibles estados (susceptibles, infectados o recuperados) la probabilidad de la infección inicial es 0,05 % y la velocidad máxima es 1/30 la proporción de recuperación será 0,02 %, el código completo de la metodología empleada se encuentra en el repositorio de GitHub [4].

```
pi <- 0.05
pr <- 0.02
v <- 1 / 30
pv <- 0
pv_paso <- 0.10
for (vac in 1:11) {
  agentes <- data.frame(x = double(), y = double(),
                        dx = double(), dy = double(),
                        estado  = character())
  rnd<-runif(1)
  for (i in 1:n) {
    rnd<-runif(1)
    if (rnd < pv) {
      e <- "R"
    } else {
      e <- "S"
      if (rnd < pi) {
        e <- "I"
      }
    }
    agentes <- rbind(agentes, data.frame(x = runif(1, 0, 1),
                                         y = runif(1, 0, 1),
                                         dx = runif(1, -v, v),
                                         dy = runif(1, -v, v),
                                         estado = e))
  }
  levels(agentes$estado) <- c("S", "I", "R")
  epidemia <- integer()
  r <- 0.1
  rm <- 0.3
  pm <- 0.2
  ka <- 5
  tmax <- 100
  digitos <- floor(log(tmax, 10)) + 1
  for (tiempo in 1:tmax) {
    infectados <- dim(agentes[agentes$estado == "I",])[1]
    epidemia <- c(epidemia, infectados)
    if (infectados == 0) {
      break
    }
    contagios <- rep(FALSE, n)
    for (i in 1:n) {
      a1 <- agentes[i, ]
      if (a1$estado == "I") {
        for (j in 1:n) {
          if (!contagios[j]) {
            a2 <- agentes[j, ]
            if (a2$estado == "S") {
              dx <- a1$x - a2$x
```

```
            dy <- a1$y - a2$y
            d <- sqrt(dx^2 + dy^2)
            if (d < r) {
              p <- (r - d) / r
              if (runif(1) < p) {
                contagios[j] <- TRUE
              }
            }
          }
        }
      }
    }
    for (i in 1:n) {
      a <- agentes[i, ]
      if (contagios[i]) {
        a$estado <- "I"
      } else if (a$estado == "I") {
        if (runif(1) < pr) {
          a$estado <- "R"
        }
      }
      a$x <- a$x + a$dx
      a$y <- a$y + a$dy
      if (a$x > l) {
        a$x <- a$x - l
      }
      if (a$y > l) {
        a$y <- a$y - l
      }
      if (a$x < 0) {
        a$x <- a$x + l
      }
      if (a$y < 0) {
        a$y <- a$y + l
      }
      agentes[i, ] <- a
    }
    aS <- agentes[agentes$estado == "S",]
    aI <- agentes[agentes$estado == "I",]
    aR <- agentes[agentes$estado == "R",]
    tl <- paste(tiempo, "", sep="")
    while (nchar(tl) < digitos) {
      tl <- paste("0", tl, sep="")
    }
    salida <- paste("p6_t", tl, ".png", sep="")
    tiempo <- paste("Paso", tiempo)
    png(salida)
    plot(l, type="n", main=tiempo, xlim=c(0, l), ylim=c(0, l), xlab
    ="x", ylab="y")
    if (dim(aS)[1] > 0) {
      points(aS$x, aS$y, pch=15, col="chartreuse3", bg="chartreuse3
    ")
    }
    if (dim(aI)[1] > 0) {
      points(aI$x, aI$y, pch=16, col="firebrick2", bg="firebrick2")
    }
    if (dim(aR)[1] > 0) {
      points(aR$x, aR$y, pch=17, col="goldenrod", bg="goldenrod")
    }
    graphics.off()
  }
```

4. Resultados

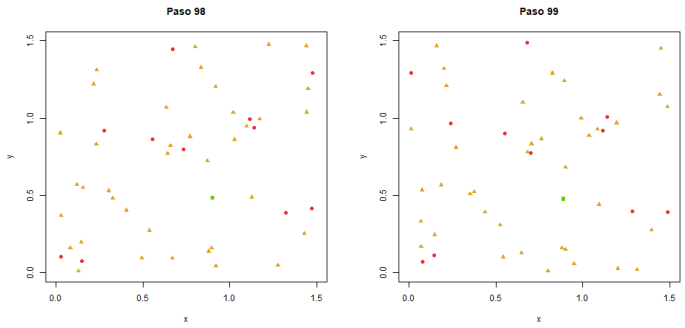
Se obtuvo el código secuencia de GitHub de Schaeffer [3] y se añadió probabilidad de vacunación a los agentes al momento de crearlos de tal forma que desde el inicio están en el estado recuperado y ya no podrán contagiarse ni propagar la infección se obtuvo el efecto estadístico del valor de probabilidad de vacunación (cuadro 1) en el porcentaje máximo de infectados durante la simulación y el momento en el cual se alcanza el máximo se generó un archivo .gif [5] de la simulación en la figura 1 se muestra la posición de los agentes cada uno al momento de iniciar tuvo una probabilidad de vacunación.



(a) (b)

Figura 1: Inicio simulación de agentes

Debido a que cada agente ya no propaga la infección y no puede contagiarse al final de la simulación el porcentaje de agentes se encuentra en un estado recuperado para mejor entendimiento ver figura 2.



(a) (b)

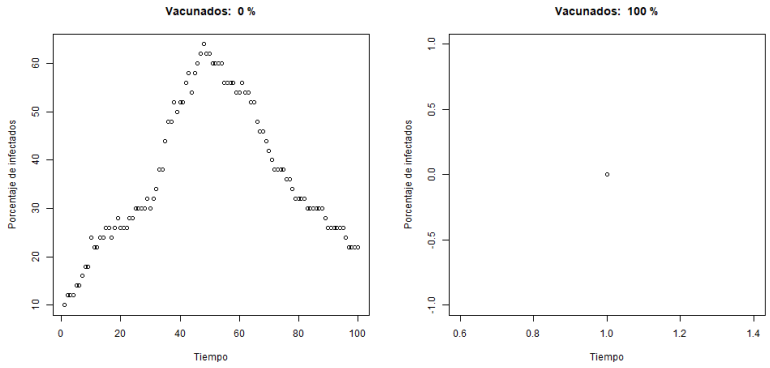
Figura 2: Fin de simulación de agentes

El efecto estadístico de la probabilidad de vacunación aunque es muy poco variable se observaria que dependiendo de la probabilidad de vacunación existe una ligera disminución en el porcentaje de contagiados salvo algunas exepciones 30 % y 50 %.

Efecto estadístico del valor de probabilidad de vacunación.

Probabilidad de vacunación	Máximo de infectados	Momento máximo de infectados (t)
0 %	65 %	49
10 %	60 %	52
20 %	58 %	19 y 38
30 %	82 %	17
40 %	56 %	20
50 %	80 %	17 y 22
60 %	76 %	20
70 %	68 %	30
80 %	53 %	26
90 %	50 %	60
100 %	0 %	0

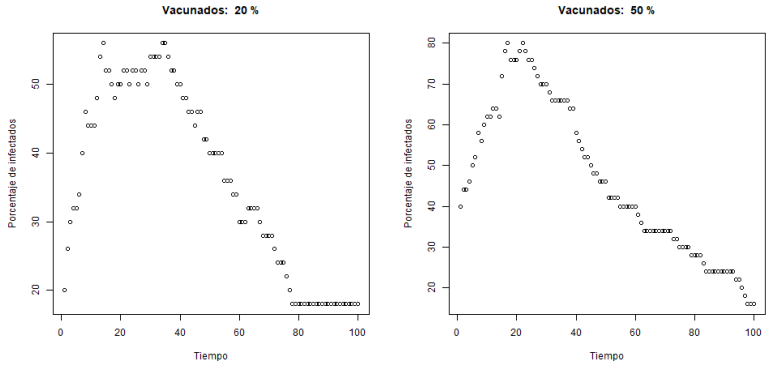
Al comparar que la población de agentes se encuentra completamente vacunada (figura 3) con una vacunación nula (figura 3a) se observa un porcentaje alto de agentes infectados (ver figura 3b)



(a) (b)

Figura 3: Comparación de agentes completamente vacunados con ningun agente vacunado.

Aunque se experaría que a mayor porcentaje de vacunados el porcentaje máximo de infectados disminuyera gradualmente existen algunas variantes como ejemplo se muestra la figura 4a donde una probabilidad del 20 % de agentes vacunados mostró un pico más alto que la probabilidad del 10 % y se observa un repunte en el contagio en los tiempos 19 y 38 o bien en la figura 4b donde ocurre lo mismo sin embargo se observa una baja en el porcentaje de infectados con respecto al tiempo (figura 4).



(a) (b)

Figura 4: Comparación de porcentaje de vacunacion con inconsistencias.

En la figura 5 se encuentra un gráfico de caja-bigote que determina el número de infectados por cada porcentaje de vacunados de manera descendente.

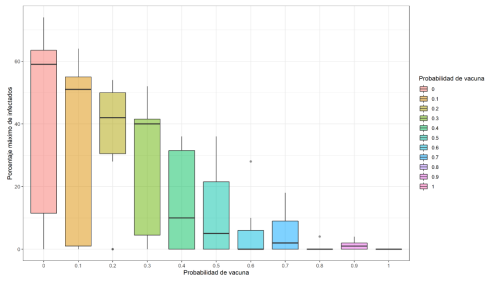


Figura 5: Comparación de porcentaje de vacunación

5. Conclusión

Al estudiar el efecto estadístico del valor de la probabilidad de vacunación con el porcentaje de agentes infectados en primera instancia se esperaría que el porcentaje de infectados disminuyera de forma gradual hasta que el total de la población de agentes estuviera completamente vacunado sin embargo en dos ocasiones se obtiene inconsistencias que tal vez estén dadas por las direcciones en las que los agentes se desplazan, un ejemplo simple sería la pandemia por Covid-19 en la que aunque la población no esté vacunada se tienen repuntes de contagios.

6. Reto 1

Cambiar los patrones de movimiento para que no tengan una trayectoria fija, cambiar los agentes a que utilize el modelo de punto intermedio aleatorio (inglés: random waypoint model): cada agente tiene una posición meta (x, y) hacia al cual se mueve con una velocidad v; al alcanzar (o superar) su meta, elige al azar una nueva meta uniformemente al azar. La velocidad de cada agente es una constante, normalmente distribuido sobre la población de agentes. Examina si surgieron cambios en el efecto de P_v por esta modificación [1].

Se obtuvo un código similar de GitHub de Vázquez [6] y se adaptó una meta cambiante para cada agente el objetivo es concluir si este nuevo tipo de movimiento de los agentes afecta o no la conclusión anterior.

```
xc<-runif(1,0,1)
yc<-runif(1,0,1)
px<-runif(1,0,1)
py<-runif(1,0,1)
```

Al declarar los puntos iniciales en los que aparecen los agentes y sus puntos meta, se genera un número aleatorio de pasos con el que se calcula la velocidad con la que avanza el punto meta cada agente en la figura 6 se muestra un diagrama caja-bigote de manera descendente aunque en comparación del gráfico de la tarea base los valores en el porcentaje de infectados no son tan separados se concentran en un rango ligeramente alto.

```
vx<-((px-xc/pasos)
vy<-((py-yc)/pasos)
```

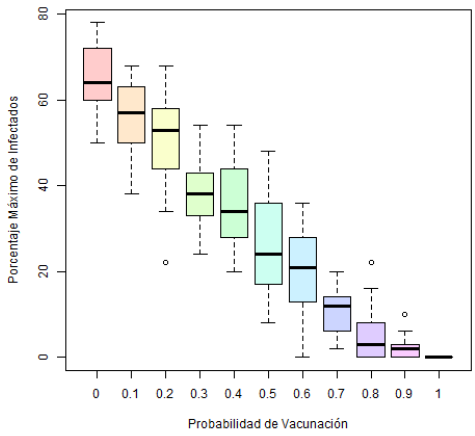


Figura 6: Gráfico caja-bigote porcentaje de infectados al establecer un punto meta aleatorio.

7. Conclusión

Se obtiene el mismo efecto en el que el valor de P_v afecta el porcentaje de infectados sin embargo debido a que las interacciones entre agentes son muy variables el porcentaje de infectados por tiempo es mayor para cada valor de P_v.

8. Reto 2

Los agentes tienen amistades: si se encuentran a una distancia euclidiana no mayor a r_a de un amigo suyo, se disminuye su velocidad a la mitad por k_a iteraciones (para saludar a su amigo). Cada par de agentes tiene una amistad con una probabilidad P_a. Examina nuevamente si surgieron cambios en el efecto de P_v por esta modificación, con valores $0 < r_a < 1$, $k_a > 1$ y $0 < P_a << 1$ de tu elección [1].

```

library(gifski)

setwd("C:/Users/ADRIAN GARCIA/Documents/practica6")

l <- 1.5
n <- 50
pi <- 0.05
pr <- 0.02
v <- l / 30
pa<-0.02
pv <- 0
pv_paso <- 0.10

for (vac in 1:11) {
  agentes <- data.frame(x = double(), y = double(),
                        dx = double(), dy = double(),
                        estado  = character())
  rnd<-runif(1)
for (i in 1:n) {
  rnd<-runif(1)
  if (rnd < pv) {#vacunados
    e <- "R"
  } else {
    e <- "S"
    if (rnd < pi) {
      e <- "I"
    }
  }
  agentes <- rbind(agentes, data.frame(x = runif(1, 0, l),
                                       y = runif(1, 0, l),
                                       dx = runif(1, -v, v),
                                       dy = runif(1, -v, v),
                                       estado = e))
}

levels(agentes$estado) <- c("S", "I", "R")
epidemia <- integer()
r <- 0.1
rm <- 0.3
pm <- 0.2
ka <- 5
tmax <- 100
```

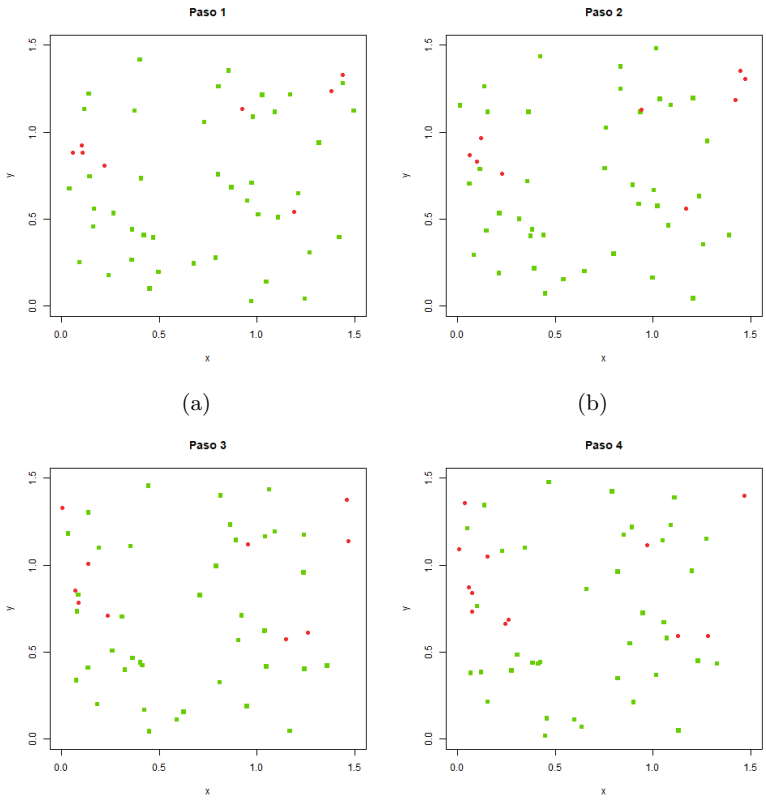
```
digitos <- floor(log(tmax, 10)) + 1

for (tiempo in 1:tmax) {
  infectados <- dim(agentes[agentes$estado == "I",])[1]
  epidemia <- c(epidemia, infectados)
  if (infectados == 0) {
    break
  }
  contagios <- rep(FALSE, n)
  for (i in 1:n) {
    a1 <- agentes[i, ]
    if (a1$estado == "I") {
      for (j in 1:n) {
        if (!contagios[j]) {
          a2 <- agentes[j, ]
          if (a2$estado == "S") {
            dx <- a1$x - a2$x
            dy <- a1$y - a2$y
            d <- sqrt(dx^2 + dy^2)
            if (d < rm) { # umbral amistad
              p <- (r - d) / r
              if (runif(1) < pm) {
                for (i in 1:ka) {
                dx <- (a1$x - a2$x)/2
                dy <- (a1$y - a2$y)/2
                }
              }
            }
            if (d < r) {
              p <- (r - d) / r
              if (runif(1) < p) {
                contagios[j] <- TRUE
              }
            }
          }
        }
      }
    }
  }
  for (i in 1:n) {
    a <- agentes[i, ]
    if (contagios[i]) {
      a$estado <- "I"
    } else if (a$estado == "I") {
      if (runif(1) < pr) {
        a$estado <- "R"
      }
    }
    a$x <- a$x + a$dx
    a$y <- a$y + a$dy
    if (a$x > l) {
      a$x <- a$x - l
    }
    if (a$y > l) {
      a$y <- a$y - l
    }
    if (a$x < 0) {
      a$x <- a$x + l
    }
    if (a$y < 0) {
      a$y <- a$y + l
    }
```

```
    agentes[i, ] <- a
  }
  aS <- agentes[agentes$estado == "S",]
  aI <- agentes[agentes$estado == "I",]
  aR <- agentes[agentes$estado == "R",]
  tl <- paste(tiempo, "", sep="")
  while (nchar(tl) < digitos) {
    tl <- paste("0", tl, sep="")
  }
  salida <- paste("p6_t", tl, ".png", sep="")
  tiempo <- paste("Paso", tiempo)
  png(salida)
  plot(l, type="n", main=tiempo, xlim=c(0, l), ylim=c(0, l), xlab="
    x", ylab="y")
  if (dim(aS)[1] > 0) {
    points(aS$x, aS$y, pch=15, col="chartreuse3", bg="chartreuse3")
  }
  if (dim(aI)[1] > 0) {
    points(aI$x, aI$y, pch=16, col="firebrick2", bg="firebrick2")
  }
  if (dim(aR)[1] > 0) {
    points(aR$x, aR$y, pch=17, col="goldenrod", bg="goldenrod")
  }
  graphics.off()
}
salida <- paste("vacuna=", pv, ".png", sep="")
png(salida)
plot(1:length(epidemia), 100 * epidemia / n, xlab="Tiempo", ylab="
    Porcentaje de infectados",
  main=paste("Vacunados: ",pv * 100, "%"))
graphics.off()
pv <- pv + pv_paso
}
png_files <- list.files( pattern = "p6_t.*png$", full.names = TRUE)
gifski(png_files, gif_file = "animation.gif", width = 800, height =
     600, delay = 1)
```

A partir del código de la tarea base se añadió a cada agente amistades con (P_a de 0,02 %) y si estos se encontraban a una distancia euclideana de un amigo suyo este disminuiría su velocidad se añadió probabilidad de vacunación a los agentes al momento de crearlos, se obtuvo el efecto estadístico del valor de probabilidad de vacunación (cuadro 2) en el porcentaje máximo de infectados durante la simulación y el momento en el cual se alcanza el máximo se generó un archivo .gif [5] de la simulación.

En la figura 7 se muestran los primeros 4 pasos de una simulación si observamos en el paso uno (7a) se encuentra un agente contagiado en la parte central, un poco en la parte superior derecha, en el paso 2 (7b) se encuentra con un amigo provocando disminución en su velocidad a la mitad, en el paso 3 (7c) se observa que el mismo agente no se separó mucho en distancia del agente con el que se encontró en comparación del movimiento de los otros agentes, observe el paso 4 (7d) la distancia entre todos los agentes es mayor en comparación con el agente antes mencionado.



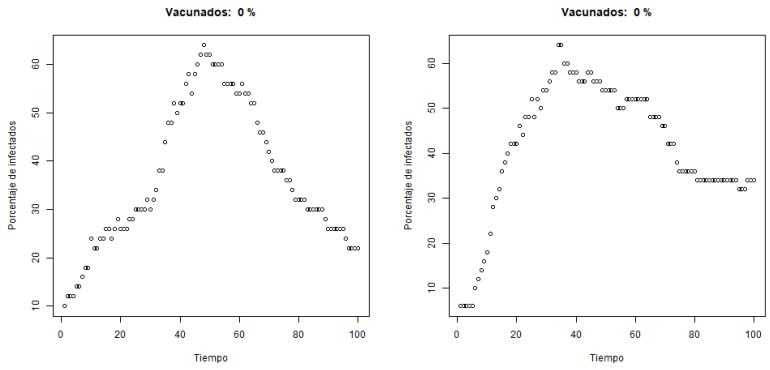
(a) (b)

(c) (d)

Figura 7: Inicio simulación de agentes

Efecto estadístico del valor de probabilidad de vacunación.		
Probabilidad de vacunación	Máximo de infectados	Momento máximo de infectados (t)
0 %	55 %	60
10 %	50 %	60
20 %	65 %	30
30 %	56 %	60
40 %	86 %	17
50 %	78 %	22
60 %	78 %	17 y 30
70 %	70 %	18
80 %	82 %	17
90 %	2 %	3
100 %	0 %	0

Al realizar una comparación con el grafico de la tarea base (figura 8a) y el efecto que tiene la probabilidad de amistad (figura 8b) en los agentes que no se encuentran vacunados (figura 8) se observa un porcentaje alto de agentes infectados (ver figura 8).



(a) (b)

Figura 8: Comparación de gráficos tarea base y reto 2.

Se obtienen gráficos muy similares sin embargo el porcentaje de infectados por tiempo es ligeramente mayor tal vez esto sea dado por el tiempo en que se desplazan los agentes cuando interactúan con un amigo y las probabilidades en que otro agente se pueda contagiar dependerán de sí se topan con él.

9. Conclusión.

El efecto estadístico de la probabilidad de vacunación en comparación de la tarea base cambia aunque en algunas ocasiones cambia muy poco se podría decir que se mantiene en un porcentaje de infectados en un rango de 55 a 80 % a excepción del 90 % de vacunados que solo se infectó un 2 %y el tiempo de contagio fue rapido tal vez estos cambios en la variacion de infectados es debido al tiempo en el que se retrasan los agentes al estar con un amigo y acorta de cierta manera la probabilidad de contagio.

Referencias

[1] E. Schaeffer, "Práctica 6: Sistema multiagente," Marzo 2021. https://https://elisa.dyndns-web.com/teaching/comp/par/p6.html.

[2] J. J. Allaire, "Rstudio," Marzo 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 6: Sistema multiagente," Marzo 2021. https://github.com/fuentesadrian/Simulation/tree/master/MultiAgent.

[4] J. A. Garcia Marzo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%206.

[5] A. Garcia, "archivo gif," 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/blob/main/Tarea%206/animation.gif.

[6] f. Vazquez, "Práctica 6: Sistema multiagente," Marzo 2021. https://github.com/fvzqa/Simulacion.

Práctica Nº 7
Búsqueda local

Alumno: José Adrián García Fuentes
Profesor: Satu Elisa Schaeffer
Fecha: 12/abril/2021

1. Introducción

Cuando se requiere optimizar un proceso se hace uso del método de búsqueda local, la característica principal de este método es la realización de movimientos en el espacio, cada uno de estos movimientos representa una solución, la cual puede ir mejorando, en pocas palabras, la búsqueda local inicia con una solución inicial, la cual remplaza con una nueva hasta encontrar la mejor solución y que no exista una solución mejor. En esta práctica implementamos una optimización heurística sencilla para encontrar máximos locales de funciones [1].

2. Objetivo

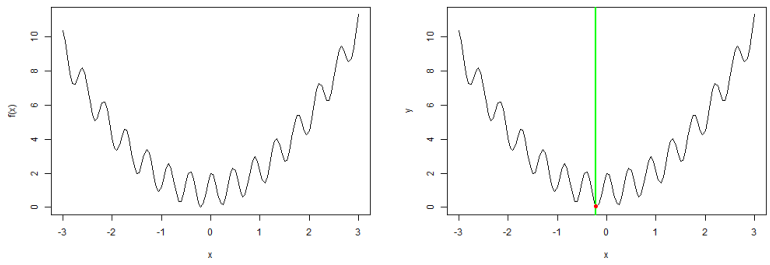
- Maximizar alguna variante de la función bidimensional ejemplo, con restricciones $-3 \leq x, y \leq 3$ [1].
- Crear una visualización animada de cómo proceden 15 réplicas simultaneas de la búsqueda encima de una gráfica de proyección plana [1].

3. Metodología

La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la práctica 7: búsqueda local [1], a partir del código en el repositorio de Schaeffer [3], se realizaron modificaciones para obtener una curva función objetivo en que se requiere estar tan arriba como posible, calculando el valor de los puntos en $y = f(x)$ (valores al azar y en movimiento), se agrega una linea que marca el mayor valor de y. Agregando una función de menos simetría con un plot tridimensional y controlando el angulo para una mejor visualización de el punto más alto.

4. Resultados

En la figura 1 se muestra una curva con una función objetivo en la que queremos maximizar estar tan arriba como posible, se requiere realizar una simulación en la que nuestro punto caiga en puntos al asar de la curva y estos se muevan sobre el eje, por tanto calcular el valor de $y = f(x)$ para múltiples diferentes valores de x. En la figura 1b se muestra el punto más bajo posible, este código fue modificado con la finalidad de mostrar el punto más alto posible para realizar una comparación ver figura 2 donde se maximiza el valor de la variable (figura 2a) y se modifica la función obteniendo menos simetría (figura 2b), el mayor valor que yo he visto se marcara con una línea y no se moverá si mi punto no ha tenido un valor mayor.

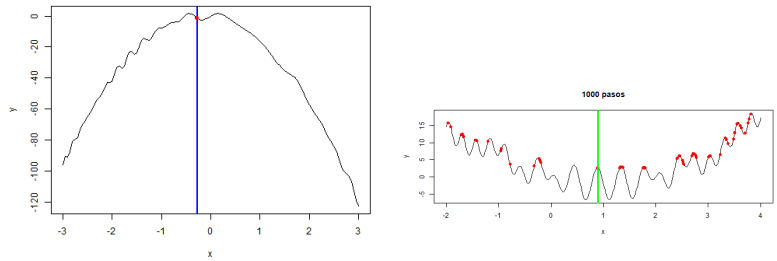


(a) Curva función objetivo. (b) Punto más bajo posible.

Figura 1: Función de la curva de Schaeffer donde se encuentra el punto tan abajo como posible

```
g <- function(x, y) {
    return(((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y
    ^2 - 20 * y)/100)
repeticiones<-15
```

En el siguiente código se muestra la función de la curva que fue sustituida n veces para comprender cual función tenia menos simetria.



(a) Maximo punto de la variable. (b) Replicas de puntos más altos posibles modificando la función.

Figura 2: Función de la curva donde se encuentra el punto más alto como posible.

Para mayor visualización observar los archivos .gif en el repositorio de GitHub [4], ahora se requiere agregar una función tridimensional tal como se observa en la figura 3 y controlar el ángulo para observarlo desde la parte superior en dos direcciones marcando con un punto el mayor valor obtenido de por lo menos 15 replicas. En la figura 4 se muestran los resultados en mapas de calor de las 15 replicas en secuencias marcando con puntos de colores la posición máxima alcanzada, tal como se muestra en la figura 4b donde el valor maximo alcanzado sobre el eje z es muy cercano a 0 para todas las replicas.

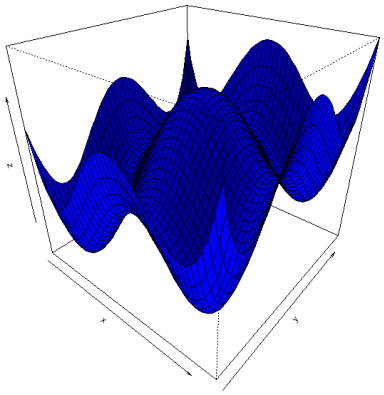
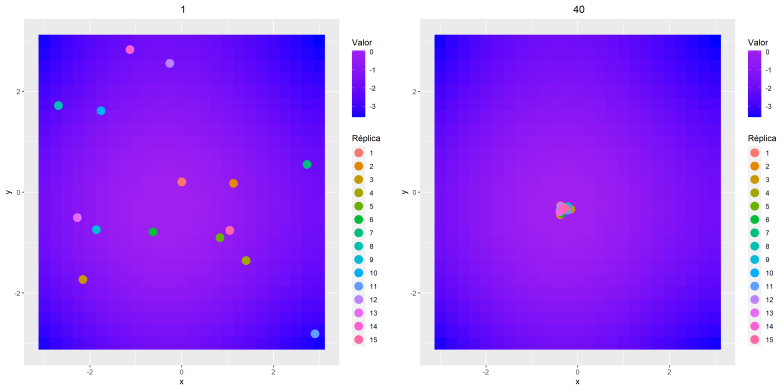


Figura 3: Función tridimensional.



(a) Máximo inicial sobre eje z. (b) Máximo valor sobre eje z.

Figura 4: Mapa de calor de 15 replicas simultaneas con máximo valor en z.

5. Conclusión

En cuanto tengamos un mayor numero de pasos de nuestro punto existe mayor probabilidad de alcanzar el punto más alto, dentro de los resultados de nuestro mapa de calor se muestra una cercancía al mayor valor de z que es 0 tal como se muestra en la figura 4b.

Referencias

[1] E. Schaeffer, "Práctica 7: Búsqueda local," abril 2021. https://elisa.dyndns-web.com/teaching/comp/par/p7.html.

[2] J. J. Allaire, "Rstudio," abril 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 7: Búsqueda local," ABRIL 2021. https://github.com/fuentesadrian/Simulation/tree/master/LocalSearch.

[4] J. A. Garcia Marzo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%207.

Práctica 8: Modelo de urnas

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer

Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica
16 de junio de 2021

1. Introducción

El modelo de la urna es uno de los elementos de mayor uso en la aplicación de probabilidades estadísticas ya que tiende a ser un concepto que permite la facilidad de comprender gráficamente las distintas formas en las cuales puede aplicarse dicho modelo, un modelo de urnas es aquel que trata de simular un contenedor con elementos para calcular la probabilidad de extraer un elemento de la jornada, alguna característica del elemento, por ejemplo, podemos tener una urna con bolas de un tamaño y bolas de otro donde se desea conocer cuál es la probabilidad de una bola de menor tamaño. En la práctica se simula un sistema en donde se abordan los fenómenos de coalescencia y fragmentación de partículas, donde las partículas se unen y se descomponen para formar cúmulos, estos fenómenos son de gran utilidad al realizar análisis en muchas áreas como en física y química. Esto puede servir en la práctica de laboratorio como para lograr predecir qué cantidad de partículas quedaran atrapadas en un filtro de cierta apertura de poro, por ejemplo, supongamos que tenemos alguna solución y deseamos filtrar una determinada partícula una de las características más relevantes de dicha partícula es su tamaño se cuenta con una red que sólo captura las partículas de tamaño que se especifique, mediante el procedimiento solicitado por el uso del modelo de urnas aplicando el principio para este modelo generaríamos un número de enteros distribuido de tal manera que se agrupen al tamaño de los cúmulos originalmente iniciado.

2. Objetivo

Graficar el porcentaje que se logra filtrar en cada iteración.

3. Resultados

Para la simulación se toman en cuenta dos parámetros principales, que son el número de partículas $n = 16k$, $32k$, $64k$, $128k$ y el número de cúmulos $k = 1000$ [1]. La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la práctica 8: modelo de urnas [1], a partir del código en el repositorio de Schaeffer [3], se realizaron modificaciones, los resultados de la experimentación los podemos ver en la figura 1 donde el eje vertical nos indica el porcentaje de las partículas que se logran filtrar y en el eje horizontal la iteración, correspondiendo los colores a la cantidad de cúmulos.

```
vectorn <- c(16*k,32*k,64*k,128*k)

  for (replica in 1:30) {
    basefiltrados <- c()
      for (n in vectorn) {
```

Los valores se agregan al vector n solicitado por el número de cúmulos y se realizan un total de 30 réplicas, se crea la variable filtrado para guardar los valores.

```
filtrado <- c()

 h <- cumulos[cumulos > c]
      filtrado[paso] <- sum(h) / n
    }
    basefiltrados <- cbind(basefiltrados,
    filtrado)
  }
  colnames(basefiltrados) <- vectorn

```

Para mejorar la resolución de la imagen se agregó formato `.tiff` sustituyendo el formato `.png`, se añadió una leyenda en la parte inferior derecha para una mayor comprensión visual del gráfico.

```
  tiff(paste("p8_", replica, ".tiff", sep=""),
     width=12, height=12, units="cm", res=600,
     pointsize = 10)
```

```
  plot(basefiltrados[,1], type = "l", col= "
    red", ylim=c(min(basefiltrados), max(
    basefiltrados)),
      main = paste("Replica", replica), xlab
    = "Iteraciones", ylab = "Porcentaje de
    filtraciones")
  lines(basefiltrados[,2], type = "l", col= "
    blue")
  lines(basefiltrados[,3], type = "l", col= "
    black")
  lines(basefiltrados[,4], type = "l", col= "
    green")
  legend("bottomright",legend = c("16k","32k",
     "64k", "128k"),fill=c("red","blue", "
    black", "green"))
  dev.off()
  print(replica)
}
```

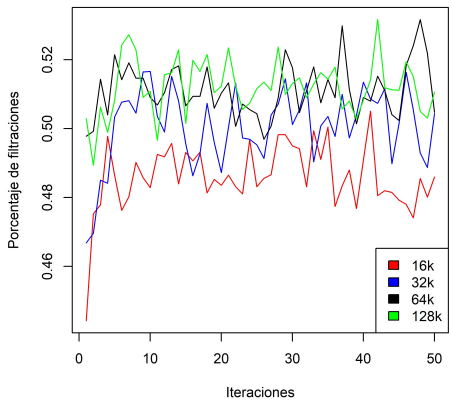


Figura 1: Porcentaje de filtración de los cúmulos en la réplica 1.

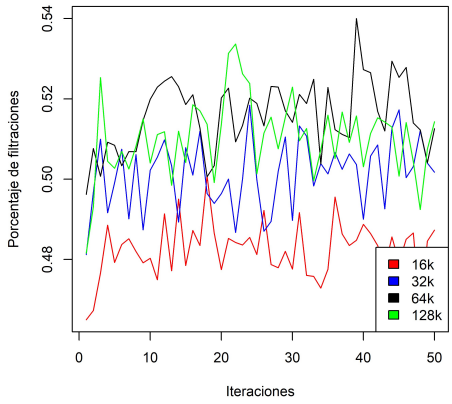


Figura 2: Porcentaje de filtración de los cúmulos en la réplica 30.

4. Conclusión

En la figura 1 se muestra que los cúmulos de mayor tamaño ($64k$, $128k$) tienen un mayor porcentaje de filtración que los de menor tamaño ($16k$, $32k$). A diferencia de lo que se puede observar en la figura 2 el valor más alto de la curva se encuentra en los cúmulos principales en este caso en la iteración 40 de 64 cúmulos (color negro) y a medida que avanza la iteración se observa que el filtrado avanza, por tanto los cúmulos más grandes se filtran con mayor facilidad pero tardan más en formarse que los cúmulos pequeños.

5. Reto 1

Determina si algún intervalo de iteraciones en el que el filtrado alcance un óptimo. Realiza réplicas para determinar si el momento en el cual se alcanza el máximo tiene un comportamiento sistemático. Incluye visualizaciones para justificar las conclusiones.

```
x1 <- c()
  for (m in 1:ncol(basefiltrados)) {
    x<-which.max(basefiltrados[,m])
    x1<-c(x1, x)
  }
  x2 <- rbind(x2,x1)
}
colnames(x2) <- vectorn
rownames(x2) <- seq(1:replica)
```

Mediante which.max se obtienen los valores más altos de las 50 iteraciones agrupadas en filas para los 4 diferentes tamaños de n (número de partículas) los resultados obtenidos se muestran en las figuras 3, 4, 5, 6 obteniendo resultados variables.

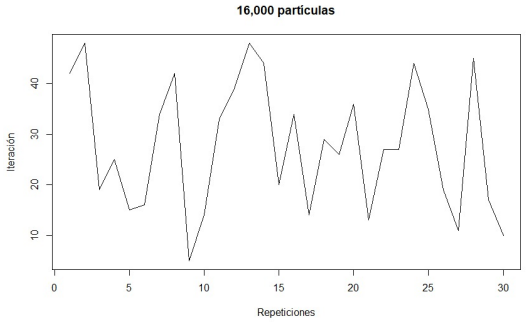


Figura 3: Gráfico de réplicas filtrando los mejores valores de $16k$.

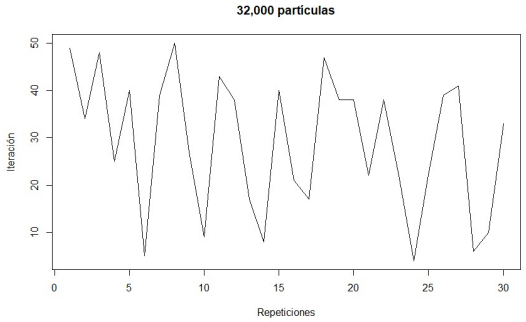


Figura 4: Gráfico de réplicas filtrando los mejores valores de $32k$.

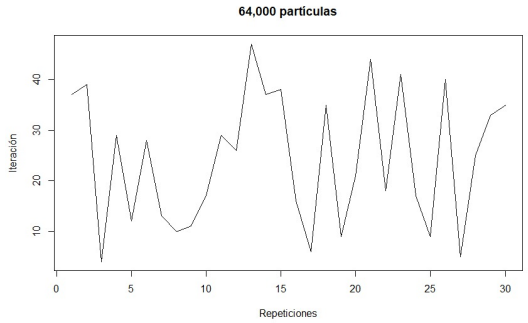


Figura 5: Gráfico de réplicas filtrando los mejores valores de $64k$.

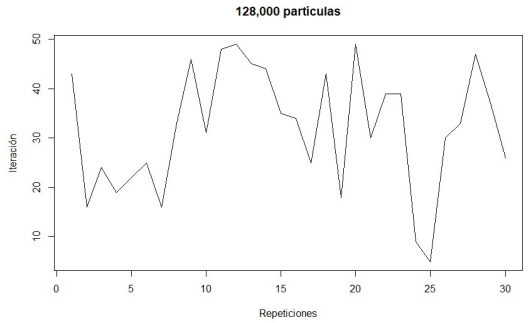


Figura 6: Gráfico de réplicas filtrando los mejores valores de $128k$.

Para el reto uno se puede concluir que el intervalo de iteraciones realmente es muy variable sin embargo para el caso de $32k$ y $128k$ puede llegar a alcanzar picos más altos que los cúmulos de $16k$ y $64k$, en la figura 4 se muestra un comportamiento uniforme en el largo y ancho de los picos comparado con el resto.

6. Reto 2

Determina cómo los resultados de la tarea y del primer reto dependen del valor de c. ¿Qué cambia y cómo si c ya no se asigna como la mediana inicial sino a un valor menor o mayor?

```
for (q in 1:2) {
```

Primero se genera un ciclo for para la variable q cuando q sea igual a 1 la variable c tomará el valor y se le restará la desviación estándar en otro caso se sumará, provocando la modificación de c.

```
if (q == 1) {
      c <- median(cumulos) - sd(cumulos) #
    tamanio critico de cumulos
    } else {c <- median(cumulos) + sd(cumulos)
    }
```

En las figuras 7 y 8 se muestran los gráficos con valores mayores y menores que c respectivamente, comparando con la tarea base y el primer reto los valores de filtración son más uniformes salvo a excepción del menor número de cúmulos $16k$ para el caso de c menor.

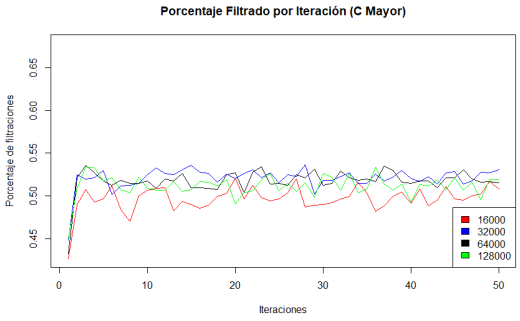


Figura 7: Gráfico del porcentaje filtrado por iteracción c mayor.

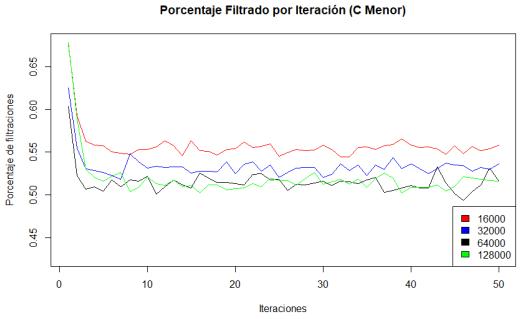


Figura 8: Gráfico del porcentaje filtrado por iteracción c menor.

Para el caso de los mejores filtrados y llevando a cabo los pasos del reto 1 se observan en las figuras 9 y 10 que si c es menor el porcentaje de filtración será muy bajo sin embargo si c es mayor el porcentaje de filtración será variable.

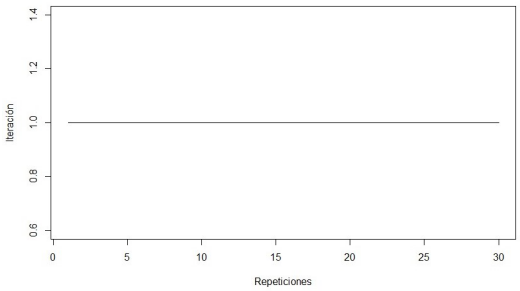


Figura 9: Gráfico de $16k$ con un valor de c menor.

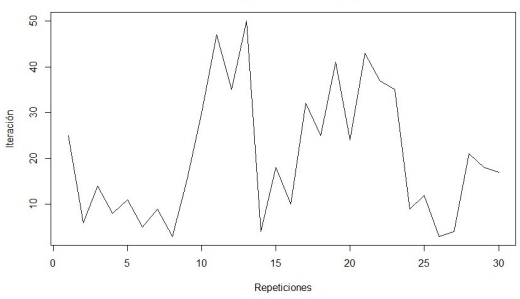


Figura 10: Gráfico de $16k$ con un valor de c mayor.

Para el reto dos se puede concluir que al modificar el valor de c ya sea mayor o menor se obtiene un porcentaje de filtrado más uniforme para las 4 cantidades diferentes de n salvo alguna excepción para el caso de c menor, pero esto puede deberse a la cantidad de n en $16k$.

Referencias

[1] E. Schaeffer, "Práctica 8: Modelo de urnas," abril 2021. https://elisa.dyndns-web.com/teaching/comp/par/p8.html.

[2] J. J. Allaire, "Rstudio," mayo 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 7: Búsqueda local," ABRIL 2021. https://github.com/fuentesadrian/Simulation/tree/master/LocalSearch.

Práctica 9: Interacciones entre partículas

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer.

Universidad Autónoma de Nuevo León. Facultad de Ingeniería Mecánica y Eléctrica.
25/abril/2021

Resumen

En esta práctica se analizará un modelo de atracción y repulsión de partículas con carga. Cada una de estas partículas tiene una propiedad inicial. El principal objetivo es agregar una nueva propiedad llamada masa. Esto permite que se pueda agregar por medio de ecuaciones el comportamiento que ejerce las fuerzas electrostáticas. En cierta distancia las partículas pueden ser atraídas por una fuerza, pero también mantienen su propia masa constante, lo que significa que presentan repulsión a cierta medida.

Palabras Claves: *atracción, repulsión.*

1. Introducción

En la práctica 9 se detallan las instrucciones y códigos para simular una interacción entre partículas con carga en un espacio bidimensional. Estas partículas siguen leyes de atracción similares a las expresadas por la ley de Coulomb, moviéndose en una dirección que depende de la interacción de cargas presentes en el sistema y con una fuerza directamente proporcional a la diferencia de cargas e inversamente proporcional a la distancia que las separa. Supongamos que contemos con partículas que habitan un cuadro unitario bidimensional y que cada partícula tiene una carga eléctrica, distribuida independiente y normalmente al azar entre $[-1, 1]$. Cargas de un mismo signo producirán una repulsión mientras cargas opuestas resultan en una atracción, la magnitud de la fuerza estará proporcional a la diferencia de magnitud de las cargas (mayores diferencias resultando en fuerzas mayores), y además la fuerza será inversamente proporcional a la distancia euclidiana entre las partículas. Vamos a comenzar creando y posicionando las partículas, usando la distribución normal para las coordenadas x y y . Ahora, cada partícula va a ejercer una fuerza sobre cada otra partícula. Vamos a implementar la atracción entre cargas con signos opuestos y la repulsión entre signos iguales. Habrá que sumar los efectos de todas las fuerzas individuales para determinar la fuerza total sobre una partícula en específico. Luego debemos normalizar el efecto de esa fuerza con un factor de descuento antes de poder trasladar la partícula con desplazamientos y que dependen de los componentes horizontales y vertical de la fuerza total [1].

2. Objetivo

- Agregar a cada partícula una masa y que la masa afecte a la velocidad de movimiento [1].
- La masa agregada causara una fuerza de atraccion a otra masa [1].
- Graficar los 3 factores Magnitud de carga, Velocidad, y Masa de partículas [1].
- Agregar reto 1 en donde las partículas más grandes absorban a las más pequeñas en caso de quedar entre dos de ellas [1].
- Agregar reto 2 con diferencia de sobreposición en las partículas [1].

3. Metodologia

La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la *Práctica 9: Interacciones entre partículas* [1], a partir del código en el repositorio de Schaeffer [3], se realizaron modificaciones. El código completo de la metodología empleada se encuentra en el repositorio de GitHub [4] donde también se pueden encontrar los archivos .Gif de los retos.

4. Resultados

En esta práctica se tienen una cantidad n de partículas en un cuadro unitario, cada una de ellas posee una carga y una masa. El objetivo es verificar gráficamente que existe una relación entre los factores velocidad, carga y masa de partículas. Las partículas son posicionadas normalmente al azar en el cuadro unitario tal como se muestra en la figura 1.

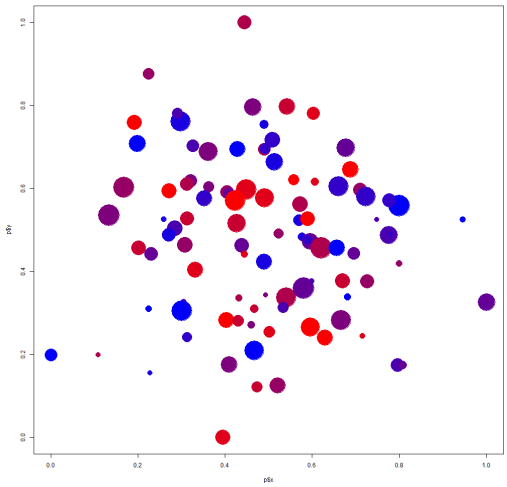


Figura 1: Partículas Carga, Masa, Velocidad.

Además, éstas posen una carga que causa fuerzas de repulsión y atracción con las demás partículas si dos partículas tienen el mismo signo o diferente, respectivamente. Se desea añadir otro atributo a las partículas que afecte dicha interacción, este atributo es la masa. Se muestra la interacción entre partículas, el atributo de la masa es representado con círculos de diferente tamaño para mostrar las diferencias entre las masas de las partículas. Esta interacción se puede apreciar de una mejor manera en una animación [4]. Toda la información se almacena en un data.frame, del cual se muestra un fragmento en el cuadro 1 para observar los datos obtenidos de cada partícula ver el apartado de factores en el repositorio de github [4].

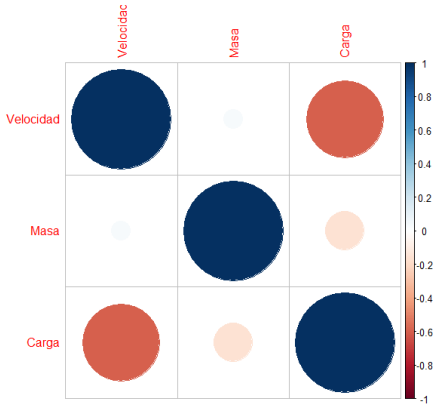


Figura 2: Gráfico corrplot Velocidad, Masa, Carga.

Con la información obtenida, se realizan diversos gráficos los factores: velocidad, masa y carga. La figura 2 muestra dichos gráficos y como se aprecia la carga y la masa afectan a la velocidad y su relación se muestra en el cuadro 2. En la figura 3 se muestran las partículas con masa mas grande que absorben alas demás cuando se colocan entre dos partículas y en la figura 4 las partículas interactúan entre si, sin sobreponerse.

Observe los archivos .gifs en el repositorio de GitHub [4].

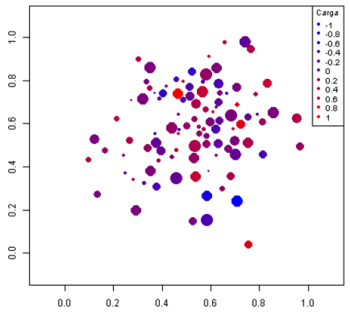


Figura 3: Partículas Carga, Masa, Velocidad(Reto 1).

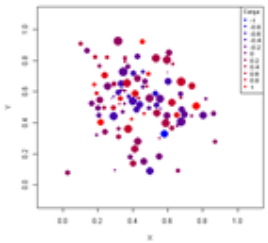


Figura 4: Partículas Carga, Masa, Velocidad(Reto 2).

Cuadro 1: Fuerzas de las n partículas.

Velocidad	Masa	Carga
0,01	0,96	0,02
0.00	0.97	0.20

```
n <- 50
p <- data.frame(x = rnorm(n), y=rnorm(n), c= rnorm(n), m=runif(n,min=.001, max=1))
xmax <- max(p$x)
xmin <- min(p$x)
p$x <- (p$x - xmin) / (xmax - xmin)
ymax <- max(p$y)
ymin <- min(p$y)
p$y <- (p$y - ymin) / (ymax - ymin)
cmax <- max(p$c)
cmin <- min(p$c)
p$c <- 2 * (p$c - cmin) / (cmax - cmin) - 1
p$g <- round(5 * p$c)
paso <- floor(256 / 10)
niveles <- seq(0, 255, paso)
colores <- rgb(niveles, rep(0, 11), rev( niveles), max=255)
eps <- 0.001
fuerza <- function(i) {
  xi <- p[i,]$x
  yi <- p[i,]$y
  ci <- p[i,]$c
  mi <- p[i,]$m
  fx <- 0
  fy <- 0
  for (j in 1:n) {
    cj <- p[j,]$c
    mj <- p[j,]$m
    dir <- (-1)^(1 + 1 * (ci * cj < 0))
    dx <- xi - p[j,]$x
    dy <- yi - p[j,]$y
    if (dir>0) {
      factor <- dir * abs(ci - cj) * ((1+mi * (1+mj)^3)^(1/8)) / (sqrt(dx^2 + dy^2) + eps)
    } else {
      factor <- dir * abs(ci - cj) / (sqrt(dx^2 + dy^2) + eps)
    }
    fx <- fx - dx * factor
    fy <- fy - dy * factor
  }
  return(c(fx, fy))
}
suppressMessages(library(doParallel))
registerDoParallel(makeCluster(detectCores() - 1))
tmax <- 100
digitos <- floor(log(tmax, 10)) + 1
tl <- "0"
while (nchar(tl) < digitos) {
  tl <- paste("0", tl, sep="")
}
plot(p$x, p$y, col=colores[p$g+6], pch=15, cex=1.5, xlim=c(-0.1, 1.1), ylim=c(-0.1, 1.1),
     main="Estado inicial", xlab="X", ylab="Y")
f.iter=matrix(ncol=tmax, nrow=n*2)
for (iter in 1:tmax) {
  f <- foreach(i = 1:n, .combine=c) %dopar% fuerza(i)
  delta <- 0.02 / max(abs(f))
  p$x <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$x + delta * f[c(TRUE, FALSE)][i], 1), 0)
  p$y <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$y + delta * f[c(FALSE, TRUE)][i], 1), 0)
  for(i in 1:(n*2)) { f.iter[i,iter] <- f[i] * delta }
  tl <- paste(iter, "", sep="")
  while (nchar(tl) < digitos) {
    tl <- paste("0", tl, sep="")
  }
  plot(p$x, p$y, col=colores[p$g+6], pch=15, cex=1.5, xlim=c(-0.1, 1.1), ylim=c(-0.1, 1.1),
       main=paste("Paso", iter), xlab="X", ylab="Y")
}
stopImplicitCluster()
f.iter.distance=matrix(ncol=tmax, nrow=n)
f.iter.media ="0"
for (t in 1:tmax) {
  for (i in 1:n) {
    k = i*2
    f.iter.distance[i,t] <- sqrt(f.iter[k,t]^2 + f.iter[k-1,t]^2)
  }
}
for (i in 1:n) {
  p$v[i]<-mean(f.iter.distance[i,])
}
factores<-cbind(p$v,p$m, p$c)
colnames(factores) <- c("Velocidad","Masa", "Carga")
corrplot(cor(factores) )
```

Cuadro 2: Relación de fuerzas.

	Velocidad	Masa	Carga
Velocidad	1,0	0,213	0,13
Masa	0,213	1,0	0,10
Carga	0,13	0,10	1,0

5. Conclusión

La interacción entre partículas con carga y agregando propiedades como la masa pueden afectar directamente el movimiento en la dirección y la velocidad de otras ya que a una mayor masa las partículas tenderán a moverse lentamente debido a su peso pero su fuerza electrostática o fuerza de atracción repulsión será mayor en caso contrario como se ve en las partículas más pequeñas muchas tenderán a moverse más rápidamente para el caso del reto 1 se puede tomar como una reacción de aglomeración en que al sobreponerse o pasar muy cerca de partículas de mayor tamaño estas tenderán a unirse a el caso contrario en el reto dos en el que no existe una superposición a la partícula de mayor masa.

Referencias

[1] E. Schaeffer, "Práctica 9: Interacciones entre partículas," abril 2021. https://elisa.dyndns-web.com/teaching/comp/par/p9.html.

[2] J. J. Allaire, "Rstudio," abril 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 9: Interacciones entre partículas," abril 2021. https://github.com/fuentesadrian/Simulation/tree/master/LocalSearch.

[4] J. A. Garcia abril 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%209.

Práctica 10: Algoritmo genético

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer

Universidad Autónoma de Nuevo León. Facultad de Ingeniería Mecánica y Eléctrica.
30/abril/2021

Resumen

En esta práctica se modifica un algoritmo genético con la finalidad de resolver el problema de la mochila y se compara la mejor solución alcanzada con la solución optima obtenida con un método exacto.

Palabras Claves: *Algoritmo, Knapsack.*

1. Introducción

En la práctica 10 se utiliza el problema de la mochila este es un problema clásico de optimización particularmente de programación entera donde la tarea consiste en seleccionar un subconjunto de objetos de tal forma que no exceden la capacidad de la mochila en términos de la suma de los pesos de los objetos incluidos y que el valor total de los objetos incluidos sea lo máximo posible [1], mediante el problema de la mochila se implementó un algoritmo genético el cual representa posibles soluciones que satisfacen a un problema, creando valores óptimos del problema mediante una buena codificación a las diferentes variables que podrían cuantificarse para dicho método para fines de esta práctica se encontrara representado como un vector de verdadero y falso, indicando cuales objetos vamos a incluir en la mochila ($True$ o 1 llevamos el objeto, $False$ o 0 se descarta el objeto).

2. Objetivo

- Cambiar selección de mutación y de padres para reproducción a que use selección de ruleta [1].
- Genere instancias con tres distintas reglas [1].
- Determinar para cada uno de los tres casos a partir de qué tamaño de instancia el algoritmo genético es competitivo con el algoritmo exacto [1].

3. Metodologia

La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la *Práctica 10: Algoritmo genético* [1], a partir del código en el repositorio de Schaeffer [3], se realizaron modificaciones. El código completo de la metodología empleada se encuentra en el repositorio de GitHub [4].

4. Resultados

Simulando la aplicación del problema de la mochila, conocido en ingles como *Knapsack problem*, donde se tienen un contenedor de capacidad C limitada y se tienen N elementos que se pueden meter en el contenedor , cada elemento tiene un valor de beneficio b_i y un valor de capacidad c_i que ocupan en el contenedor, se busca tener en el contenedor aquellos que nos maximizan el beneficio máx , donde x_i es la decisión de tener o no el elemento en el contenedor sin exceder la capacidad C.

A continuación se muestra la función de generador de valores que fue modificada.

```
generador.valores <- function(cuantos, min,
    max) {
  return(sort(round(normalizar(rnorm(cuantos))
    * (max - min) + min)))
}
```

Se realizaron cambios para la creación de variables.

```
p1 <- poblacion.inicial(n, init)
p<- p1
tam <- dim(p)[1]
assert(tam == init)
pm <- sum(runif(tam) < 0.05)
rep <- 50
tmax <- 50
mejorescon <- double()
```

Se crean los vectores de factibilidad y objetivos.

```
tam <- dim(p)[1]
  obj <- double()
  fact <- integer()
  for (i in 1:tam) {
    obj <- c(obj, objetivo(p[i,], valores))
    fact <- c(fact, factible(p[i,], pesos,
    capacidad))
  }
```

Se generan las mutaciones.

```
  for (i in 1:tam) {
    prob.mut[i] = 1/(obj[i]*(fact[i]+1)*sum(
    obj*(fact+1)))
  }
  mutantes<-sample(1:tam, pm, prob = prob.mut)
  for (i in mutantes) {
    p <- rbind(p, mutacion(p[i,], n))
  }
```

Se llevan acabo las reproducciones.

```
for (i in 1:tam) {
    prob.reprod[i] = obj[i]*(fact[i]+1)/sum(
    obj*(fact+1))
  }

```

Creación de variables y fabricación de las interacciones del algoritmo genético sin ruleta, aplicando una probabilidad de mutación, se determina una cantidad fija de reproducciones.

```
mejorescon <- c(mejorescon, mejor)
}
p<- p1
mejoressin <- double()
for (iter in 1:tmax) {
  p$obj <- NULL
  p$fact <- NULL
  for (i in 1:tam) {
    if (runif(1) < pm) {
      p <- rbind(p, mutacion(p[i,], n))
    }
  }
  for (i in 1:rep) {
    padres <- sample(1:tam, 2, replace=FALSE)
    hijos <- reproduccion(p[padres[1],], p[
    padres[2],], n)
    p <- rbind(p, hijos[1:n])
    p <- rbind(p, hijos[(n+1):(2*n)])
  }
  tam <- dim(p)[1]
  obj <- double()
  fact <- integer()
  for (i in 1:tam) {
    obj <- c(obj, objetivo(p[i,], valores))
    fact <- c(fact, factible(p[i,], pesos,
    capacidad))
  }
  p <- cbind(p, obj)
  p <- cbind(p, fact)
  mantener <- order(-p[, (n + 2)], -p[, (n +
    1)])[1:init]
  p <- p[mantener,]
  tam <- dim(p)[1]
  assert(tam == init)
  factibles <- p[p$fact == TRUE,]
  mejor <- max(factibles$obj)
  mejoressin <- c(mejoressin, mejor)
}
```

Los resultados obtenidos al correr el código se muestran en la figura 1 se analizan los valores óptimos así como el mayor valor alcanzado en la tabla 1.

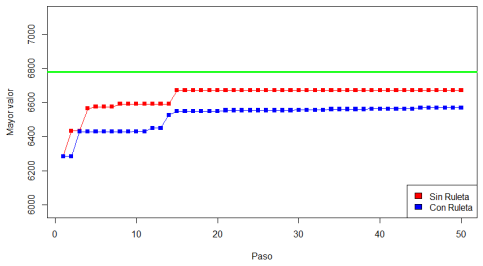


Figura 1: Gráfico competitivo con ruleta y sin ruleta de la primera instancia.

Tabla 1: Mayor valor alcanzado vs valor optimo

Mejor	Optimo	Porcentaje
6672	6780	0,01

Ahora en base al primer código modificado se realizaron ligeras adecuaciones con el fin de cumplir con los objetivos de la práctica y se generen distintas reglas, los cambios realizados se muestran a continuación y se muestra en la figura 2, se analizan los valores óptimos así como el mayor valor alcanzado en la tabla 2, se observa una mejora en los valores óptimos aplicando la ruleta y sin ruleta en comparación con la capacidad máxima.

```
generador.pesos <- function(valores, min, max)
    {
  n <- length(valores)
  pesos <- double()
  for (i in 1:n) {
    media <- valores[i]
    desv <- runif(1, max=.1)
    ruido <- rnorm(1, sd=.1)
    pesos <- c(pesos, rnorm(1, (1/media), desv
    ) + ruido)
  }
  pesos <- normalizar(pesos) * (max - min) +
    min
  return(pesos)
}
```

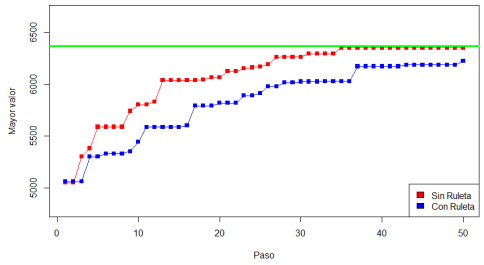


Figura 2: Gráfico competitivo con ruleta y sin ruleta de la segunda instancia.

Tabla 2: Mayor valor alcanzado vs valor optimo

Mejor	Optimo	Porcentaje
6352	6367	0,002

Con el fin de cumplir con la tercera y ultima instancia se modifica la función generador de valores observándose valores menos favorables debido aque el porcentaje es menor en comparación con los valores anteriores, los datos obtenidos se muestran en la tabla 3 y se representan en la figura 3.

```
generador.valores <- function(pesos, min, max) {
  n <- length(pesos)
  valores <- double()
  for (i in 1:n) {
    media <- pesos[i]
    desv <- runif(1)
    ruido <- rnorm(1, sd=.1)
    valores <- c(valores, rnorm(1, media^2, desv) + ruido)
  }
  valores <- normalizar(valores) * (max - min) + min
  return(valores)
}
```

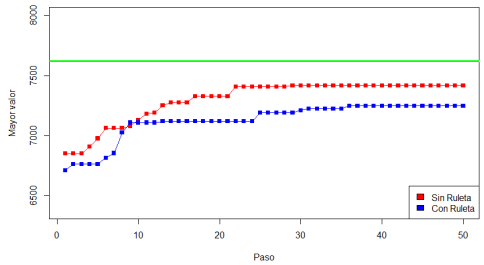


Figura 3: Gráfico competitivo con ruleta y sin ruleta de la tercera instancia.

Tabla 3: Mayor valor alcanzado vs valor optimo

Mejor	Optimo	Porcentaje
7417,10	7620,28	0,02

5. Reto 1

Extender la selección de ruleta a la fase de supervivencia: en vez de quedarnos con las mejores soluciones, cada solución tiene una probabilidad de entrar a la siguiente generación que es proporcional a su valor de la función objetivo, incorporando el sí o no es factible la solución en cuestión, permitiendo que los k mejores entre las factibles entren siempre (donde k es un parámetro). Estudia nuevamente el efecto de este cambio en la calidad de la solución en los tres casos.

```
prob.supervivencia <- NULL


for (i in 1:tam) {
    prob.supervivencia[i] = obj[i]*(fact[i]+1)
    /sum(obj*(fact+1))
  }
  supervivientes<-sample(1:tam,init,prob =
    prob.supervivencia)
  p <- p[supervivientes,]

```

El código fue modificado para cada una de las 3 instancias anteriores realizadas en la tarea base las figuras 4, 5, 6 muestran los valores obtenidos respectivamente, comparándolos con la tarea base observamos que los resultados son muy variables sin embargo la función de generador de pesos nos arroja un valor ideal para los casos con ruleta y sin ruleta, los valores se representan en la tabla 4.

Tabla 4: Mayor valor alcanzado vs valor optimo

	Mejor	Optimo	Porcentaje
fig.4	8982	9417	0,04
fig.5	4318	4939	0,12
fig.6	6476	7258	0,1

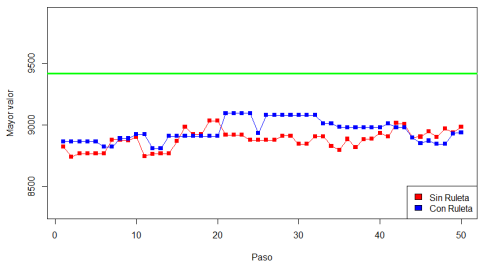


Figura 4: Gráfico competitivo con probabilidad de entrar ala siguiente generación.

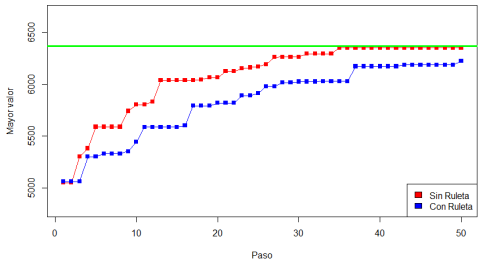


Figura 5: Gráfico competitivo con probabilidad de entrar ala siguiente generación.

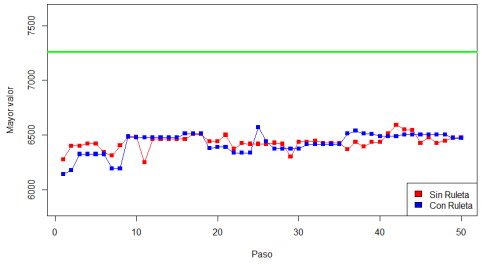


Figura 6: Gráfico competitivo con probabilidad de entrar ala siguiente generación.

6. Reto 2

Paraleliza el algoritmo genético y estudia los efectos en su tiempo de ejecución con pruebas estadísticas y visualizaciones, variando el número de objetos en la instancia.

```
  cluster <- makeCluster(detectCores() - 1)
clusterExport(cluster,"factible")
clusterExport(cluster,"objetivo")
clusterExport(cluster,"normalizar")
clusterExport(cluster,"poblacion.inicial")
clusterExport(cluster,"mutacion")
clusterExport(cluster,"reproduccion")
datos=data.frame()
for(init in c(50,100,200))

```

```
clusterExport(cluster,"n")
  clusterExport(cluster,"capacidad")
  clusterExport(cluster,"init")
  clusterExport(cluster,"pesos")
  clusterExport(cluster,"valores")

```

A partir de la página RDocumentation [5] se encontraron algunas funciones con el fin de paralelizar el código de Scaeffer [6], sin embargo al correr el código se encuentra un error en la interpretación de la imagen.

```
      clusterExport(cluster,"p")
      mutan=sample(1:tam,round(pm*tam))
      p <- rbind(p,(t(parSapply(cluster,
  mutan,function(i){return(mutacion(unlist(p
  [i,]), n))}))))
      clusterExport(cluster,"tam")
      clusterExport(cluster,"p")
      padres <- parSapply(cluster,1:rep,
  function(x){return(sample(1:tam, 2,
  replace=FALSE))})
      clusterExport(cluster,"padres")
      hijos <- parSapply(cluster,1:rep,
  function(i){return(as.matrix(unlist(
  reproduccion(p[padres[1,i],], p[padres[2,i
  ],], n)),ncol=n))})
      p = rbind(p,hijos)
      tam <- dim(p)[1]
      clusterExport(cluster,"p")
        obj=parSapply(cluster,1:tam,function
  (i){return(objetivo(unlist(p[i,]), valores
  ))})
       fact=parSapply(cluster,1:tam,
  function(i){return(factible(unlist(p[i,]),
   pesos, capacidad))})

```

7. Conclusión

Al realizar cambios como la selección de mutación y de padres para la reproducción a que usen selección de ruleta y cambiar las instancias se observa que el mejor valor alcanzado variara, se podría deducir que el algoritmo genético se puede mejorar al paralelizarlo, esto sé notaria en los tiempos de ejecución donde el tiempo de ejecución del código paralelizado será menor que el código secuencial.

Referencias

[1] E. Schaeffer, "Práctica 10: algoritmo genético," mayo 2021. https://https://elisa.dyndns-web.com/teaching/comp/par/p10.html.

[2] J. J. Allaire, "Rstudio," mayo 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 10: algoritmo genético," mayo 2021. https://github.com/satuelisa/Simulation/tree/master/GeneticAlgorithm.

[4] J. A. Garcia mayo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%2010.

[5] Datacamp, "clusterapply aplicar operaciones mediante clústeres," mayo 2021. https://www.rdocumentation.org/packages/parallel/versions/3.6.2/topics/clusterApply.

[6] E. Schaeffer, "Práctica 10: algoritmo genético," mayo 2021. https://github.com/fuentesadrian/Simulation/tree/master/MultiAgent.

Práctica 11: Frentes de Pareto

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer

Universidad Autónoma de Nuevo León. Facultad de Ingeniería Mecánica y Eléctrica.
09/mayo/2021

Resumen

Se realizara una optimización de multicriterio implementando un frente de pareto con la finalidad de definir un conjunto de soluciones que son buenas al considerar todos los objetivos, determinando los puntos con mejor compromiso en cada función.

Palabras Claves: *pareto, multicriterio, polinomios.*

1. Introducción

En muchas ocasiones es muy difícil tomar decisiones, sobre todo cuando tenemos varios objetivos para alcanzar y no nos decidimos cuál es la mejor ruta para obtener el óptimo desempeño, por lo cual existe una herramienta llamada frentes de pareto que nos permite contribuir en la mejor optimización de un objetivo. Muchos problemas reales deben ser predecibles al mismo tiempo y en ocasiones de forma conjunta, sin embargo, es muy común que los objetivos se encuentren en conflictos entre sí, para llegar al objetivo hay que considerar ciertos conceptos que nos permiten elegir las condiciones necesarias para obtener un mejor resultado, tales como la simplicidad, en muchas ocasiones los problemas tienden a modelarse para cumplir con un solo. Cuando se intentan optimizar simultáneamente múltiples objetivos, estos se contraponen entre sí, pues mientras uno mejora los otros empeoran y viceversa. Si analizamos las soluciones de estos problemas y sus evaluaciones es fácil notar la dificultad de comparar en vía de encontrar la mejor. Para estudiar este problema, vamos a primero implementar un generador de polinomios aleatorios [1]. Estos polinomios los utilizaremos como funciones objetivo. Vamos a permitir solamente una variable por término y un término por grado por variable. Para simplicidad de visualización, vamos a concentrarnos en el caso de dos funciones objetivo [1]. El análisis busca mostrar hasta cuantas funciones objetivo tiene sentido considerar, cómo encontrar un subconjunto de soluciones no dominadas que sea diverso a partir de un frente dado y como encontrar el frente de pareto. Pues bien, existen soluciones que en sí misma cumplen la definición de optimas al poder asegurar que no existe solución que sea mejor que ella misma; sin embargo, también existen soluciones que no se pueden comparar entre sí. Este conjunto de soluciones es mejor conocido como soluciones eficientes y su evaluación como frente de pareto o conjunto de soluciones no dominadas.

2. Objetivo

- Grafica el porcentaje de soluciones de Pareto como función del número de funciones objetivo para $k \epsilon [2, 8]$ en pasos de dos con diagramas de violín combinados con diagramas de caja-bigote [1].
- Razona en escrito a qué se debe el comportamiento observado [1].

3. Metodologia

La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la *Práctica 11: Frentes de Pareto* [1], a partir del código en el

repositorio de Schaeffer [3], se realizaron modificaciones. El código completo de la metodología empleada se encuentra en el repositorio de GitHub [4].

4. Resultados

Con la finalidad de poner un numero mínimo de réplicas se añade el valor de 30 a r posteriormente se agregan las funciones objetivo, se crean los polinomios y las soluciones son evaluadas para todos los objetivos.

```
r <- 30
vectorfunciones <- c(2,4,6,8)
poncentajefrente<-matrix(0, nrow=r, ncol=
    length(vectorfunciones))
for(f in vectorfunciones) {
k <- f
for (q in 1:r) {
for (i in 1:k) {
  obj[[i]] <- poli(md, vc, tc)
}
minim <- (runif(k) > 0.5)
sign <- (1 + -2 * minim)
n <- 200
sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
for (i in 1:n) {
  for (j in 1:k) {
    val[i, j] <- eval(obj[[j]], sol[i,], tc)
  }
}
```

Se obtiene el porcentaje de cada frente entre soluciones y es graficado, se añaden pruebas estadísticas para comparar las medias obteniendo resultados estadísticamentes diferentes entre las diferentes cantidades de funciones objetivo.

```
poncentajefrente[q,(f/2)]  <- (length(frente
    [,1])/n)*100
}
}
frentes<-c(poncentajefrente[,1],
    poncentajefrente[,2], poncentajefrente
    [,3], poncentajefrente[,4])
funciones<-c(rep(2, r), rep(4, r), rep(6, r),
    rep(8, r))
funciones<-as.factor(funciones)
data2 <- data.frame(pos=funciones, dom=frentes
    )
g2 <- ggplot(data2, aes(x=pos, y=dom)) +
    geom_violin(fill="orange", color="red")
g2 + geom_boxplot(width=0.2, fill="blue",
    color="green", lwd=2) +
  xlab("") +
  ylab("Frecuencia") +
  ggtitle("Porcentaje de Frentes de Pareto
    entre las Soluciones Totales")
t.test(poncentajefrente[,1], poncentajefrente
    [,2])$p.value
t.test(poncentajefrente[,1], poncentajefrente
    [,3])$p.value
t.test(poncentajefrente[,1], poncentajefrente
    [,4])$p.value
t.test(poncentajefrente[,2], poncentajefrente
    [,3])$p.value
t.test(poncentajefrente[,2], poncentajefrente
    [,4])$p.value
t.test(poncentajefrente[,3], poncentajefrente
    [,4])$p.value
```

Basándonos en los resultados del "violín plot"(figura 1) podemos observar que mientras mayor sea el numero de funciones objetivo, mayor sera el porcentaje de frentes de pareto entre la cantidad de soluciones existentes. Se pudiera inferir que este resultado tiene como causa que al haber muchos funciones que se tienen que optimizar, las soluciones existentes si bien no cumplen con algunas funciones en específico, pudieran estar optimizando en gran manera alguna otra, en cierta manera lo veo de esta forma, cada solución al menos le tiene que gustar a alguna función,

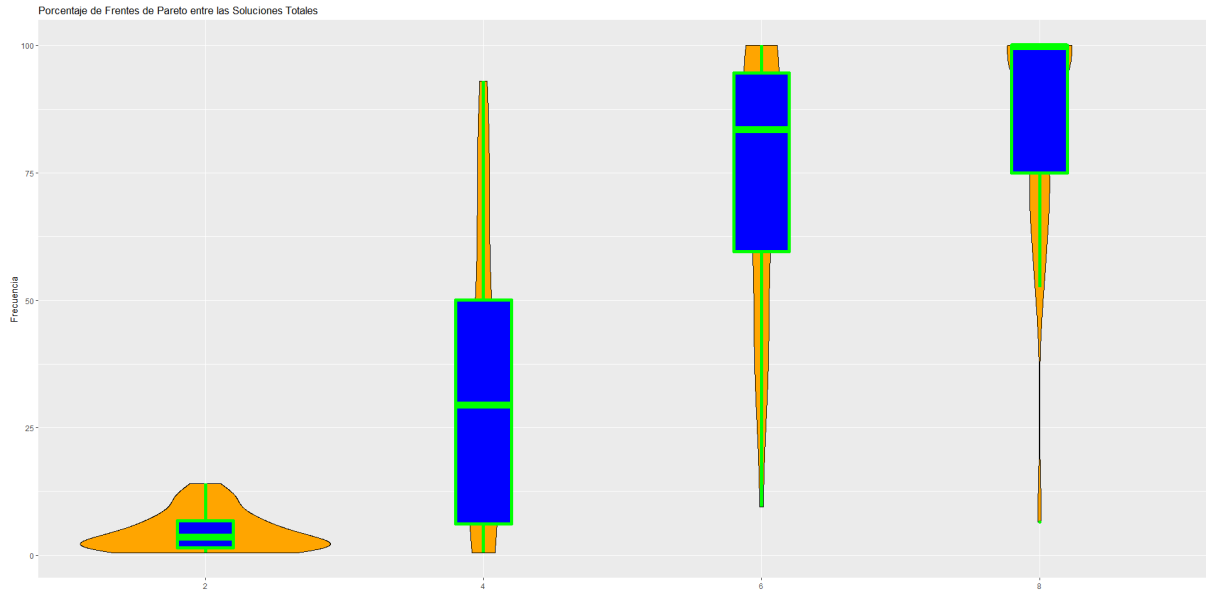


Figura 1: Porcentaje de frentes de pareto entre las soluciones totales.

$3{,}959976e-06$
$3{,}450318e-14$
$1{,}729815e-26$
$2{,}321364e-05$
$1{,}711739e-12$
$2{,}845312e-05$

Cuadro 1: % Porcentaje estadístico de pareto

En el cuadro 1 se muestran los valores estadisticos en cuanto a comparacion de las medias del porcentaje de frente de pareto

5. Reto 1

Seleccionar un subconjunto (cuyo tamaño como un porcentaje del frente original se proporciona como un parámetro) del frente de Pareto de tal forma que la selección esté diversificada, es decir, que no estén agrupados juntos en una sola zona del frente las soluciones seleccionadas. Graficar los resultados de la selección, indicando con un color cuáles se incluyen en el subconjunto diverso.

```
distancia <- function(frente) {
  m <- length(frente[,1])
  k <- length(frente[1,])
  distgrandes <- matrix(0, nrow=m, ncol=m)
  for (p1 in 1:m) {
    coordenadas1<-frente[p1,]
    dist<-c()
    t=1
    for (punto2 in 1:m) {
      if (p1 != punto2) {
      coordenadas2<-frente[punto2,]
      dist[t] <-sqrt(sum((coordenadas2-
    coordenadas1)^2))
      t=t+1
      }
    }
  if (p1==1) { dist <- c(max(dist), dist)
  } else if (p1 == m) { dist <- c(dist, max(
    dist))
  } else { dist<-c(dist[1:(p1-1)],max(dist),
    dist[p1:length(dist)])
  }
  distgrandes[p1,] <-dist > max(dist)*1
  }
  for (i in 1:m) {
    tope<-length(distgrandes[,1])
    if (tope>m) {
    distgrandes<-subset(distgrandes,
    distgrandes[i,] > 0)
    }
  }
  return(distgrandes)
}
length(distancia(frente)[,1])
```

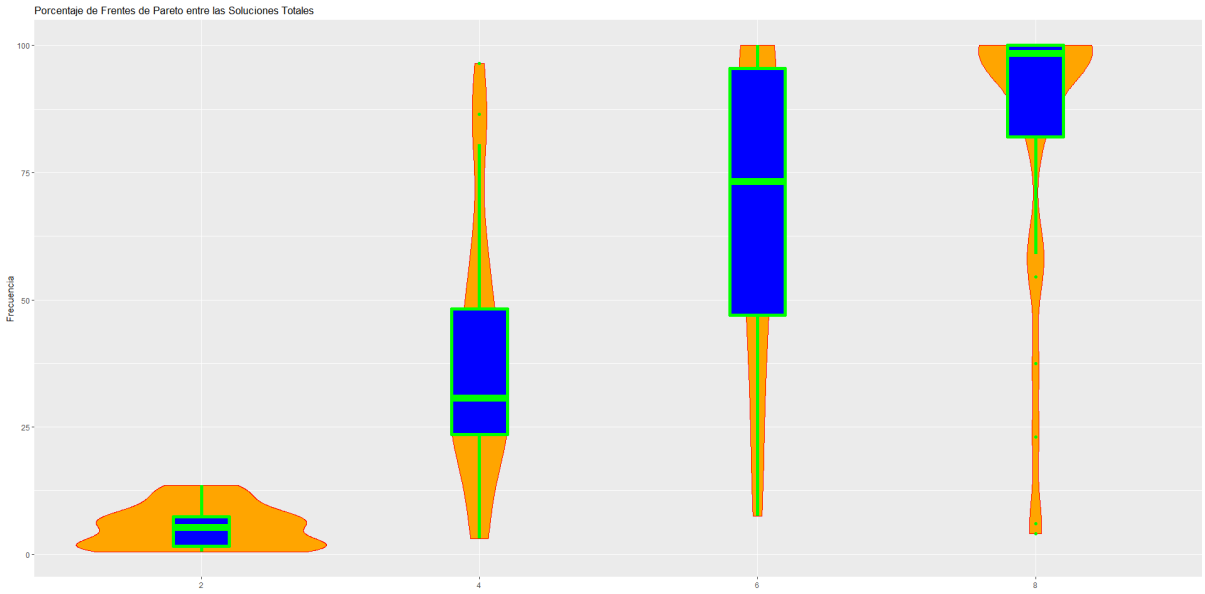


Figura 2: Porcentaje de frentes de pareto (subconjunto).

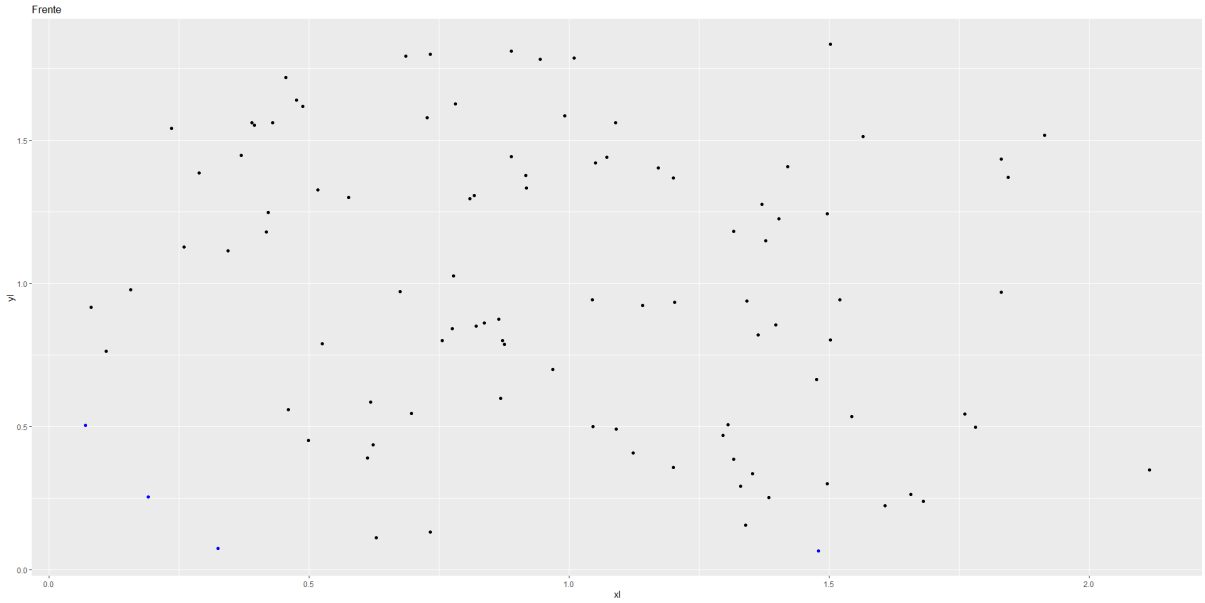


Figura 3: Frentes de pareto .

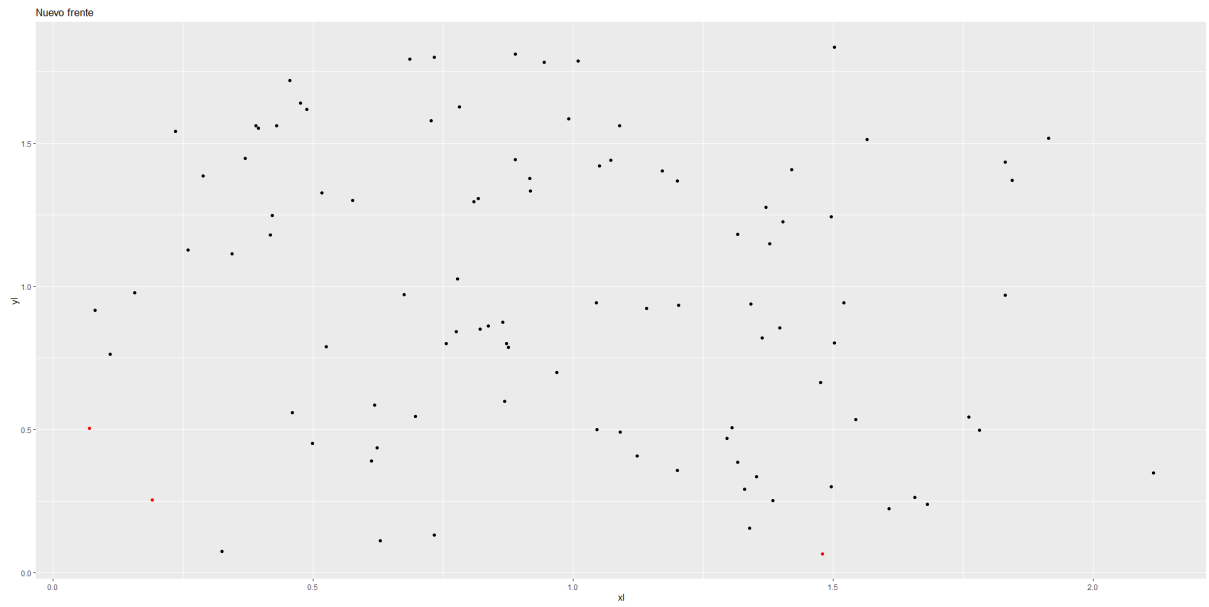


Figura 4: Frentes de pareto (puntos rojos).

En la figura 3 se muestra un frente de pareto sin embargo lo que se busca es de todas las mejores opciones mostrar solo algunas con la finalidad de que no estén agrupadas tal como se muestra en la figura 4 en comparación con la tarea base la figura 2 nos indica un porcentaje similar al de la figura 1

6. Reto 2

Adaptar el algoritmo genético de la tarea anterior para que vaya buscando mejora a un frente; la población inicial es el frente generado en la tarea y se aplica la diversificación del primer reto a cada generación después de los cruzamientos y las mutaciones. Visualiza con un GIF animado cómo avanza la frente de una generación a otra.

7. Conclusión

La mayoría de las pruebas realizadas muestran que las medias son estadísticamentes diferentes entre las diferentes cantidades de funciones objetivo basándonos en los resultados se observa que mientras mayor sea el numero de funciones objetivo, mayor sera el porcentaje de frentes de pareto entre la cantidad de soluciones existentes.

Referencias

[1] E. Schaeffer, "Práctica 11: frentes de pareto," mayo 2021. https://https://elisa.dyndns-web.com/teaching/comp/par/p11.html.

[2] J. J. Allaire, "Rstudio," mayo 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 11: frentes de pareto," mayo 2021. https://github.com/satuelisa/Simulation/tree/master/ParetoFronts.

[4] J. A. Garcia mayo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%2011.

Práctica 12: red neuronal

Alumno: José Adrian Garcia Fuentes
Profesor: Satu Elisa Schaeffer

Universidad Autónoma de Nuevo León. Facultad de Ingeniería Mecánica y Eléctrica.
16/mayo/2021

Resumen

Reconocer dígitos de imágenes pequeñas en blanco y negro con una red neuronal, utilizando un perceptrón que separe las entradas verdaderas y falsas.

Palabras Claves: *Red neuronal.*

1. Introducción

La última práctica es una demostración básica de aprendizaje a máquina: se requiere reconocer dígitos de imágenes pequeñas en blanco y negro con una red neuronal. El elemento básico de una red neuronal es un perceptrón que esencialmente es un hiperplano (una línea si nos limitamos a dos dimensiones) que busca colocarse en la frontera que separa las entradas verdaderas y las entradas falsas. La dimensión d del perceptrón es el largo del vector x que toma como entrada, y su estado interno se representa con otro vector w que contiene sus pesos [1].

2. Objetivo

Estudia de manera sistemática el desempeño de la red neuronal en términos de su puntaje F para los diez dígitos en función de las tres probabilidades asignadas a la generación de los dígitos, variando a las tres en un experimento factorial adecuado [1].

3. Metodologia

La metodología empleada se realizó a través de Rstudio [2] llevando a cabo los pasos señalados en la *Práctica 12: red neuronal* [1], a partir del código en el repositorio de Schaeffer [3], se realizaron modificaciones. El código completo de la metodología empleada se encuentra en el repositorio de GitHub [4].

4. Resultados

A continuación se muestran los cambios al codigo, en la figura 1 se muestra un dígito en blanco y negro.

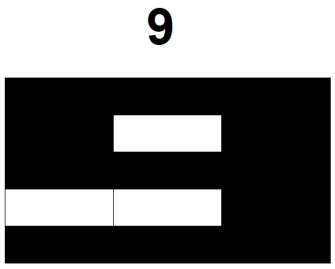


Figura 1: Dígito blanco y negro 3 x 5.

```
n_prob <- c(0.995, 0.98, 0.95, 0.30)
g_prob <- c(0.920, 0.90, 0.85, 0.50)
b_prob <- c(0.002, 0.01, 0.05, 0.80)

prom_fscore <- c()

for (iter in 1:4) {
```

```
  modelos <- read.csv("digits.txt", sep=" ",
    header=FALSE, stringsAsFactors=F)
  modelos[modelos=='n'] <- n_prob[iter]
  modelos[modelos=='g'] <- g_prob[iter]
  modelos[modelos=='b'] <- b_prob[iter]
```

```
precision <- diag(contadores) / rowSums(
    contadores)
  recall <- diag(contadores) / colSums(
    contadores)[1:(ncol(contadores)-1)]
  fscore <- (2 * precision * recall) / (
    precision + recall)
  prom_fscore[iter] <- mean(fscore)
}
prom_fscore
```

5. Reto 1

Extiende y entrena la red neuronal para que reconozca además por lo menos doce símbolos ASCII adicionales, aumentando la resolución de las imágenes a 5 x 7 de lo original de 3 x 5 (modificando las plantillas de los dígitos acorde a este cambio).

Se modificó el codigo de la tarea base cambiando las dimensiones de la resolución de la imagen a 7 x 5 (ver figura 2, 3, 4).

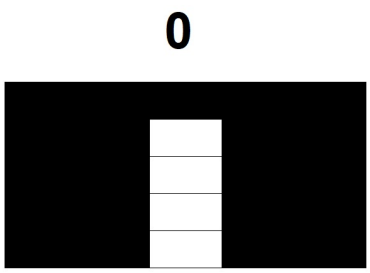


Figura 2: Dígito blanco y negro 7 x 5.

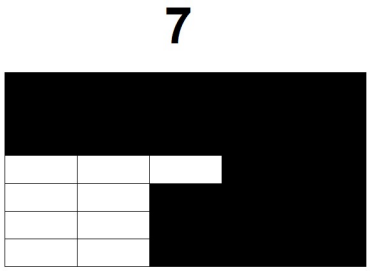


Figura 3: Dígito blanco y negro 7 x 5.

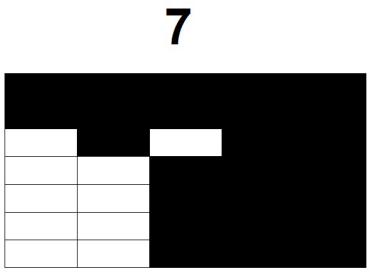


Figura 4: Dígito blanco y negro 7 x 5.

```
modelos <- read.csv("digitos2.csv", sep=",",
   header=FALSE, stringsAsFactors=F)
modelos[1,1] <- "n"

modelos[modelos=='n'] <- 0.995
modelos[modelos=='g'] <- 0.92
modelos[modelos=='b'] <- 0.002

r <- 7
c <- 5
dim <- r * c

tasa <- 0.15
tranqui <- 0.99

tope <- 11

```

6. Reto 2

Agrega ruido *sal-y-pimienta* en las entradas para una combinación ngb con la cual la red desempeña bien; este tipo de ruido se genera cambiando con una probabilidad los pixeles a blanco o negro (uniformemente al azar entre las dos opciones). Estudia el efecto de en el desempeño de la red (no importa si se hace esto con la red de la tarea base o la red extendida del primer reto).

A partir del código del primer reto se añadió el siguiente código para estudiar el efecto de desempeño en la red y se obtuvo el gráfico *fscore* de *pr* (ver figura 5 un ejemplo de dígito se muestra en la figura 6

```
 pr <- seq(0,.5,.025)
prom_fscore <- c()
iter = 1

for (prob in pr) {
```

```
s <- (runif(dim) > prob)
    for (m in 1:length(s)) {
      if (s[m] == FALSE) {pixeles[m]<- !
    pixeles[m]}
    }
    correcto <- binario(d, n)
```

```
iter = iter + 1
}
prom_fscore
plot(pr, prom_fscore, type= "l")
```

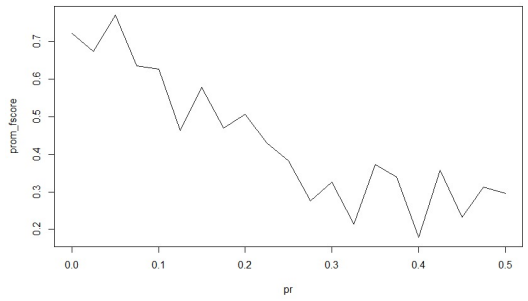


Figura 5: Gráfico f score.

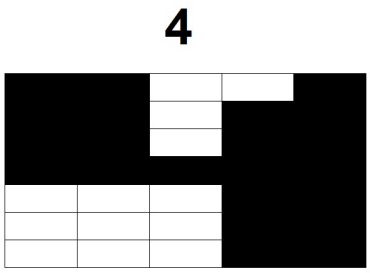


Figura 6: Dígito blanco y negro 3 x 5.

7. Conclusión

El valor *Fscore* depende de la probabilidad con la que aparezcan los colores y estos pixeles dominantes predominan los colores negro y blanco.

Referencias

[1] E. Schaeffer, "Práctica 12: red neuronal," mayo 2021. https://https://elisa.dyndns-web.com/teaching/comp/par/p12.html.

[2] J. J. Allaire, "Rstudio," mayo 2021. https://rstudio.com.

[3] E. Schaeffer, "Práctica 12: red neuronal," mayo 2021. https://github.com/satuelisa/Simulation/tree/master/NeuralNetwork.

[4] J. A. Garcia mayo 2021. https://github.com/fuentesadrian/SIMULACION-DE-NANOMATERIALES/tree/main/Tarea%2012.

Modelado del fenómeno de maduración de Ostwald

Garcia Fuentes J. A.[a,b,*,1]
Correo electrónico: jose.garciafnt@uanl.edu.mx[*1]

[a] *Maestría en Ciencias de la Ingeniería con Orientación en Nanotecnología*

[b] *Facultad de Ingeniería Mecánica y Eléctrica, Universidad Autónoma de Nuevo León*

Resumen

Se muestra un descripción de los sistemas que tienden a descomponerse mediante una variedad de mecanismos fisicoquímicos enfocando principalmente al fenómeno de maduración de Ostwald, se presenta un modelo del fenómeno tomando en cuenta el Movimiento Browniano que tienen las partículas, así como también una carga electrostática y una masa que influyen en el tamaño de estas de acuerdo al fenómeno.

Palabras clave: Floculación, Emulsión, Carga, Browniano, Simulación.

1. Introducción

Las emulsiones son sistemas termodinámicamente desfavorables que tienden a descomponerse con el tiempo debido a una variedad de mecanismos fisicoquímicos (figura 1), incluida la separación gravitacional, la floculación, la coalescencia y la maduración de Ostwald [1]. El engrosamiento de la textura (también conocido como maduración de Ostwald, equilibrio o maduración de la textura) se ha propuesto como un proceso importante en rocas plutónicas y algunas rocas volcánicas. En un contexto ígneo, el engrosamiento es la reabsorción de cristales pequeños y el crecimiento simultáneo de cristales más grandes como un proceso de minimización de la energía superficial total (libre). Aunque no existe consenso sobre el proceso exacto de engrosamiento de la textura, se utiliza la teoría de Comunicando Vecinos y la teoría Lifshitz-Slyozov-Wagner [2]. Durante la maduración de Ostwald , se consumen cristales dispersos por debajo de un radio crítico, mientras que los granos mayores continúan creciendo a expensas de los más pequeños el radio critico no es una constante, sino que aumenta con el tiempo. Esto se debe a que el sistema no estará en un equilibrio termodinámico estable hasta que todo el material de un determinado mineral se reúna en un solo cristal, o al menos hasta que todos los granos en un determinado dominio rocoso tengan aproximadamente el mismo tamaño [3].

Una vez que cesó la nucleación y/o el crecimiento del sistema abierto, quizás debido al agotamiento de un determinado elemento en el reservorio, no se formarán nuevos cristales y las poblaciones existentes pueden continuar creciendo de manera que minimizan su energía libre superficial [3]. En la mayoría de las muestras, la evolución temporal alcanza rápidamente un estado estable. Sin embargo algunas muestras como la plagioclasa en fundidos de silicato, el tamaño de los cristales se amortigua rápidamente, ya que en los fundidos de silicato se ha interpretado principalmente como una maduración controlada por un proceso de nucleación superficial en la interfaz cristal-líquido [4]. Para algunos modelos se pueden mencionar tres etapas basicas en la maduración empezando por una evolución microestructural durante la transformación de fase. La nucleación heterogénea inicial a partir del grano en descomposición da como

resultado un intercrecimiento a lo largo de los límites del grano. En la naturaleza, este intercrecimiento se conoce como simplicidad de grano. Sin embargo, en algunos casos las escalas de tiempo de engrosamiento son largas, lo que resulta en un engrosamiento del borde de reacción. Suponiendo que el sistema avanza de una manera lenta como para que la micro o nano estructura avance hacia el equilibrio, el crecimiento se lleva acabo a través de uniones triples. Finalmente, si el sistema avanza más hacia el equilibrio, los pequeños granos intersticiales serán absorbidos por vecinos más grandes [5].

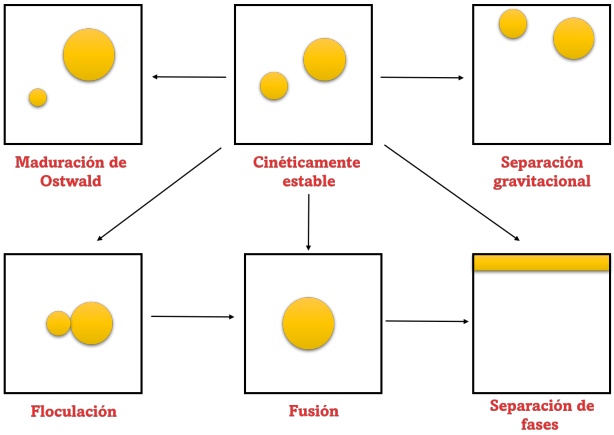


Figura 1: Diagrama esquemático inspirado en McClements (2015) [1] de los mecanismos de inestabilidad más comunes que ocurren en los sistemas de administración coloidal: separación gravitacional, floculación, coalescencia, maduración de Ostwald e inversión de fase.
Fuente: D. J McClements (2015). Elaboración propia.

La superposición rítmica de escala fina es otra variedad de superposición difusa de este tipo. Normalmente se encuentra en una escala de unos pocos centímetros y se cree que se debe a un proceso de reequilibrio en el que pequeñas diferencias iniciales de tamaño de grano o proporciones modales se acentúan y repiten por la solución cíclica y el crecimiento de cristales en condiciones similares a las de Maduración de Ostwald [6].

En el presente trabajo se planea realizar una simulación con modelado del fenómeno de maduración de Ostwald empleando los modelos de Movimiento Browniano, y adecuando a cada partícula una carga y una masa, el modelado se describe en mayor detalle en la sección de resultados y discusión.

2. Maduración de Ostwald

La maduración de Ostwald se debe al crecimiento de las gotas más grandes a costa de las más pequeñas, hasta que estas últimas prácticamente desaparecen. El proceso va de acuerdo con una velocidad que está en función de la solubilidad de la fase, dispersa en la fase continua y se debe a que la presión interna de las gotas, la presión es mayor a las gotas más pequeñas [7]. La maduración de Ostwald estará ocurriendo tan pronto como las interfaces curvas están presentes, ya que esta curvatura causa alta solubilidad desde la fase dispersa en el límite de la partícula comparada con la fase volumétrica proxima. A las partículas grandes el gradiente de concentración de la fase, dispersa en la fase continua, provoca partículas grandes que crecen a expensas de las más pequeñas (figura 2) [8]. La maduración se dirige por la diferencia de la presión entre gotas que tiene diferentes radio, por lo que la fase dispersa se difunde de las gotas más pequeñas a las de mayor tamaño, la transferencia de masa en emulsiones no sólo se dirige por la diferencia en la curvatura de las gotas, sino también por la diferencia de sus composiciones [7]. Algunos autores mencionan los efectos muy menores en los cristales de más de $20\mu m$. De acuerdo con esto, estas porciones más gruesas del umbral no se distinguen por densidades de población especialmente altas en las clases de tamaño de cristal más grande. Más bien, tienen densidades de población especialmente bajas en las clases de tamaño de cristal pequeño [9].

Este proceso espontáneo, controlado termodinámicamente [10], se produce porque las partículas más grandes están energéticamente favorecidos con respecto a las más pequeñas. Esto proviene del hecho de que las moléculas en la superficie de una partícula son energéticamente menos estables que los que están en el interior. Si todas las partículas pequeñas hacen esto, la concentración de moléculas libres en la solución aumenta. Cuando las moléculas libres en solución se sobresatura, las moléculas libres tienen una tendencia a condensarse sobre la superficie de las partículas

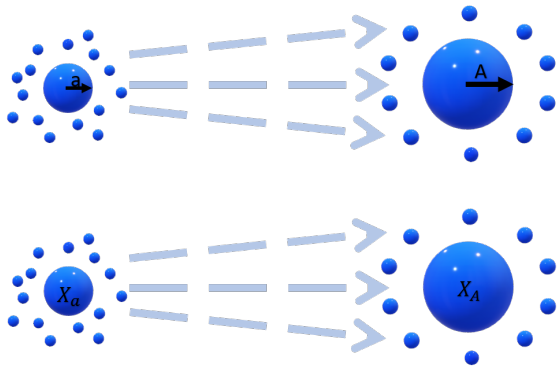


Figura 2: Diagrama esquemático inspirado en A. S. Kabalnov (1994) [8] muestra la trasferencia de masa en emulsiones mixtas.
Fuente: A. S. Kabalnov (1994). Elaboración propia.

más grandes. Por lo tanto, las partículas más pequeñas se reducen, mientras que las partículas grandes crecen, y en general el tamaño medio irá en aumento. Después de un tiempo determinado, la población total de partículas se convierte en una sola partícula esférica grande minimizando la superficie total.

2.1. Agregación y engrosamiento

Tanto la agregación como el engrosamiento son procesos estrechamente relacionados de importancia en la formación de acumulados. La agregación es un término general que describe la agrupación de granos en una masa fundida. En algunos casos, los cristales parecen estar adheridos a lo largo de caras cristalográficamente similares, lo que garantiza que se minimice el desajuste estructural y la energía interfacial. Por el contrario, también se ha propuesto que el crecimiento se ve reforzado por la anexión de granos pequeños por cristales más grandes, seguida de la migración de los límites de los granos [2].

2.2. Disolución-reprecipitación en la masa de cristales

Con subenfriamientos bajos, la distribución del tamaño de grano de una suspensión polidispersa dentro de un líquido evoluciona bajo la influencia de energías interfaciales. La mayor energía libre de los cristales pequeños en relación con los cristales más grandes significa que los granos grandes crecen a expensas de los más pequeños. El efecto del tamaño sobre la solubilidad equivale a un sobrecalentamiento efectivo de los cristales más pequeños. Este proceso tiene un efecto insignificante si los cristales están muy separados, particularmente para granos de más de unas pocas micras, pero ocurre a una velocidad mayor cuando los cristales se tocan, ya que los granos pueden coalescer o si hay ciclos térmicos. La pérdida de granos pequeños también puede ocurrir por disolución preferencial causada por el estrés resultante de la presión del material superpuesto en la pila de cristales [11].

3. Experimentación

Se toman en cuenta distintos puntos los cuales se asocian de diferentes maneras como el número de cristales, el tamaño de los cristales y el tiempo de cristalización. Esto introduce tres constantes características que solo se pueden encontrar empleando modelos cinéticos específicos de cristalización. El tiempo característico de cristalización depende solo débilmente de la cinética exacta de cristalización, pero el número y tamaño de cristal típicos son más sensibles a la cinética; la nucleación y el crecimiento pueden estar relacionados exponencialmente con el tiempo, el tamaño de los cristales es principalmente el resultado de la nucleación heterogénea y la anexión de grano rápida y continua y la migración de los límites [12]. Otra característica a tomar en cuenta es el estudio del potencial de la partícula (carga electrostática) ya que a un mayor potencial electrostático se proporciona una mayor estabilización [13, 14] Los cambios en la tasa de crecimiento y nucleación fuera de las condiciones constantes alterarán la pendiente y la intersección una población de cristales (temperatura) [15].

3.1. Herramientas

La simulación se realizó en una laptop-dfbogt8t con un procesador AMD Ryzen 5. Para la simulación se utilizó el paquete estadístico R [16] para generar visualizaciones y analisis de resultados.

3.2. Simulación

La simulación se llevó a cabo en base a los temas relacionados comportamiento browniano [17] e interacciones entre partículas [18] con la finalidad de agregar un movimiento aleatorio de una partícula en una solución y ver como interactúan los factores como la masa, la energía electrostática y la temperatura al generar partículas de mayor tamaño simulando el fenómeno de Ostwald.

3.3. Análisis de resultados

Se describen varios métodos sencillos para simular la evolución browniana en un tiempo continuo el movimiento Browniano es un modelo en que los cambios de un tiempo a otro son aleatorios de una distribución normal media $0,0$ y una varianza $0,2$ multiplicado por el tiempo.

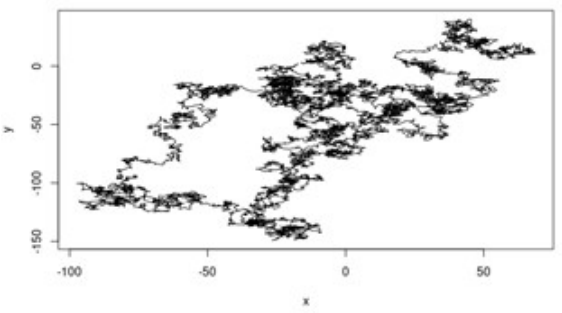


Figura 3: Instancia de movimiento Brwoniano.

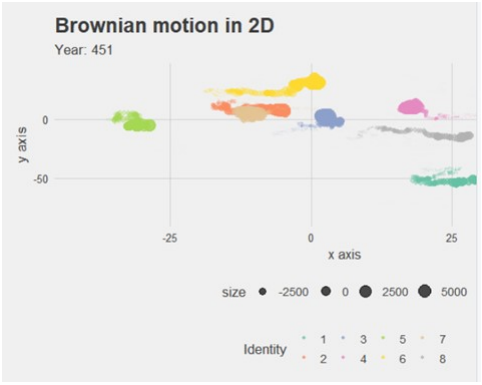


Figura 4: Movimiento Brwoniano.

4. Resultados y discusión

El objetivo final fue crear una simulación de difusión de partículas dependientes del tiempo con un movimiento aleatorio variando el tamaño, donde las partículas más pequeñas se disuelven y las más grande se agrandan hasta que se genera una sola, tomando en cuenta la temperatura el movimiento será lentos si el medio es frío y rápido si hace calor. Para la comprensión de la simulación se realizo una unica instancia de movimiento Browniano a lo largo de 100 generaciones de tiempo con una varianza de 0,01 (figura **??**)

Una vez comprendido se añadieron detalles para crear un tamaño de partícula cambiante, añadiendo un umbral al tamaño inicial, si es mayor que el valor dado, la partícula seguir creciendo, y viceversa.

Posteriormente para una particula, se diseño una nueva función que indica el movimiento Browniano una vez obtenidos los datos utilizando ggplot2 y gganimate generamos nuestra animación [19].

5. Conclusiones

Se realizo una simulación de difusión de particulas dependiendo del tiempo se realizo un grafico de el tiempo aunque se requeriría realizar la simulación nuevamente en el futuro, incluyendo propiedades específicas de un material y una solución con la finalidad de obtener datos informativos sobre la saturación o sobresaturación de una solución o bien para aprovechar las energías libres de los compuestos ya sea para estabilizar un fluido o emulsión o bien mantenerlo siempre activo con la adicción de algún otro compuesto. Se deberá tomar en cuenta que la aglomeración tipo Ostwald también depende de fenómenos como floculación o agregación que de cierta manera aceleraran el proceso debido a que estas partículas siempre están en movimiento lo que provocaría una coalescencia.

5.1. *Trabajos futuros*

Complementar de manera correcta las cargas electrostáticas de compuestos así como el tamaño de cada uno ya que de estos dependerá la maduración, determinar temperaturas no solo pequeñas sino expandir a temperaturas críticas en las que los compuestos puedan fusionarse, determinar tamaños promedio de diferentes compuestos para relacionarlo con materiales específicos.

6. Agradecimientos

Se agradece encarecidamente a la Dra. Elisa Schaeffer por sus correcciones en la estructura original del trabajo y las bases enseñadas para el desarollo de el mismo.

Referencias

[1] D. J. McClements, Reduced-fat foods: The complex science of developing diet-based strategies for tackling overweight and obesity, Advances in nutrition 6 (3) (2015) 338–352. doi:10.3945/an.114.006999.

[2] N. Vinet, M. D. Higgins, Magma solidification processes beneath Kilauea volcano, Hawaii: a quantitative textural and geochemical study of the 1969–1974 Mauna Ulu Lavas, Journal of Petrology 51 (6) (2010) 1297–1332. doi:10.1093/petrology/egq020.

[3] A. Zeh, Crystal size distribution (CSD) and textural evolution of accessory apatite, titanite and allanite during four stages of metamorphism: an example from the Moine supergroup, Scotland, Journal of Petrology 45 (10) (2004) 2101–2132. doi:10.1093/petrology/egh049.

[4] E. Mollard, C. Martel, J.-L. Bourdier, Decompression-induced crystallization in hydrated silica-rich melts: Empirical models of experimental plagioclase nucleation and growth kinetics, Journal of Petrology 53 (8) (2012) 1743–1766. doi:10.1093/petrology/egs031.

[5] C. J. Grose, J. C. Afonso, Chemical disequilibria, lithospheric thickness, and the source of ocean island basalts, Journal of Petrology 60 (4) (2019) 755–790. doi:10.1093/petrology/egz012.

[6] A. R. McBirney, A. Nicolas, The Skaergaard layered series. part II. magmatic flow and dynamic layering, Journal of Petrology 38 (5) (1997) 569–580. doi:10.1093/petroj/38.5.569.

[7] L. A. Hernandez, Estudio de la estabilidad y evolución de emulsiones del tipo O/W mediante calorimetría diferencial de barrido (DSC) y dispersión dinámica de luz, Master's thesis, Escuela Superior de ingeniería Quimica e Industrias extractivas. Instituto Politecnico Nacional (diciembre 2009).

[8] A. S. Kabalnov, Can micelles mediate a mass transfer between oil droplets? 10 (3) (1994) 680–684. doi:10.1021/la00015a015.

[9] M. Zieg, B. Marsh, Multiple reinjections and crystal-mush compaction in the Beacon Sill, McMurdo Dry Valles, Antarctica, Journal of Petrology 53 (12) (2012) 2567–2591. doi:10.1093/petrology/egs059.

[10] J.-C. Liu, Y. Tang, Y.-G. Wang, T. Zhang, J. Li, Theoretical understanding of the stability of single-atom catalysts, National Science Review 5 (5) (2018) 638–641. doi:10.1093/nsr/nwy094.

[11] O. Namur, M. C. Humphreys, M. B. Holness, Crystallization of interstitial liquid and latent heat buffering in solidifying gabbros: Skaergaard intrusion, Greenland, Journal of Petrology 55 (7) (2014) 1389–1427. doi:10.1093/petrology/egu028.

[12] B. D. Marsh, On the interpretation of crystal size distributions in magmatic systems, Journal of Petrology 39 (4) (1998) 553–599. doi:10.1093/petroj/39.4.553.

[13] A. G. Murillo, D. Aguilar, G. H. Norris, D. M. DiMarco, A. Missimer, S. Hu, S. G. Joan

A. Smyth, C. N. Blesso, Y. Luo, M. L. Fernandez, Compared with powdered lutein, a lutein nanoemulsion increases plasma and liver lutein, protects against hepatic steatosis, and affects lipoprotein metabolism in guinea pigs, The Journal of Nutrition 146 (10) (2016) 1961–1969. doi:10.3945/jn.116.235374.

[14] J.-W. Moon, C. J. Rawn, A. J. Rondinone, L. J. Love, Y. Roh, S. M. Everett, R. J. Lauf, T. J. Phelps, Large-scale production of magnetic nanoparticles using bacterial fermentation, Journal of Industrial Microbiology and Biotechnology 37 (10) (2010) 1023–1031. doi:10.1007/s10295-010-0749-y.

[15] A. Mock, D. Jerram, C. Breitkreuz, Using quantitative textural analysis to understand the emplacement of shallow-level rhyolitic laccoliths—a case study from the Halle volcanic complex, Germany, Journal of Petrology 44 (5) (2003) 833–849. doi:10.1093/petrology/44.5.833.

[16] The R Foundation, The R project for statistical computing (2021).
URL https://www.r-project.org/

[17] E. Schaeffer, Práctica 1: Movimiento browniano (mayo 2021).
URL https://elisa.dyndns-web.com/teaching/comp/par/p1.html

[18] E. Schaeffer, Práctica 9: Interacciones entre partículas (mayo 2021).
URL https://elisa.dyndns-web.com/teaching/comp/par/p9.html

[19] G.-F. J.A, AnimaciÓn (JUNIO 2021).