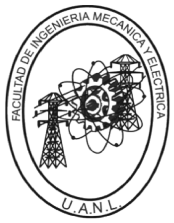
UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Maestría en Ciencias de la Ingeniería con Orientación a la Nanotecnología

Portafolio de Evidencias

Simulación Computacional de Nanomateriales

Segundo semestre

enero-junio 2022

Estudiante:
Ismael Hernández Crespo
1792324

Docentes:
Dr. Virgilio González
Dra. Elisa Schaeffer

Calificación:
Prácticas:6+8+8+8+8+7+7+8+5+6+8+5=84

Proyecto Final:20

Calificación Final:100+

URL GITHUB:
https://github.com/IsmaelHC/Simulacion-NANO-2022

UANL
1 de junio del 2022

P1

Ismael Crespo

16 de febrero de 2022

1. Introducción

El movimiento Browniano fue reportado por primera vez en 1785, por el físico Jan Ingenhausz, al observar el comportamiento de carbón pulverizado en la superficie del alcohol. Este fenómeno fue nombrado como tal por Robert Brown que en 1828 reporto el movimiento de partículas finas, incluyendo el polen, polvo y hollín en la superficie del agua. Einstein lo explica mas tarde en 1905 como movimientos térmicos aleatorios de moléculas de fluido que chocan contra las partículas microscópicas, lo que hace que experimenten un paseo aleatorio.La idea de un movimiento Browniano es la de una caminata de pasos aleatorios en dirección y magnitud Figura 1 con una distribución Gausiana para la posición después de un tiempo t, comprender el comportamiento de las partículas bajo estos movimientos es importante para conocer la difusión entre las fases solidas y líquidas, así como para la dinámica de las micelas[2].

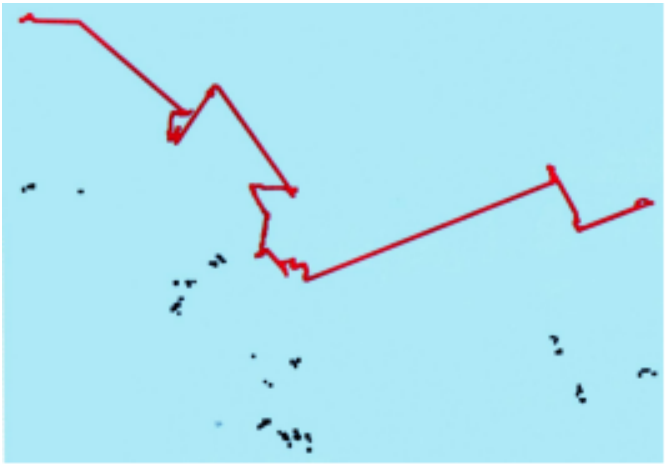


Figura 1: Camiata Brwoniana en 2 dimensiones obtenida de [2]

2. Objetivo

Por medio de la simulación repetida un numero de veces n_r de distintas caminatas aleatorias se pretende analizar y comprender la relación entre el número de pasos n_p y el número de dimensiones con la distancia d final con respecto al origen .

Cuadro 1: Tiempo y espacio requerido para diferentes experimentos

n_p	**Dimensiones**	n_r	**Tiempo [Seg]**
100	1	10	0.34
100,1000	2	20	0.83
100,1000,1000	5	30	14.56

3. Equipo y Método

El equipo utilizado para la medición de rendimientos es una computadora portátil *Acer E5-573* con un Procesador *Intel Core i5 2.20 GHz*,una memoria RAM instalada de 8.0 GB y un sistema operativo de 64 bits y la herramienta *R 4.1.2* se realizo la simulación de un movimiento Browniano. El código computacional genera caminatas aleatorias de $100, 1000$ y 10000 pasos en $1, 2, 3, 4, 5$ dimensiones, cada uno las repite 30 veces y las gráfica por separado en un diagrama caja-bigote.

3.1. Programación en R

Basándose en la publicación previa sobre la simulación del movimiento Brwoniano [1] el Código 1 utiliza la función `runif` para decidir, primero en que dimensión de las que se esta simulando se realizará el recorrido y posteriormente si será un retrocesos o avance en esta dimensión, el código guarda la distancia euclideana $d = \sqrt{(\sum(p1^2, p2^2, p3^2.....))}$ máxima recorrida para cada uno de los experimentos y posteriormente los grafica. El tiempo de computo para el total del experimento es de 14,63. En el cuadro 1 se presentan algunos tiempos de computo variando el número de pasos, las dimensiones y las repeticiones.

4. Resultados

Los resultado de las caminatas se presentan en la Figura 2, se observa un comportamiento constante para la media de cada experimento, esto es debido a que los pasos son de la misma magnitud para cada experimento. Para el experimento donde $n_p = 10000$ en 1 dimensión se observa una gran diferencia entre la mayor distancia y la menor distancia .

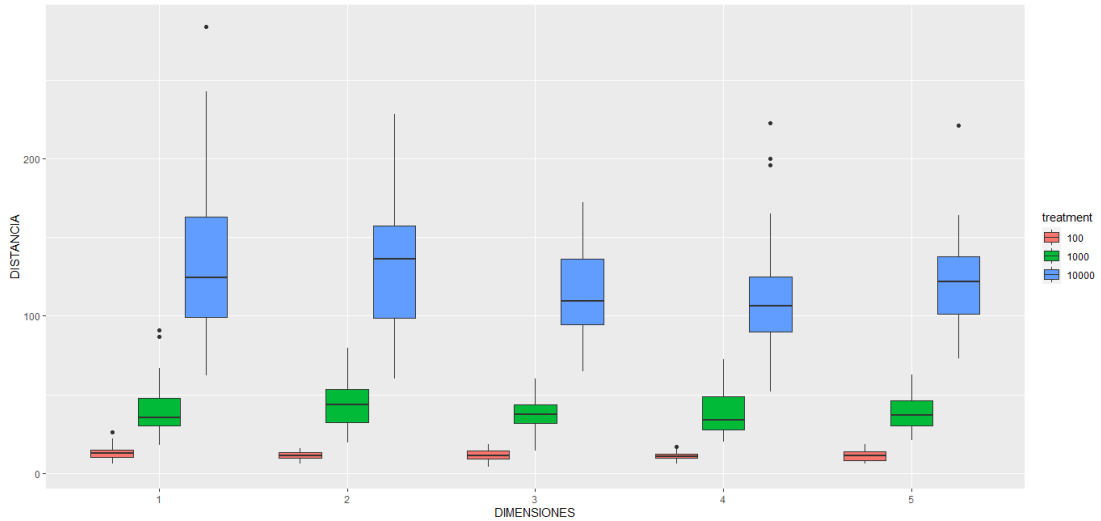


Figura 2: Resultados de cada una de las repeticiones de 15 caminatas, dividas en cada una de las 5 dimensiones y para cada n_p=100,1000,10000

```
start_time <- Sys.time()
p1=c(0,0)
cor=30
dur <- c(100,1000,10000)
num_ejes<-5
eje<-0
#DISTANCIA=rep(0,cor*num_ejes*length(dur))
DISTANCIA=numeric()
mayor<-array(rep(0,cor*num_ejes*length(dur)),dim=c(cor,num_ejes,length(dur)))
pos<- array(rep(0, cor*num_ejes*num_ejes), dim=c(cor,num_ejes, num_ejes))
 for(p in dur){
  for (pasos in 1:p) {
  for (caminata in 1:cor){
  for (e in 1:num_ejes){
    eje<-round(runif(1,1,e))
      if (runif(1) < 0.5){
      pos[caminata,eje,e] <- pos[caminata,eje,e] + 1
      }
      else{
       pos[caminata,eje,e]<- pos[caminata,eje,e] - 1
      }
   euclideana = sqrt(sum((pos[caminata,,e]**2))) #
 if (p==100){
  j=1
}
  if (p==1000){
  j=2
}
  if (p==10000){
  j=3
}
 mayor[caminata,e,j] = max(mayor[caminata,e,j],euclideana)
}}}}
```

Listing 1: Codigo en R

5. Conclusiones

Los movimientos Brownianos son definidos como caminatas aleatorias, es posible simularlas, bajo varias limitantes, para analizar y encontrar relaciones estadísticas. A pesar del cambio en el número de dimensiones, la distancia recorrida esta mas bien gobernada por el número de pasos realizados.

Referencias

[1] E.Schaaeffer. Practica 1: Movimiento brwoniano. 2022. URL `https://satuelisa.github.io/simulation/p1.html`.

[2] G.Zumofen K.Joseph, M.Shlesinger. Beyond brownian motion. *Physics Today*, 49(2):33–40, 1996. doi: 10.1063/1.881487.

P2

Ismael Crespo

23 de febrero de 2022

1. Introducción.

Este trabajo presenta tres experimentos distintos donde se simula un autómata celular en dos dimensiones, la generación de vida a partir de semillas aleatorias, y finalmente un autómata celular en tres dimensiones. La programación se realizo en *Python 3.10.2* y se utilizó la selección pseudoaleatoria de números como base para la simulación de cada caso.

2. Objetivo.

1.-Estudiar la probabilidad de vida en un autómata celular en 2 dimensiones después de un número determinado de pasos n_p, variando el número de elementos en la matriz num y el valor mínimo para dejar vivas las celdas al inicio p.

2.-Modelar algún tipo de crecimiento o cristalización en la microestructura de un material. Núcleos aparecen al azar en celdas desocupadas y expanden con una tasa constante a celdas vecinas hasta agotar el espacio disponible. Examina la distribución de los tamaños de los núcleos que no toquen el borde al finalizar la simulación.

3.-Ejecutar un autómata celular en un modelo tridimensional.

3. Programación en Python.

Se realizaron tres códigos diferentes, la base en todos los casos es el uso de herramientas (`random`, `rand.int`, `random.choose`) que generan y seleccionan de manera pseudoaleatoria números para posicionar vida o semillas así como elegir las características de las semillas, como antecedentes se utilizó como base la lógica de programación presentada en trabajos previos por E.Schaeffer [1].Véase referencia [2] para consulta completa de los códigos.

3.1. Autómata Celular en 2 dimensiones variando n_p y num.

Para comenzar con la simulación de un autómata celular se generaron valores con la herramienta `random` estos valores van desde 0,0 a 1,0 posteriormente el código elimina del vector de valores aquellos que sean menores a p, para finalizar, durante un determinado número de veces se analizan las vecindades de cada elemento, y aquellas donde a sus alrededores existan más de 3 elementos con vida, serán eliminaran. Para nuestros análisis el valor de p tuvo valores de $0{,}2, 0{,}4, 0{,}6$ y 0,8 y se hizo cada caso para matrices con num=100, 225 y 400 (Código 1).

Una vez terminado el experimento se encuentra la probabilidad de terminar con vida cada replica, por lo que se divide el número de replicas que tuvieron vida al final R_v entre el total de replicas R_t.

$$Probabilidad = \frac{R_v}{R_t} * 100 \tag{1}$$

Código 1: Programación de un autómata celular variando el valor de p y num

```
for d in (dim):
      xp=0
      for p in (p_value):
          num = d**2
          total=0
          for rep in range(replicas):
              valores = [1 * (random() < p) for i in range(num)]
              actual = np.reshape(valores, (d, d))
              def mapeo(pos):
                  fila = pos // d
                  columna = pos % d
                  return actual[fila, columna]
              assert all([mapeo(x) == valores[x]  for x in range(num)])
              def paso(pos):
                  fila = pos // d
                  columna = pos % d
                  vecindad = actual[max(0, fila - 1):min(d, fila +2),
                     max(0, columna - 1):min(d, columna + 2)]
                  return 1 * (np.sum(vecindad) - actual[fila, columna] == 3)
              for iteracion in range(dur):
                  valores = [paso(x) for x in range(num)]
                  actual = np.reshape(valores, (d, d))
```

3.2. Simulación de un crecimiento o cristalización en una microestructura

La simulación de un crecimiento se generó modificando el código 1, de manera similar se utilizó la herramienta `random.choice` para colocar semillas de distintos valores, las semillas fueron colocadas un número determinado de veces y posteriormente se analizó cada elemento de la matriz que no tuviera valor para que seleccionara de manera pseudoaleatoria un valor que estuviera dentro de su vecindad, y así generar crecimientos hasta que todos los elementos tuvieran valores.

3.3. Autómata tridimensional.

El autómata tridimensional fue creado utilizando `matplotlib` para generar la gráfica de una matriz tridimensional y utilizando la lógica descrita en la sección 3.1 se eliminaron los elementos que estuvieran rodeados por más de 5 elementos en las tres dimensiones, dicho elementos fueron seleccionados de manera aleatoria para ser analizados por medio de la herramienta `random.choice` que selecciona un número pseudoaleatorio de matriz, columna y celda.

4. Resultados.

La figura 1 presenta la probabilidad, obtenida con la ecuación 1, de encontrar vida variando los valores de p y de num en un autómata celular, de esta gráfica podemos deducir que al existir mayores espacios que puedan ser ocupado por vida, las vecindades no están tan saturadas por lo que no se eliminarán tantas celdas con vida. El valor de p nos indica que tan saturado va a estar cada matriz, a valores altos de p naturalmente la población inicial va a ser alta y por lo tanto existe menos posibilidad de vida (figura 2). Encontramos un valor óptimo cuando $p = 0{,}4$ en una matriz de $num = 400$.

La generación de vida a partir de la colocación de semillas aleatorias se presenta en la figura 3. El propósito de esta simulación es evaluar el crecimiento de las semillas que no tocaron las fronteras de la estructura, la variación entre

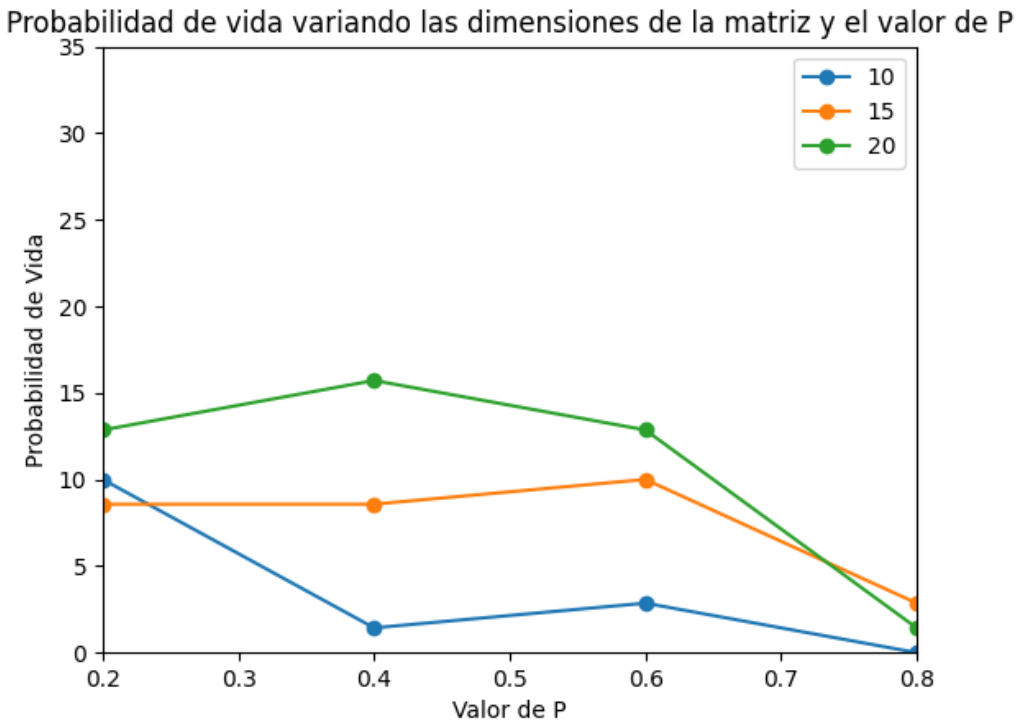


Figura 1: Probabilidad de que la matriz contenga vida después de 50 iteraciones, cada experimento se repitió 30 veces.

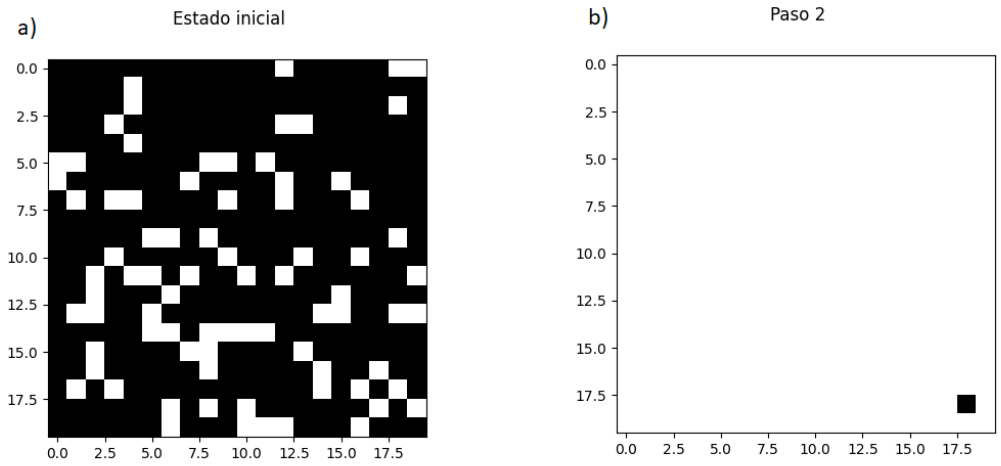


Figura 2: a)Estado inicial de una matriz cuando $p = 0{,}8$ y $num = 400$, se observa una saturación de las celdas que provoca vecindades igualmente saturadas por lo que la vida acaba a las pocas iteraciones. b)Paso 2 de la simulación, la vida es casi nula.

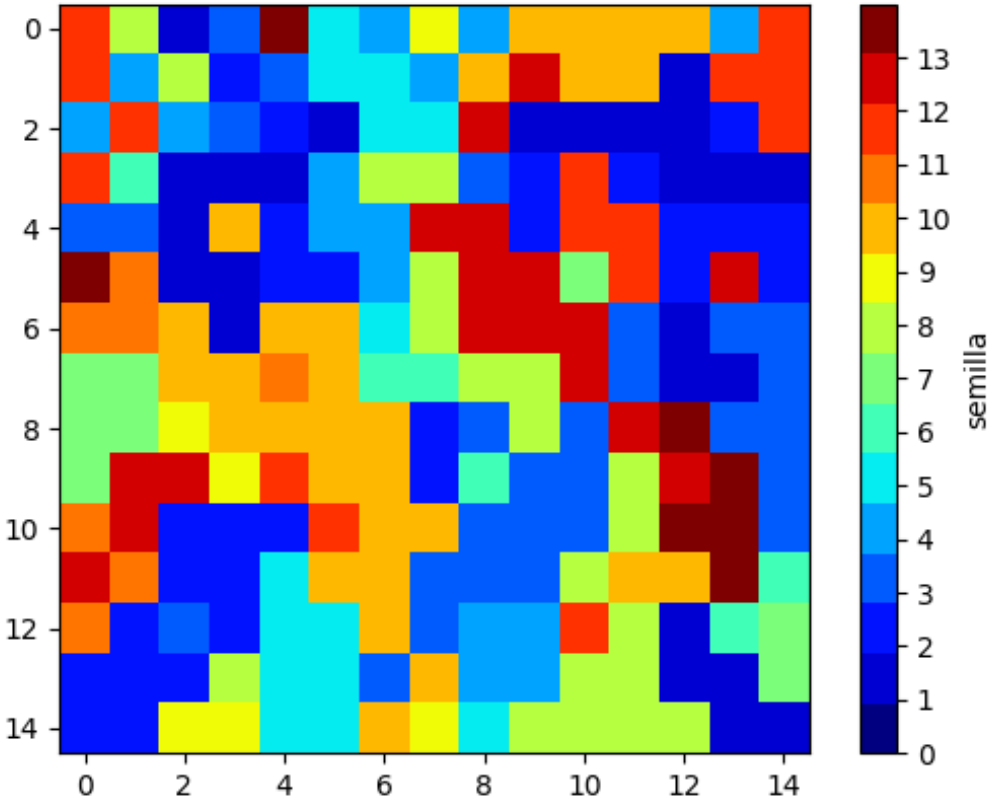


Figura 3: Estado final de una matriz donde se sembraron semillas con valores de 1 a 14 de manera aleatoria al inicio de la simulación, el mayor crecimiento observado se presenta para la semilla 10 con una cobertura de 17 elementos.

los tamaños de estos crecimientos van desde un elemento hasta 17 elementos para la semilla de tipo 10. Los mayores crecimientos tienen cuando menos un extremo ubicado en la parte central de la matriz (semilla 13,10 y 3).

En la figura 4 se presenta el paso inicial y final de un autómata celular en tres dimensiones que sigue la lógica presentada en la sección 3.3. Para esta simulación p=0,5 para una matriz tridimensional de $10 * 10 * 10$ elementos. El número inicial de elementos vivos es de 515, después de 3000 iteraciones pseudoaleatorias quedaron vivos 205 elementos.

5. Conclusiones.

Las simulaciones pseudoaleatorias son herramientas poderosas que, como un primer acercamiento, nos dan la oportunidad de observar como podrían ocurrir algunos crecimientos en las estructuras. En este trabajo el valor de p toma gran importancia, en otras palabras, nos indica la saturación que tendremos en el sistema y por tanto las interacciones que pueden ocurrir dentro del sistema.

Referencias

[1] E.Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

[2] I.Crespo. P2:autómata celular. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P2`.

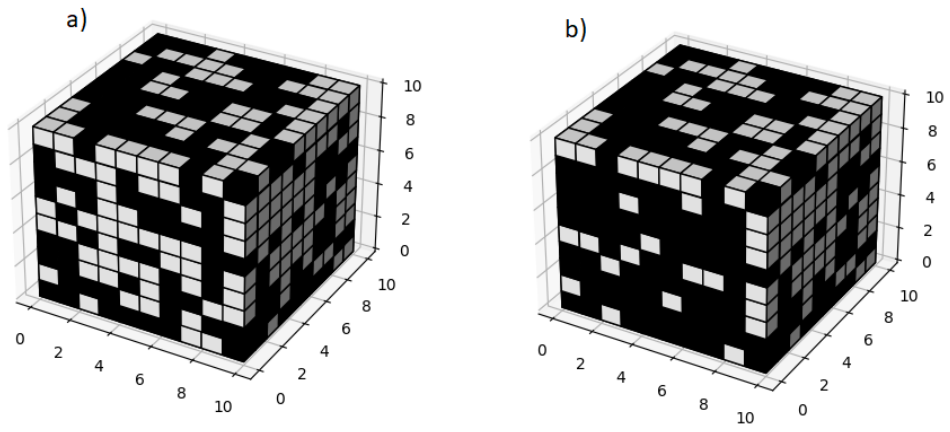


Figura 4: Simulación de un autómata en 3 dimensiones $p = 0{,}5$ elimina elementos que en sus vecindades tenga más de 5 elementos vivos. a)Inicio de la simulación,elementos vivos=515, b) Final de la simulación,elementos vivos=205.

P3

Ismael Crespo

2 de marzo de 2022

1. Introducción.

Conocer el orden como se ejecutan los códigos computacionales es de suma importancia para realizar rutinas eficientes, este trabajo presenta diferentes rutas para llegar a un mismo objetivo: encontrar los números primos p en una secuencia numérica. Se comparan los tiempos de ejecución para concluir cuál es la mejor opción, la comparación entre los distintos métodos se realizó en *R 4.1.2*.

2. Objetivos.

1.-Examinar las diferencias en los tiempos de ejecución variando, el orden de los números, la cantidad de núcleos asignados al cluster y la variante de la rutina para determinar si el número es primo.

2.-Encontrar todos los divisores de un número (todos los enteros mayores a uno y menores al número mismo que lo dividen exactamente) y examinar si las conclusiones cambian.

3.-Encontrar los factores y sus multiplicidades, es decir, que encuentre para n aquellos números primos $1 < p \leq n$ y sus potencias para que el producto de los factores con esas potencias de n y examinar nuevamente si este cambio afectó las conclusiones.

3. Programación en R.

Fueron tomadas como base tres rutinas desarrolladas por E.Schaeffer [2] para encontrar números primos, pasando de rutinas robustas a rutinas mas simplificadas que retiran obviedades de los ciclos, p. ej., si un número no es divisible por 2, tampoco lo será para los demás números pares (Código 1). El equipo utilizado para la medición de rendimientos es una computadora portátil *Acer E5-573* con un Procesador *Intel Core i5 2.20 GHz*,una memoria RAM instalada de 8.0 GB y un sistema operativo de 64 bits con 4 núcleos.

Código 1: Rutina No3 eliminando los números no divisibles entre 2 y utilizando la funcipon `ceiling`.

```
1   primo_3 <- function(n) {
2     if (n == 1 || n == 2) {
3         return(TRUE)
4     }
5     if (n %% 2 == 0) {
6         return(FALSE)
7     }
8     for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
9         if ((n %% i) == 0) {
10            return(FALSE)
11        }}
12    return(TRUE)}
```

Cada una de las tres rutinas fue evaluada con los mismos números, sin embargo, para su análisis estas se ordenaron de manera ascendente (`ot`), descendente (`it`) y aleatorio (`at`), dichas combinaciones de rutinas y ordenamiento numérico fueron evaluadas tanto para 3 núcleos y 1 núcleo por lo que para la primera parte del trabajo se realizaron 18 pruebas diferentes (Código 2). Para consulta más amplia de los códigos véase la Referencia [4].

Código 2: La rutina 3 evaluada para cada ordenamiento de los números y la selección de los núcleos a utilizar.

```
registerDoParallel(makeCluster(detectCores() - 1)) #Nucleos a utilizar
ot_3<- c(ot_3, system.time(foreach(n = original, .combine=c)
%dopar% primo_3(n))[3]) # de menor a mayor
it_3 <- c(it_3, system.time(foreach(n = invertido, .combine=c)
%dopar% primo_3(n))[3]) # de mayor a menor
at_3<- c(at_3, system.time(foreach(n = aleatorio, .combine=c)
%dopar% primo_3(n))[3]) # orden aleatorio
```

La segunda parte del trabajo encuentra el total de números que dividen exactamente a n, posteriormente se encuentran nuevamente todos los valores (p) y la potencia x a la que p tiene que ser elevado para ser igual a n, La Ecuación 1 describe el enunciado y la Ecuación 2 muestra la forma algebraica para encontrar el valor de x (véase el Código 3).La segunda parte del trabajo en su totalidad fue evaluada utilizando tres núcleos.

$$p^x = n \tag{1}$$

$$x = log_p n \tag{2}$$

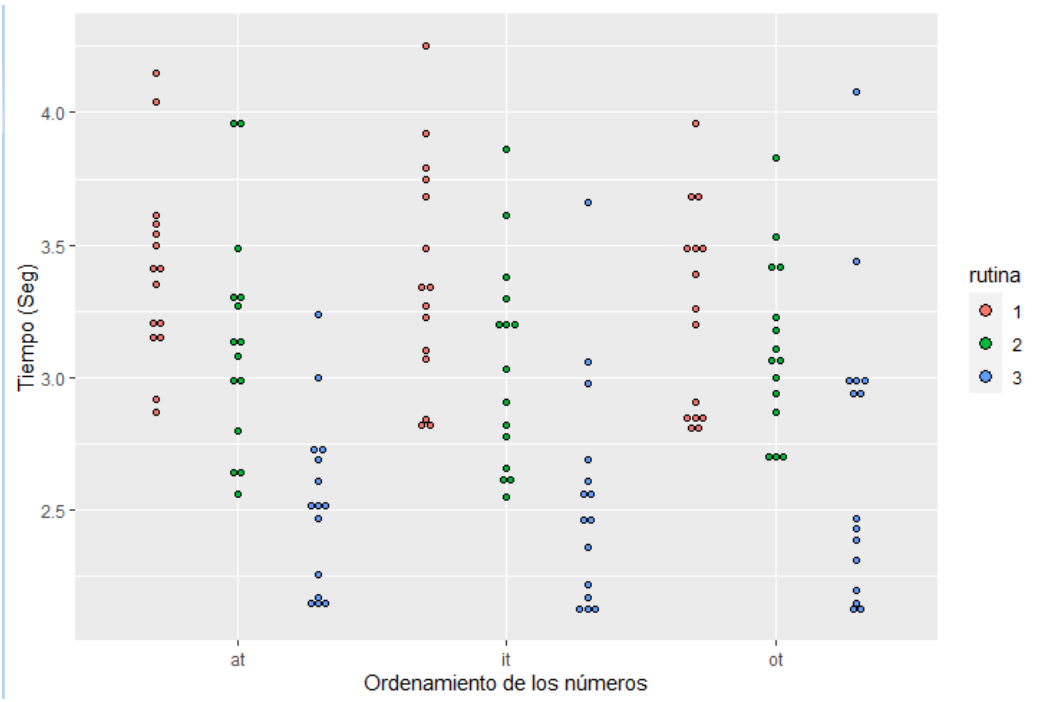
Código 3: Función para encontrar el valor de la potencia x.

```
potencia<- function(n) {
    if (n == 1 || n == 2) {
        return(TRUE)
    }
    if (n %% 2 == 0) {
        return(FALSE)
    }
    for (i in seq(3, max(3, ceiling(sqrt(n))), 2)) {
        if ((n %% i) == 0) {
            return(FALSE)
        } else { #Encuentra numero primo
           return(x=log(i,base=hasta)) #Encuentra la potencia
           }}}
```

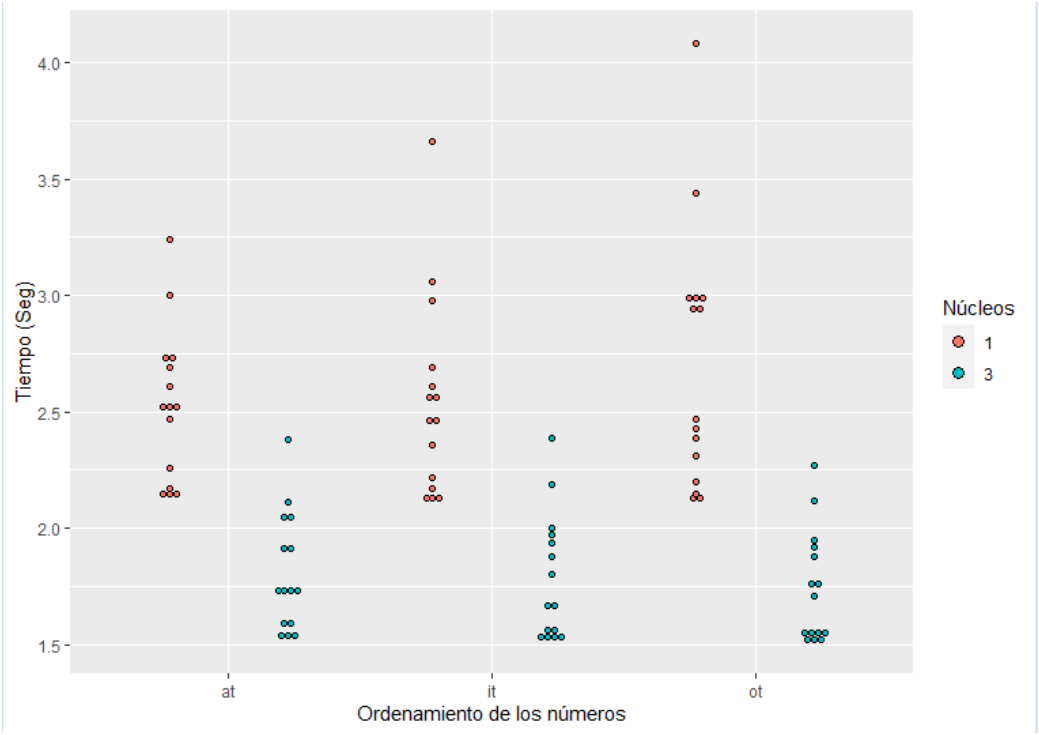
4. Resultados.

La figura 1 muestra los tiempos en segundos que tomo encontrar los números primos desde 1 a 4000, la Figura 1a muestra los tiempos requeridos utilizando solamente un núcleo, de aquí, por conveniencia se decidió comparar la rutina No.3 (Código 1) con cada ordenamiento utilizando 1 y 3 núcleos Figura 1b. De la Figura 1a se deduce que la rutina No.3 es la mas eficiente, posteriormente en la 1b se observa un mejor rendimiento cuando se dedican 3 núcleos a esta operación.

En el Cuadro 1 se observan los valores para la media, máximos y mínimos de cada caso así como su desviación estándar, existe una clara reducción en los tiempos de ejecución utilizando tres núcleos, la tabulación se realizó con la función `stargazer` [3]. En el Cuadro 2 se observan valores p muy por debajo del 0.05 % cuando variamos el número de núcleos, es decir que el cambio en esta variable tiene un impacto importante en los rendimientos, así mismo observamos que variar el ordenamiento no tiene significancia, sin embargo, el uso de diferentes rutinas también tiene valores p menores a 0.05 %.Los valores de p fueron obtenidos con la función `wilcox.test`, véase Referencia [1].



(a) Tiempos requeridos para correr cada rutina(1,2,3),para cada orden numérico(ot,it,at), utilizando solamente un núcleo.



(b) Tiempos requeridos para correr la rutina 3, con cada orden numérico, para 1 y 3 núcleos.

Figura 1: Comparación de los tiempos requeridos para cada configuración de las rutina.

Cuadro 1: Datos estadísticos de las rutinas, (ot_3,it_3 y at_3, representan los valores para la rutina No.3 utilizando 3 núcleos)

Statistic	Promedio	Median	Max	Median	St. Dev.
ot_1_1	2.33	2.98	3.28	2.94	0.23
it_1_1	2.84	3.02	3.95	2.87	0.36
at_1_1	2.81	2.97	3.69	2.89	0.23
ot_2_1	2.30	2.70	2.95	2.71	0.14
it_2_1	2.60	2.71	2.83	2.70	0.08
at_2_1	2.53	2.72	3.11	2.66	0.18
ot_3_1	2.09	2.27	2.70	2.17	0.20
it_3_1	2.07	2.25	2.92	2.20	0.21
at_3_1	2.08	2.28	2.72	2.20	0.20
ot_3	1.51	1.61	1.96	1.55	0.14
it_3	1.50	1.69	2.34	1.59	0.25
at_3	1.53	1.69	1.88	1.70	0.13

Cuadro 2: Valores p entre diferentes experimentos, variando el número de núcleos, la rutina y el ordenamiento.(ot_3_1 representa los tiempos para la rutina No.3 utilizando un núcleo).

Variable 1	Variable 2	$p - value$
ot_3	ot_3_1	$3{,}58 \times 10^{-9}$
ot_3	it_3	$3{,}19 \times 10^{-1}$
ot_1	ot_3	$5{,}19 \times 10^{-4}$
ot_2	it_2	1,00
ot_1	it_1	0,60
at_1	it_1	0,90
at_3	it_3	0,38

Los resultados de la segunda etapa del trabajo, se presentan en la Figura 2, nuevamente se utilizó la rutina No.3 para comprar los rendimientos, con las rutinas para encontrar todos los números divisores exactos dentro de n y para encontrar el valor de x a partir de la Ecuación 2, ejecutando cada rutina con tres núcleos. No se observa un cambio importante, Figura 1, en los tiempos requeridos para correr cada rutina, el Cuadro 3 presenta los datos estadísticos para estas rutinas, se observa un tiempo promedio mayor para las tres rutinas que encuentran la potencia, sin embargo los valores de la desviación estándar no nos permiten llegar a una conclusión concreta. La rutina que encuentra el divisor exacto (ot_residuo,it_residuo,at_residuo) tiene los promedios de tiempo menores, al igual que sus desviaciones estándar. Los resultados del valor p (Cuadro 4), muestran comportamientos similares a los observadores en el Cuadro 2, en cuanto a la dependencia entre las variables de cada rutina, nuevamente los ordenamientos utilizados no tienen significancia en los tiempos de ejecución, sin embargo la estructura de las rutinas si repercute en los tiempos finales.

Cuadro 3: Datos estadísticos de la rutina No.3, la rutina que encuentra los divisores exactos (RESIDUO) y la rutina para encontrar la potencia x (POTENCIA).

Statistic	Min	Promedio	Max	Median	St. Dev.
ot_3	1.51	1.61	1.96	1.55	0.14
it_3	1.50	1.69	2.34	1.59	0.25
at_3	1.53	1.69	1.88	1.70	0.13
ot_residuo	1.55	1.61	1.83	1.60	0.07
it_residuo	1.56	1.61	1.84	1.59	0.07
at_residuo	1.55	1.60	1.74	1.58	0.05
ot_potencia	1.52	1.91	3.37	1.84	0.46
it_potencia	1.53	1.84	2.75	1.85	0.31
at_potencia	1.53	2.01	4.22	1.86	0.66

Cuadro 4: Valores p entre los tiempos variando las rutinas (No.3, residual y potencia) y el ordenamiento numérico.

Variable 1	Variable 2	$p - value$
at_potencia	it_potencia_1	0,45
at_potencia	ot_potencia	0,46
it_potencia	ot_potencia	0,85
ot_residuo	ot_potencia	0,02
it_residuo	at_residuo	0,39
at_residuo	ot_residuo	0,90
ot_residuo	it_potencia	0,11
ot_residuo	at_potencia	$6{,}0 \times 10^{-3}$

5. Conclusiones.

Es importante evitar obviedades dentro de la programación para obtener rutinas eficientes y constantes. Debemos conocer a fondo los requerimientos de cada rutina y las capacidades del equipo para destinar el número de núcleos óptimos con los que se computan los programas. Los ordenamientos presentados no tienen un impacto en los tiempos de ejecución tan importante como la estructura de la rutina y el número de núcleos utilizados.

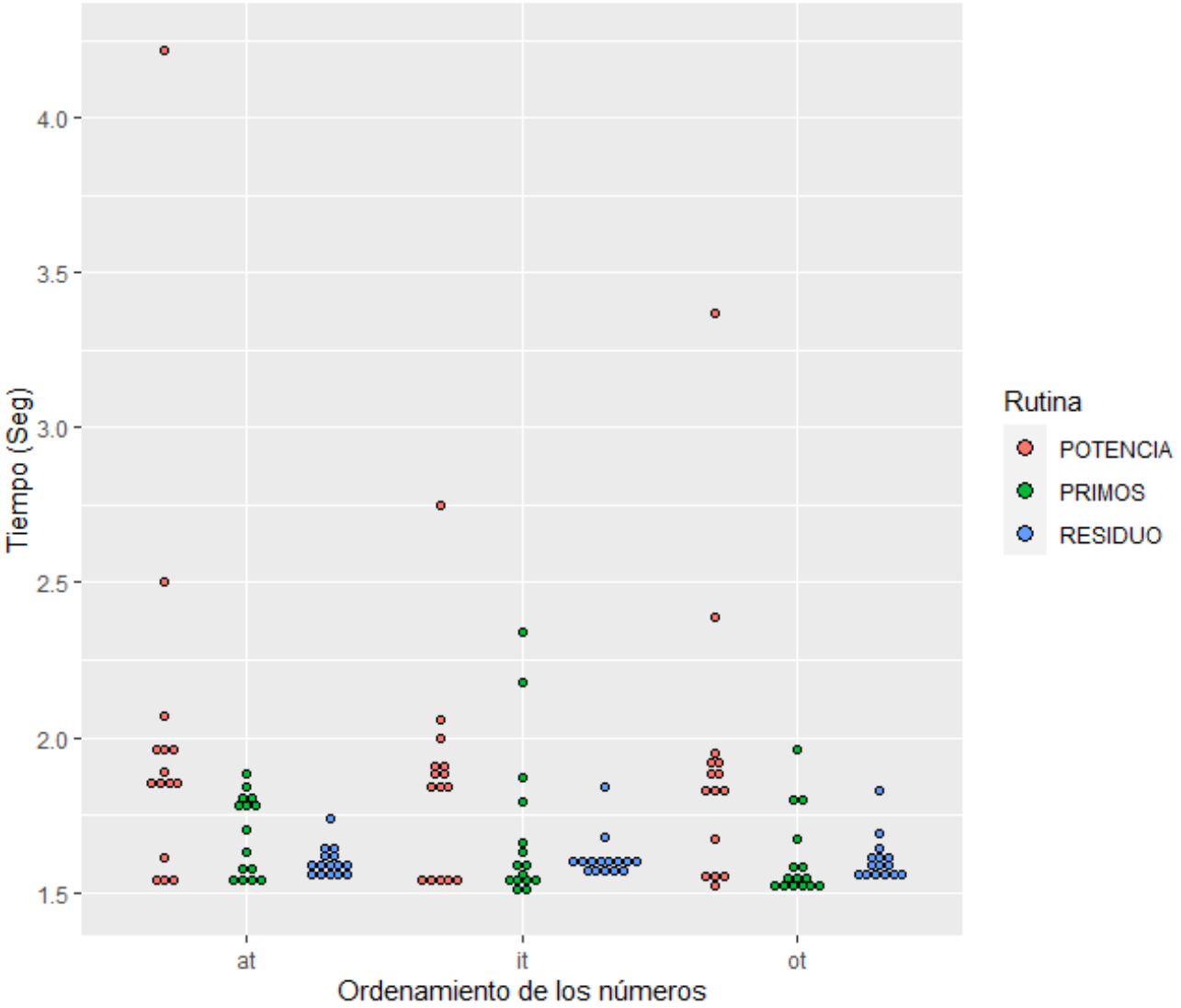


Figura 2: Tiempos requeridos,utilizando tres núcleos, para correr la rutina No.3 (PRIMOS), la rutina para encontrar los números divisorios exactos(RESIDUO) y para encontrar la potencia x (POTENCIA).

Referencias

[1] Data Analytics.org.uk. Basic statistical tests using r. 2022. URL https://www.dataanalytics.org.uk/basic-statistical-tests-using-r/.

[2] E.Schaeffer. Simulación. 2022. URL https://satuelisa.github.io/simulation.

[3] H.Marek. stargazer.well-formatted regression and summary statistics tables. r package version 5.2.2. 2018. URL https://CRAN.R-project.org/package=stargazer.

[4] I.Crespo. P2:autómata celular. 2022. URL https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P3.

P4

Ismael Crespo

9 de marzo de 2022

1. Introducción.

Este trabajo presenta la creación de diagramas de Voronoi en dos dimensiones, la simulación de estas estructuras pueden asimilarse como los mecanismos de cristalización en la ciencia de los materiales. La creación de estos diagramas se realizó creando rutinas en *R 4.1.2*.

2. Objetivos.

1.-Examina el efecto del tasa n contra k (por lo menos tres niveles de la densidad de semillas), en la probabilidad de una segunda grieta llegue a tocar una primera grieta (es decir, fracturando la pieza dos veces con posiciones iniciales generadas independientemente al azar, sobre varias réplicas), visualizando los resultados y aplicando métodos estadísticos.

2.-Crecer celdas dinámicamente alrededor de semillas de tal forma que las semillas aparecen al azar en distintas iteraciones y crecen con una tasa exponencialmente distribuida (variable entre núcleos pero constante para un núcleo específico) hasta toparse con las demás celdas, así como se muestra en la animación. Examina los cambios producidos en el fenómeno de propagación de grietas que esta nueva forma de crear las celdas provoca, ya que las semillas resultan en celdas de tamaños distintos según su edad y su tasa, además del efecto de la posición relativa a las demás semillas.

3. Programación en R.

La base de programación para la generación de los diagramas de Voronoi fue obtenido de trabajos previos por E.Schaeffer [1], la creación de estos diagramas comienza con la colocación de k cantidad de semillas aleatoriamente en una matriz de $n \times n$, para crear las zonas hacia donde hubo crecimiento de cada semilla se encontró la semilla mas cercana a cada punto de la matriz (Código 1).

Código 1: Busqueda de la semilla mas cercana para cada punto de la matriz`ceiling`.

```
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
            }
        }
```

Una vez generado el diagrama de Voronoi, fueron generadas fracturas numéricas, una primera fractura se generó ubicando un punto en la frontera de la matriz para posteriormente crecer hacia algún punto vecino, por medio la variable `prob` varia la facilidad con la que la fractura se va propagar entre fronteras o dentro de un núcleo, será cada vez mas difícil propagarse dentro de un núcleo (véase el código 2). Una vez generada una primera fractura numérica se generó una segunda fractura con la misma lógica, pero con un valor mayor a la anterior, y cuando esta fractura pase sobre la primera quedara registrada en un vector, el análisis de la probabilidad de que las fracturas se crucen se realizó variando la densidad de semillas en un matriz . Para consulta más amplia de los códigos véase la referencia [3].

Código 2: Recorrido de una fractura dentro de un núcleo o entre fronteras.

```
 if (length(frontera) > 0) { # siempre tomamos frontera cuando haya
            if (length(frontera) > 1) {
                elegido <- sample(frontera, 1)
            } else {
                elegido <- frontera # sample sirve con un solo elemento}
            prob <- 1 # estamos nuevamente en la frontera
else{if (length(interior) > 0) { # no hubo frontera para propagar
            if (runif(1) < prob) { # intentamos en el interior
                if (length(interior) > 1) {
                    elegido <- sample(interior, 1)
                } else {
                    elegido <- interior}
                prob <- dificil * prob # mas dificil a la siguiente}
```

La segunda parte del trabajo varía la creación de los diagramas de Voronoi de tal forma que cada semilla tiene tasa de crecimiento diferente, por lo que habrá una diferencia entre las dimensiones de los núcleos y la ubicación de las fronteras. Estos crecimientos se crearon delimitando el crecimiento de cada semilla k en cierto tiempo t, es decir una semilla $k = 15$ no va a comenzar a expandirse hacia sus vecindades hasta que se este en un tiempo $t = 15$ (véase el código 3),posteriormente se realiza el mismo análisis de cruzamiento de fracturas en el diagrama. Se utilizo como base de crecimiento el trabajo expuesto en [2].

Código 3: Función para crecer semillas en cada tiempo .

```
for (t in 1:100){
     ft=numeric()
      ct=numeric()
            for(pos in 1:n^2){
                f <- floor((pos - 1) / n) + 1
                c <- ((pos - 1) %% n) + 1
                if(zona[f,c]>0){
                   if(zona[f,c]<=t){
                      #print(paste(t,semilla,pos,f,c))
                      ft=c(ft,f)
                      ct=c(ct,c)
                       #print(ft)
             }}}
        for(i in 1:length(ft)){
          zona[max(ft[i] - 1,1) : min(ft[i] + 1, n),
                          max(ct[i] - 1, 1): min(ct[i] + 1, n)]+
                          [zona[max(ft[i] - 1, 1) : min(ft[i] + 1, n),
                          max(ct[i] - 1, 1): min(ct[i] + 1, n)]==0]+
                          <-zona[ft[i],ct[i]]]
```

4. Resultados.

La figura 1 muestra el crecimiento de dos fracturas en una matriz de 40×40 y una densidad de semillas $k = 0{,}5 \times 40$. El diagrama de Voronoi inicial se muestra en la figura 1a, la figura 1b muestra el caso en el que no se cruzan las fracturas y la figura 1c muestra un cruzamiento de las fracturas, es deducible que las fracturas van a tener mas posibilidades de cruzar cuando su propagación es mayor, por lo que se espera que al existir mas opciones de fronteras se den mas cruzamientos, es posible pensar que a mayor núcleos habrá mas fronteras y por lo tanto mayor propagación, sin embargo existirán mayores posibilidades de ingresar a un núcleo y la propagación se detendrá como en los casos donde $n = 160$ y $k = 0{,}75$.

La figura 2 presenta la probabilidad de que se crucen las fracturas en diferentes condiciones de densidad de semillas y dimensiones de la matriz, entendiendo que un cruzamiento de fracturas se genera debido a que tan fácil se propagó, los cruzamientos se presentaran mas veces cuando no hay un número alto de núcleos n=20 donde la fractura pueda ingresar, pero hay suficientes fronteras para propagarse $k = 0{,}75 \times 20$.

La segunda parte del trabajo genera las fracturas con la misma lógica sin embargo la generación de los diagramas de Voronoi difiere tal como se menciona en la sección 2. La figura 3 presenta la secuencia como se generó un diagrama de $n = 40$ y $k = 40 \times 0{,}25$ a partir de semillas con tasas de crecimientos diferentes, se puede observar que hay núcleos de mayor tamaño y que existen semillas que no lograron crecer, por lo que el número de fronteras es menor.En la figura 4 se presentan la probabilidad de que las fracturas crucen, en la 5 se presenta un caso para un mismo diagrama de Voronoi, donde existe cruzamiento 5a y donde no lo hay 5b.

5. Conclusiones.

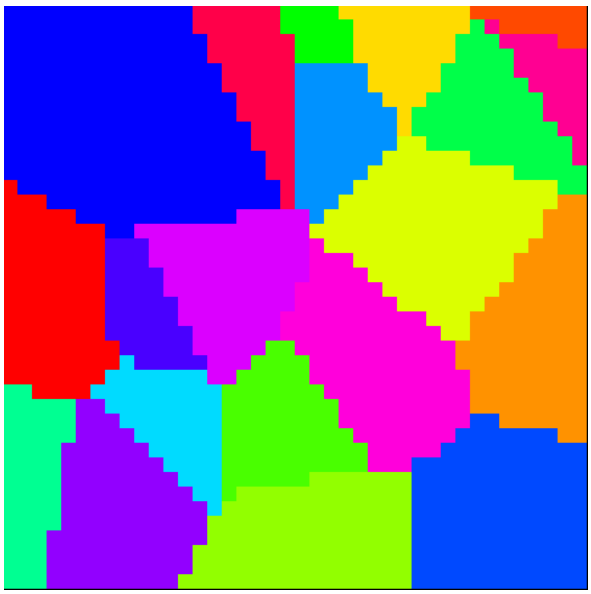
La propagación de fracturas es mas probable a lo largo de las fronteras entre núcleos, por lo que las densidades de semillas en una matriz dictan la facilidad de propagación y la probabilidad de que existan cruzamientos de fracturas. La creación de diagramas con diferentes tasas de crecimiento no afecta las probabilidades de crecimiento, eso se puede deber a que el cruzamiento depende en gran parte de la posición inicial de propagación.

Referencias

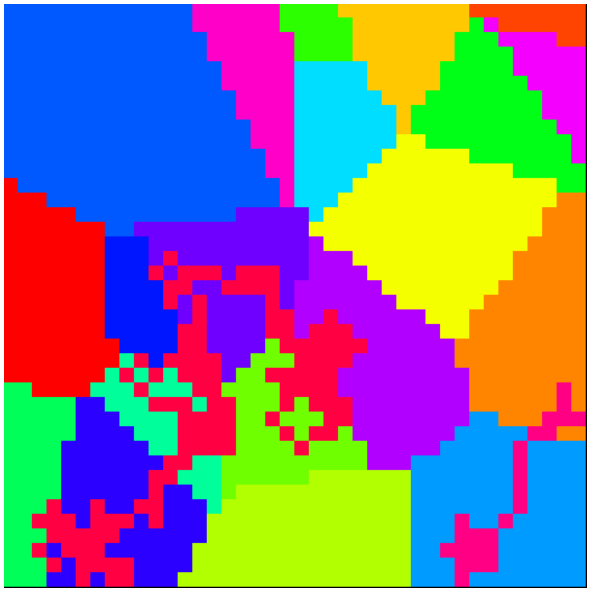
[1] E.Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

[2] I.Crespo. P2:autómata celular. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P2`.

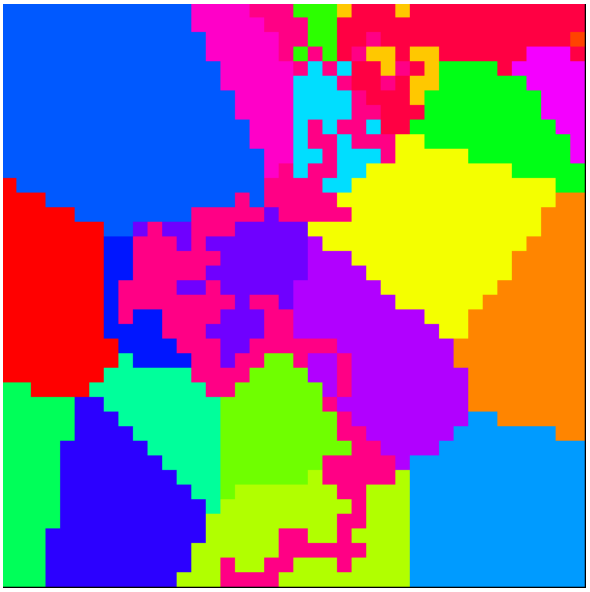
[3] I.Crespo. P4:diagrama de vornoi. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P4`.



(a) Diagrama de Voronoi inicial.



(b) Las fracturas no cruzaron, es apreciable que al penetrarse en un núcleo su propagación fue reducida.



(c) Cruzamiento de fractura favorecido por la propagación entre fronteras.

Figura 1: Crecimiento de fracturas en un diagrama Voronoi.

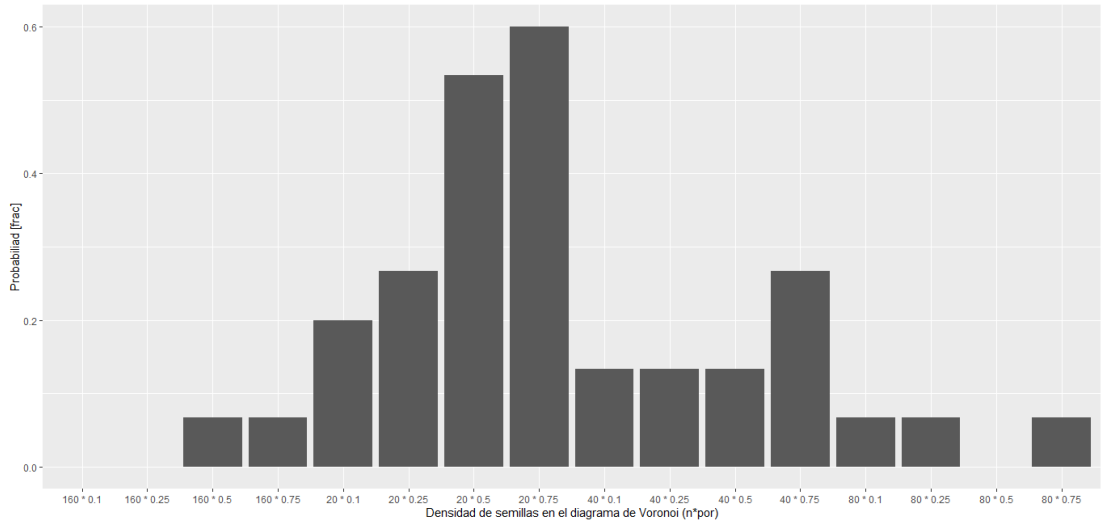
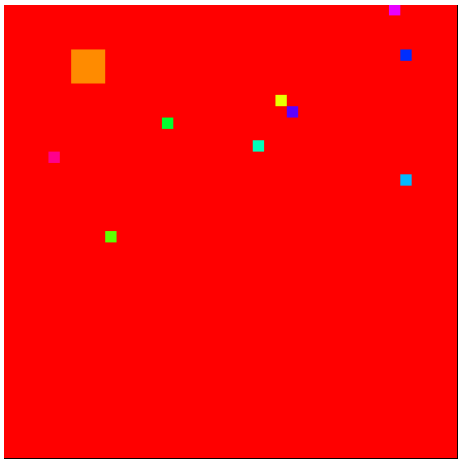
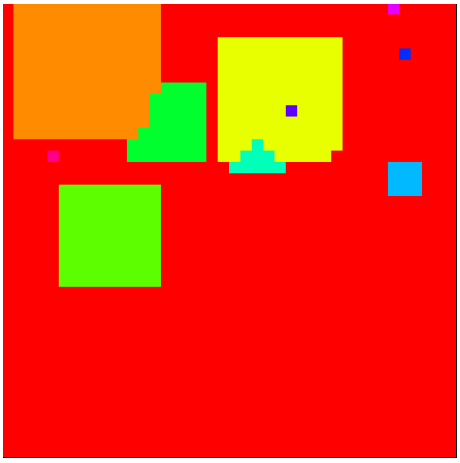


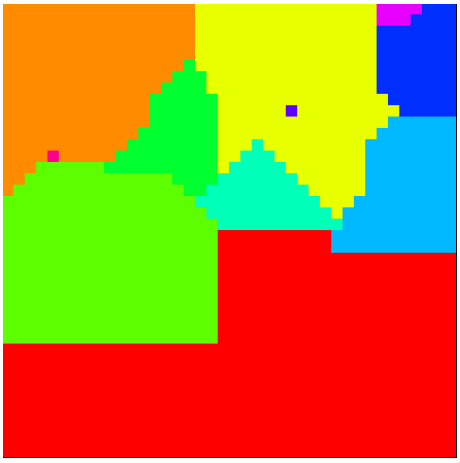
Figura 2: probabilidad de crecimiento en cada caso, en el eje x se presenta las dimensiones de la matriz n multiplicado por el factor k para determinar el número de semillas.



(a) Diagrama de Voronoi inicial $t = 1$.

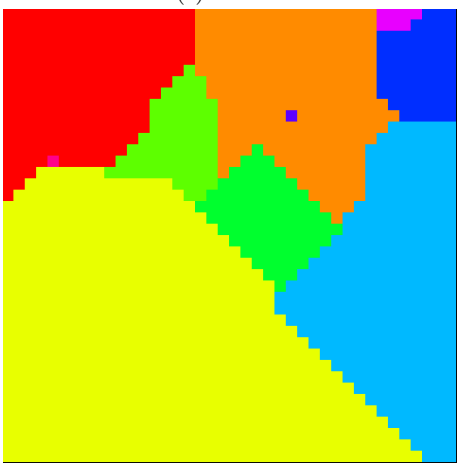


(b) $t = 6$.



(c) $t = 11$.

(d) $t = 21$.



(e) $t = 29$.

Figura 3: Crecimiento de un diagrama Voronoi con tasas de crecimiento diferentes.

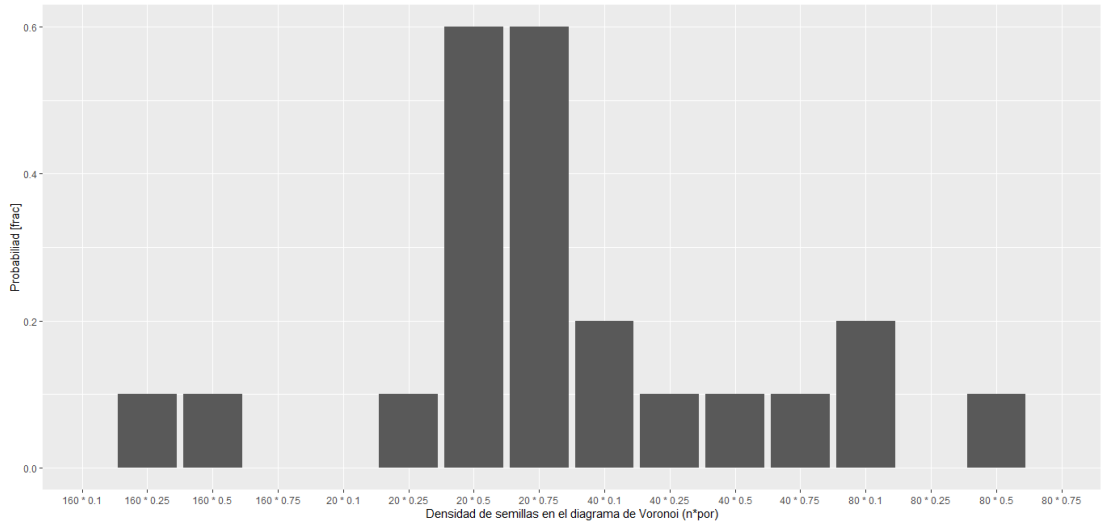
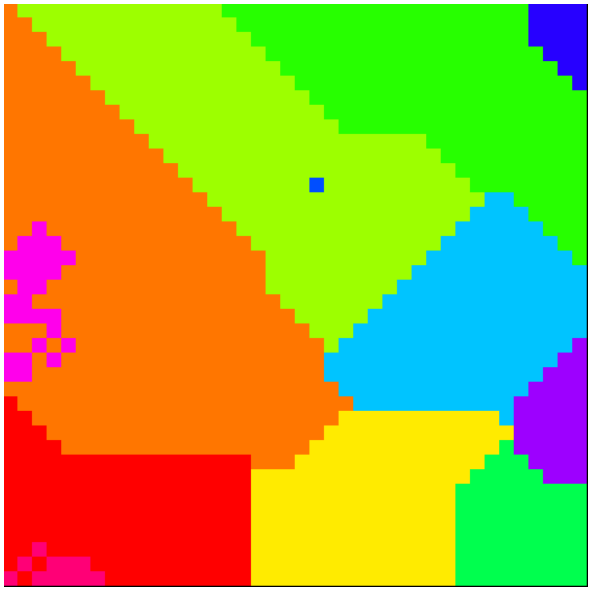


Figura 4: Probabilidad de crecimiento en un diagrama de Voronoi con tasas de crecimiento independientes para cada semilla, en el eje x se presenta las dimensiones de la matriz n multiplicado por el factor k para determinar el número de semillas.

(a) Existe cruzamiento entre fracturas.



(b) Las fracturas no cruzaron, es apreciable que al penetrarse en un núcleo su propagación fue reducida.

Figura 5: Propagación de dos fracturas para un mismo diagrama de Voronoi generado con semillas con diferentes tasas de crecimiento.

P5

Ismael Crespo

16 de marzo de 2022

1. Introducción.

Este trabajo presenta la obtención del valor de una integral definida de una función normalizada por medio del método de Monte-Carlo. Se pretende analizar como la cantidad de elementos que componen el área bajo la curva de la función afectan la precisión del valor obtenido al evaluado directamente por medio de la integración definida. La creación de este análisis se realizó creando rutinas en *R 4.1.2*.

2. Objetivos.

1.-Estudiar estadísticamente la convergencia de la precisión del estimado del integral con en método Monte Carlo, comparando con el valor producido por Wolfram Alpha, en términos del (1) error absoluto, (2) error cuadrado y (3) cantidad de decimales correctos, aumentando el tamaño de muestra.

2.-Realizar la estimación del valor de π de Kurt [3] y realizar el mismo análisis.

3. Programación en R.

Para realizar la estimación de la integral se definió una función $f(x)$ (ecuación 1) y esta se normalizo para que el área bajo la curva sea igual a 1 $g(x)$ (ecuación 2), posteriormente se construyó un generador de números aleatorios que genera valores dentro de esta función normalizada, para realizar este análisis de realizaron 20 réplicas variando el número de elementos a generar y a manera de estimar la integral definida de $g(x)$ (ecuación 3) entre 7 y 3 se realiza la sumatorio de los elementos en este rango, el código 1 realiza la rutina descrita.

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

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

$$\int_{3}^{7} \frac{2}{\pi} f(x)dx = 0{,}048834111126049310840642 37 \tag{3}$$

Código 1: Normalización de la función y estimación del área de bajo de la curva entre 7 y 3.

```
1  f <- function(x) { return(1 / (exp(x) + exp(-x))) }
2  g <- function(x) { return((2 / pi) * f(x)) }
3  generador  <- r(AbscontDistribution(d = g)) # creamos un generador
4  entre=function(n){gen=generador(n);return((pi/2)*sum(gen>=3&gen<=7)/n)}
```

El análisis de la precisión de realizo encontrando las diferencias absolutas, las diferencias cuadradas y la exactitud de cada uno de los decimales del resultado (véase el código 2). [1].

Código 2: Métodos de obtención de la precision de la estiamción comparada con el valor correcto obtenido de *Wolfram Alpha* [2].

```
 abso=abs(correcto-estimado)
absolutos=c(absolutos,abso)
abso_cuad=(correcto-estimado)**2
absolutos_cuad=c(absolutos_cuad,abso_cuad)
decimal=3
de=substring(as.character(estimado),1,decimal)
dc=substring(as.character(correcto),1,decimal)
while(de==dc){
de=substring(as.character(estimado),1,decimal)
dc=substring(as.character(correcto),1,decimal)
decimal=decimal+1
}
```

La segunda parte del trabajo realiza la estimación del valor de π despejado de la ecuación 4. El área del círculo se encuentra generando números aleatorios de manera uniforme, que al graficar se observan como en la figura 1a, posteriormente se encuentran los puntos cuya distancia euclidiana sea menor al radio definido previamente, por conveniencia los datos se generaron en el intervalo -1 a 1 tanto en x como en y, por lo tanto el radio del circulo es 1, el área graficada se observa en la figura 1b, la sumatoria de estos datos se considera el área del circulo de radio 1 y se sustituye en la ecuación 5 para estimar el valor de π (véase el código 3). Se realiza el mismo análisis de la precisión de la estimación en relación al valor de π como valor correcto.

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

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

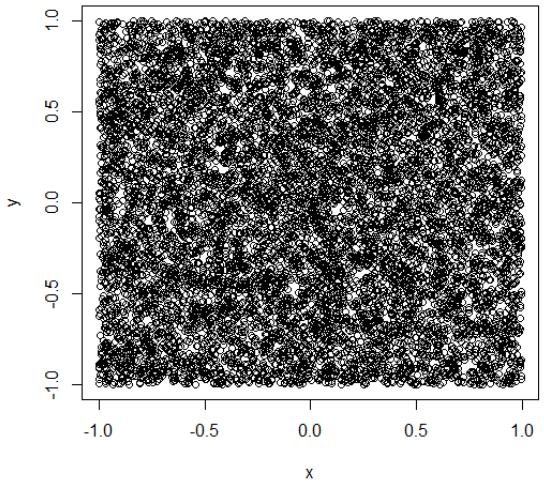
Código 3: Estimación del valor de π.

```
for (n in num){
for(r in 1:replicas)
x=runif(n,-1,1)
y=runif(n,-1,1)
d=numeric()
d=c(d,sqrt(x[i]**2 + y[i]**2))
dentro=d<=1
estimado=sum(dentro)*4/n
```

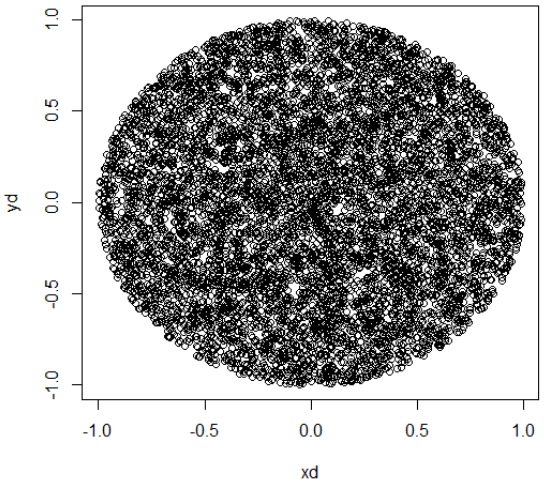
4. Resultados.

La figura 2 presenta las precisiones de cada estimación variando el valor de n, en la figura 2a se analiza las diferencias absolutas, basándose en estos resultados podríamos determinar que un valor de $n = 1000000$ nos dará valores con precisión aceptable considerando que ya no se observa un cambio importante para valores de n mayores, en la figura 2b se observa las diferencias cuadradas, utilizando este análisis como referencia nuestras deducciones cambian y utilizando un valor de $n = 10000$ tendríamos valores aceptables bajo este criterio, el análisis de la exactitud del valor, es decir cuantos decimales son exactamente iguales al valor correcto se observa en la figura 2c, este criterio nos obliga a utilizar valores altos de n para concluir que tenemos valores precisos.

El análisis de la precisión en la estimación de π se presenta en la figura 3. Las conclusiones referente a que valor de n da los valores mas precisos son similares a los obtenidos en la primera parte. La figura 3a presenta la precisión en función de la diferencia absoluta,3b presenta la precisión en función de la diferencia cuadrada y 3c presenta la precisión en función de la exactitud de los decimales. Al ser una rutina mas compleja se tuvo como limitante los valores máximos permisibles de n para simular.

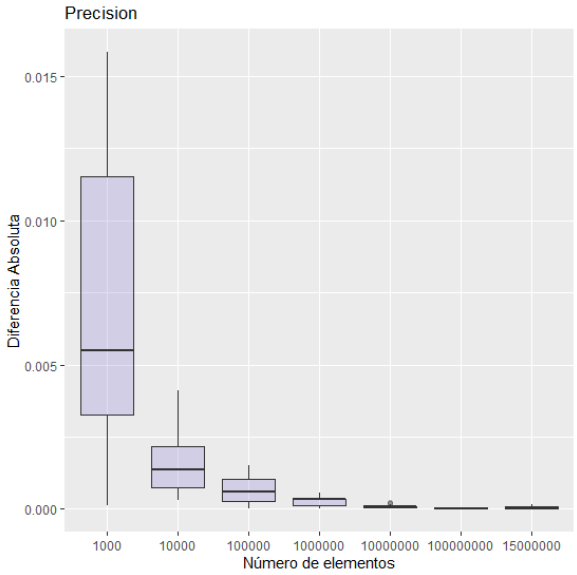


(a) Números uniformemente posicionados en x y y.

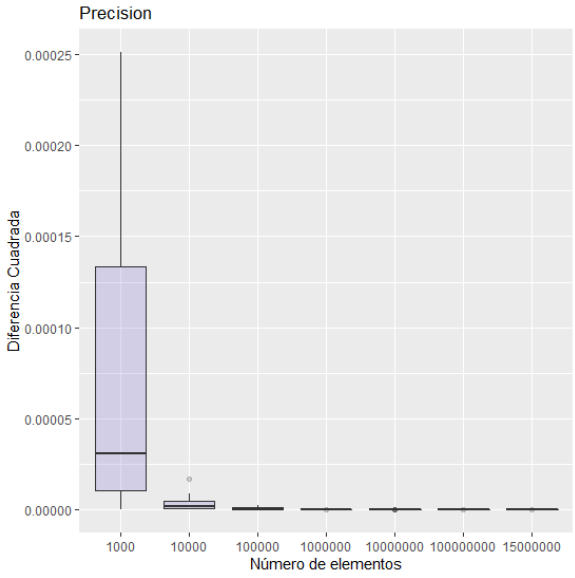


(b) x y y dentro de la circunferencia obtenidos por medio de la distancia euclidiana menor o igual a 1.

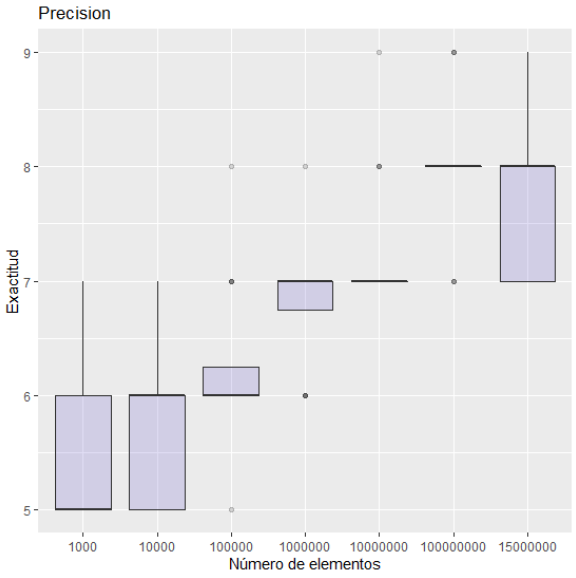
Figura 1: Área de un circulo para despejar.



(a) Diferencia absoluta de la estimación en relación al valor de n.

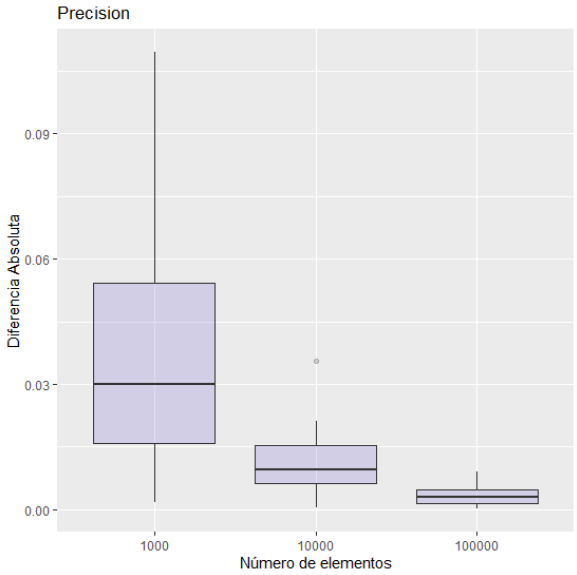


(b) Diferencia cuadrada de la estimación en relación al valor de n.

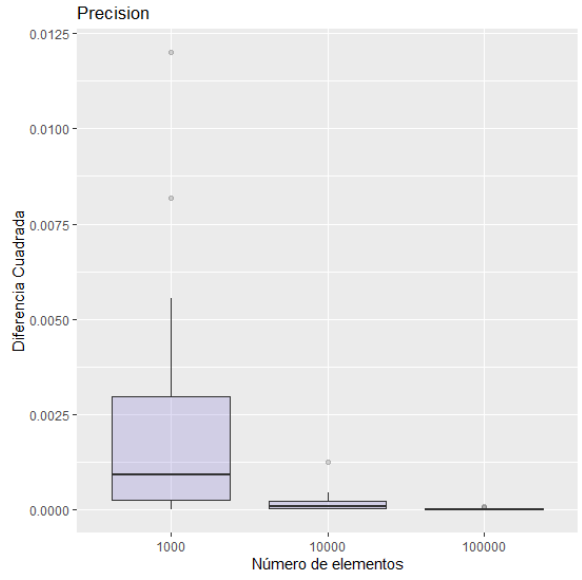


(c) Exactitud de los decimales de la estimación en relación al valor de n.

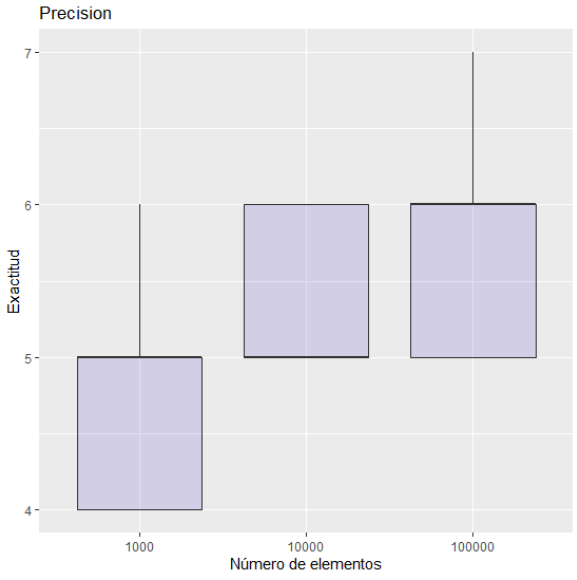
Figura 2: Diferentes análisis de la precisión de la estimación.



(a) Diferencia absoluta de la estimación de π y el valor correcto de π en relación al valor de n



(b) Diferencia cuadrada de la estimación de π y el valor correcto de π en relación al valor de n.



(c) Exactitud de la estimación de π y el valor correcto de π en relación al valor de n.

Figura 3: Diferentes análisis de la precisión de la estimación con respecto al valor correcto de π.

5. Conclusiones.

La estimación del área bajo la curva utilizando el método de Monte-Carlo es útil, sin embargo, para obtener resultados óptimos es necesario aumentar el número de elementos bajo la curva a manera de aumentar la definición de el área. Hay valores que, dependiendo la precisión requerida y el tipo de precisión requerida, nos dan resultados aceptables son rutinas cortas.

Referencias

[1] I.Crespo. P5:método de monte-carlo. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P5`.

[2] 2022 Wolfram Alpha LLC. 2022. URL `https://www.wolframalpha.com/input?i=integrate`.

[3] W.Kurt. 6 neat tricks with monte carlo simulations — count bayesie. 2022. URL `https://www.countbayesie.com/blog/2015/3/3/6-amazing-trick-with-monte-carlo-simulations`.

P6

Ismael Crespo

23 de marzo de 2022

1. Introducción

El contenido de este trabajo presenta la simulación numérica de una epidemia basado en elementos que pueden ser Infectados(I), Susceptibles (S) y Recuperados(R), las infecciones se dan al inicio de manera pseudoaleatoria entre la población y después al existir una convivencia (distancia euclidiana corta) los elementos sanos pueden infectarse, una vez infectados después de un determinado número de pasos es posible que un elemento sea recuperado y genere inmunidad. Se analiza el impacto en el desarrollo de la pandemia en relación al aislamiento (reducción de la movilidad de los infectados) y la inmunidad desarrollada por una vacunación en el inicio de la pandemia. La creación de este análisis se realizó creando rutinas en *R 4.1.2*. Los efectos de la reducción en la movilidad y la vacunación se realizan modificando el código desarrollado por E. Schaeffer [2].

2. Objetivos

1.-Estudia el efecto de contención en el sentido de que agentes infectados reduzcan su velocidad de movimiento a la mitad durante su infección. Determina con pruebas estadísticas adecuadas si este cambio produce un efecto significativo en la magnitud de la epidemia (la altura del pico en la curva del porcentaje de infectados por iteración) y en la velocidad de ella (la iteración en la cual se llega por la primera vez al valor pico).

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

3. Programación en R

La simulación de la pandemia se basa en asignar caracteres (I,S y R) al número de elementos propuestos en la variable n. En función a la probabilidad pi un elemento en estado inicial S pasará a estar en estado I, un elemento en estado S se va a contagiar con una probabilidad de contagio pc que esta en función de la distancia euclidiana entre los elementos. Los elementos se mueven en una velocidad aleatoria para cada caso en una región delimitada a $x = 1$ y $y = 1$ y esta velocidad, dx y dy en cada paso, es una fracción de la longitud en x y y asignada aleatoriamente el código P6 en la referencia [1] realiza la rutina descrita. El código 1 muestra como se asignan los factores I, R y S, para la primera parte del trabajo se muestra como la velocidad se reduce dividiendo entre la variable *movilidad* la velocidad asignada al inicio a los elementos que son infectados al inicio y conforme avanza la epidemia. El día en el que se llega al pico se obtiene utilizando la funicón `which.max`.

Código 1: Asignación de los factores para los elementos y reducción de la velocidad con la variable *movilidad* para los elementos infectados.

```
1  movilidad=c(1,2,3,4)
2 for( mov in movilidad){
```

```
for(repeticion in 1:rep){
agentes <- data.frame(x = double(), y = double(),
                      dx = double(), dy = double(),
                      estado  = character())
for (i in 1:n) {
    e <- "S"
    if (runif(1) < pi) {
        e <- "I"
    }
for(i in 1:n){
        if(agentes$estado[i] == "I"){
         agentes$dx[i]=agentes$dx[i]/mov
         agentes$dy[i]=agentes$dy[i]/mov
 }}
```

Para el efecto de la vacunación los elementos son vacunados, al inicio de la epidemia en función de la probablidad pv, la vacuna asignara el factor R a los elementos para generar una inmunidad ante los infectados. La movilidad no se modifico para evaluar solamente el efecto de la vacunación. Para ambos analisis de vacunación y movilidad se realizaron 10 replicas y fueron analizadas por medio de diagramas caja-bigote (véase el código 2).

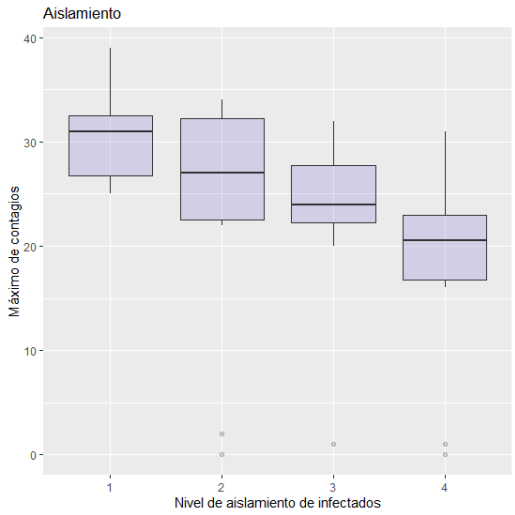
Código 2: Vacunación en función de pv para generar inmunidad al inicio de la epidemia.

```
for(pv in seq(0.0,1 , by = 0.1))
for (i in 1:n) {
    e <- "S"
     vacuna=runif(1)
     if (runif(1) < pi) {
        e <- "I"
     }
     if ( vacuna< pv) {
        e <- "R"
    }
```

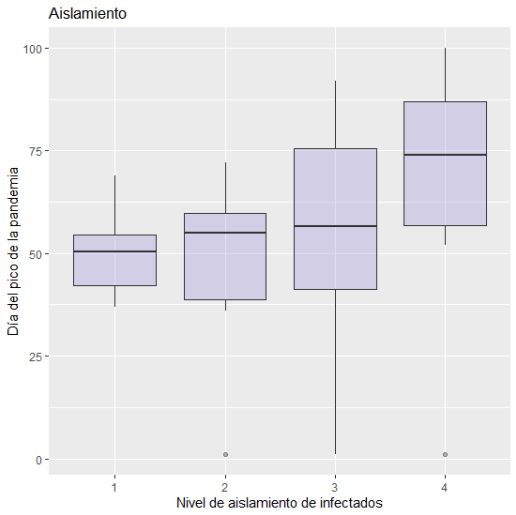
4. Resultados

La figura 1 presenta el impacto que tiene la reducción de la movilidad, similar a lo que ofrece el aislamiento, de los elementos infectados en la pandemia, en la figura 1a se analiza el porcentaje de infectados en el pico de la pandemia según el nivel de aislamiento al que fueron sometidos los elementos infectados, en la figura 1b se observa el día en el que se llegó al pico de la pandemia. La figura 1c y la figura 1d presentan la curva de la pandemia en el caso de que la movilidad no se restringe para los casos infectados y para cuando la movilidad se restringe en 1/4 de su movilidad original. Los resultados muestran una clara reducción en los contagios y una epidemia con forma de meseta alargada durante los 100 días de simulación para la reducción de la velocidad a 1/4 de la original cuando se es infectado . En las secuencias gráficas del desarrollo de las epidemias de la figura 1, disponibles en como material complementario en el repositorio de la practica [1], se observa como los elementos infectados al reducir su velocidad no llevan la infección, los contagios se dan solo por aquellos que se acercan a ellos.

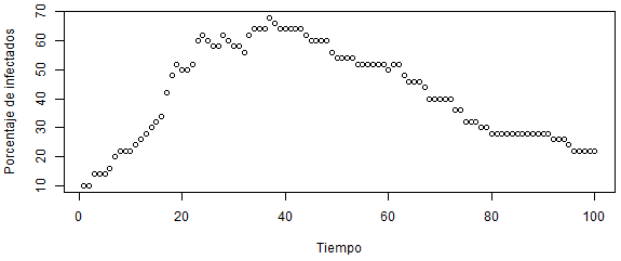
En la figura 2 se presenta el mismo análisis del máximo de contagios y el día en el que se llega al pico de la pandemia variando pv que determina el alcance de la vacunación en un estado inicial para desarrollar inmunidad a una fracción de la población ante posibles infecciones, el valor de pv varia de 0 a 1, interpretamos a 0.1 como una vacunación muy deficientes y a 1 como una vacunación universal. En la figura 2a se presenta el máximo de contagios en relación a la vacunación y la figura 2b presenta el día en el que se llega al pico de la pandemia, la figura 2c y la figura 2d presentan el comportamiento de la pandemia para la vacunación cuando $pv = 0{,}2$ y $pv = 0{,}7$. En las secuencias gráficas del desarrollo de las epidemias de la figura 2, disponibles en como material complementario en el repositorio



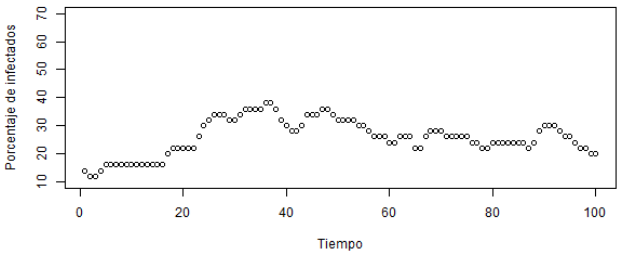
(a) Porcentaje máximo de infectados según la restricción de la movilidad de los infectados.



(b) Día en el que se llega al pico de la pandemia según la restricción de la movilidad de los infectados.

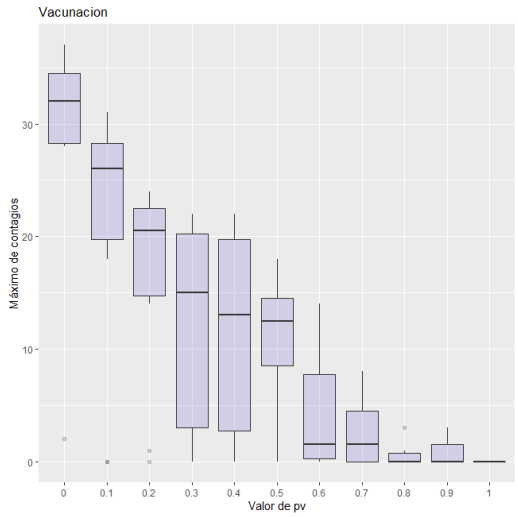


(c) Curva de la pandemia sin restricción de movilidad.

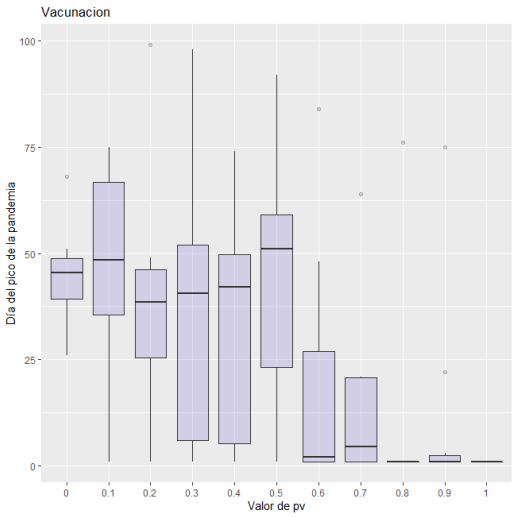


(d) Curva de la pandemia con restricción a 1/4 de movilidad.

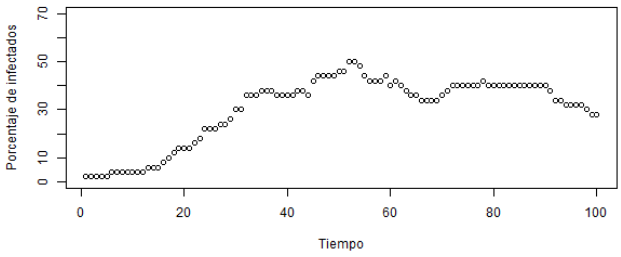
Figura 1: Desarrollo de las pandemia en función de la movilidad de los infectados.



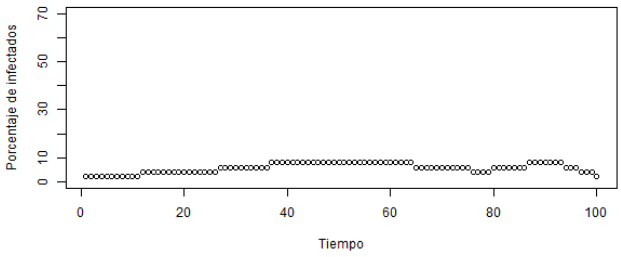
(a) Porcentaje máximo de contagios en función de la vacunación.



(b) Día del pico de la pandemia en función de la vacunación.



(c) Curva de la epidemia $pv = 0{,}2$.



(d) Curva de la pandemia $pv = 0{,}7$.

Figura 2: Desarrollo de la pandemia en función del nivel de vacunación.

de la practica [1], se observa el comportamiento de la población cuando hay inmunidad entre la población al inicio de la pandemia.

5. Conclusiones

Tanto la reducción en la movilidad como la vacunación son métodos efectivos para controlar la pandemia, se destaca la vacunación como el método mas efectivo cuando se consigue un alcance importante de la vacunación dentro de la población, sin embargo el aislamiento ofrece una herramienta para contener el contagio sin necesidad de una campaña de vacunación cuando no se tiene acceso a esta.

Referencias

[1] I. Crespo. P6: Sistema multitangente. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P6`.

[2] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

P7

Ismael Crespo

30 de marzo de 2022

1. Introducción

Durante esta practica se pretende buscar un valor óptimo, el mayor evaluado dentro de una función $g(x, y)$, por medio de una búsqueda local dentro de la matriz creada, seleccionado la mejor opción en cada iteración. Se utiliza como base el trabajo presentado E. Schaeffer [2] donde se realiza una búsqueda dentro de una función $f(x)$.

2. Objetivos

1.-Crea una visualización (animada) de cómo proceden por lo menos 5 réplicas simultáneas de por lo menos 500 pasos de la búsqueda encima de una gráfica de proyección plana.

2.-Simular el recocido simulado para optimizar una función $f(x)$, se genera para la solución actual x un sólo vecino $x' = x + \Delta x$ (algún desplazamiento local). Se calcula $\delta = f(x) - f(x')$. Si $\delta > 0$, siempre se acepta al vecino x' como la solución actual ya que representa una mejora. Si $\delta < 0$, se acepta a con probabilidad $\exp(\delta/T)$ y rechaza en otro caso. Aquí T es una temperatura que decrece en aquellos pasos donde se acepta una empeora; la reducción se logra multiplicando el valor actual de T con $\varepsilon < 1$, como por ejemplo 0,955. Examina los efectos estadísticos del valor inicial de T y el valor de ε en la calidad de la solución, es decir, qué tan alto el mejor valor termina siendo.

3. Programación en R

La búsqueda comienza seleccionando un punto en la matriz, coordenada en y y coordenada en x, este ejercicio se replica 5 veces para tener 5 puntos en donde para el primer objetivo se mueve aleatoriamente entre sus vecinos. El código 1 muestra como se generan aleatoriamente y posteriormente como es la dinámica de movimiento aleatorio de cada replica hacia sus vecinos, la mejor opción encontrada (`best`) es seleccionada utilizando un ciclo `if` con la condición de que este valor cambiará a uno nuevo cada que una replica actual sea mayor al mejor valor obtenido hasta este tiempo.

Código 1: Generación de cada replica y movimiento aleatorio hacia los vecinos

```
#Generaci n de cada replica
for(i in 1:replicas){
 curr_x[i] <- sample(seq(low, high, step),1)
 curr_y[i]<- sample(seq(low, high, step),1)
 curr[i]<-z[as.character(curr_y[i]),as.character(curr_x[i])]
 best_y[i]<-curr_y[i]
 best_x[i]=curr_x[i]
 bests[i]=curr[i]
}
#Movimiento hacia un vecino
 for(i in 1:replica){
```

```
    min_fila<-max(curr_y[i]- delta, low)
    max_fila <-min(curr_y[i]+ delta,high)
    min_col<-max(curr_x[i] - delta, low)
    max_col <-min(curr_x[i] + delta,high)
     filas<-(seq(min_fila,max_fila,delta))
     columnas<-(seq(min_col,max_col,delta))
     #vecindad<-z[as.character(filas),as.character(columnas)]
     curr_y[i]<- sample(filas,1)
     curr_x[i] <- sample(columnas,1)
```

En la segunda parte del trabajo el movimiento no es completamente aleatorio, tal como se describe en el objetivo 2, cada replica se va a mover siempre que la posición vecina, seleccionada aleatoriamente, evaluada en la función sea mejor, cuando esta posición no represente una mejora en la búsqueda se va a mover con una probabilidad dependiente de la temperatura (véase el código 2). Para consulta detallada de las rutinas desarrolladas véase la referencia [1]

Código 2: Selección de un vecino con una probabilidad dependiente de la temperatura si el valor del vecino seleccionado no es mejor.

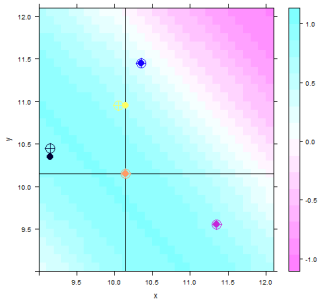
```
for(f in 1:length(filas)){
        prob_y=sample(filas,1)
        prob_x=sample(columnas,1)
        prob <-z[format(round(prob_y, 2),
        nsmall = 2),format(round(prob_x, 2), nsmall = 2)]
        resta=curr[i]-prob
         if(resta<0){
          curr_y[i]<- prob_y
          curr_x[i] <- prob_x
          curr[i] <-z[format(round(curr_y[i], 2),
          nsmall = 2),format(round(curr_x[i], 2), nsmall = 2)]
          break
         }
        if(resta>0){
        expo=exp(-abs((resta)/temperatura))
        pick=runif(1)
        if(pick<expo){
          curr_y[i]<- prob_y
          curr_x[i] <- prob_x
          curr[i] <-z[format(round(curr_y[i], 2),
          nsmall = 2),format(round(curr_x[i], 2), nsmall = 2)]
          temperatura=temperatura*e
          break
         }}}
```

Para

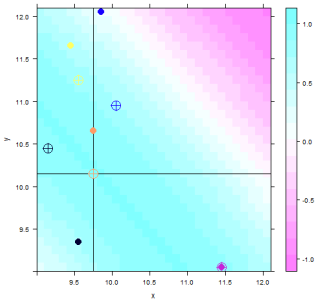
4. Resultados

Las secuencias gráficas de la búsqueda aleatoria del mejor resultado se presenta en la figura 1, y esta disponible de manera extensa como material complementario en el repositorio de la practica [1], se observa que la calidad búsqueda aleatoria depende de donde aparece por primera vez la replica, ya que la función en los limites evaluadas de 7 a 12 tanto en x y y acumula las mejores opciones en una sola zona.

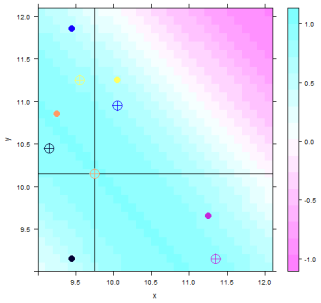
Para el análisis entre la búsqueda aleatoria y la búsqueda probabilística las rutinas descrita en la sección 3 se



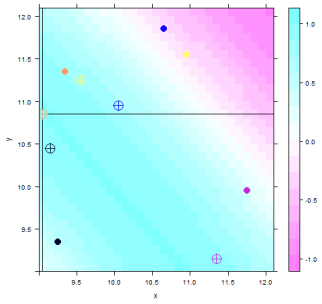
(a) Tiempo=2.



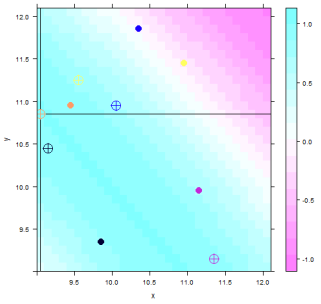
(b) Tiempo=50.



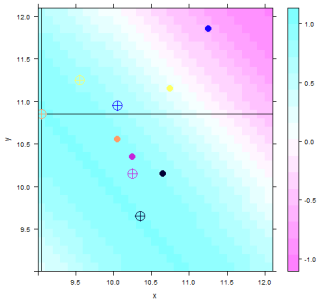
(c) Tiempo=100.



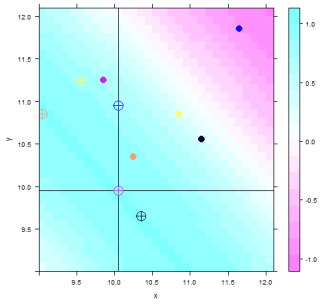
(d) Tiempo=150.



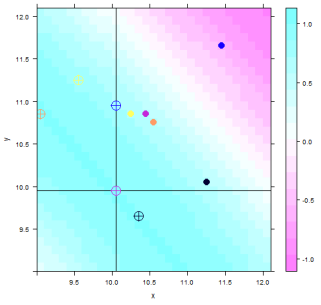
(e) Tiempo=200.



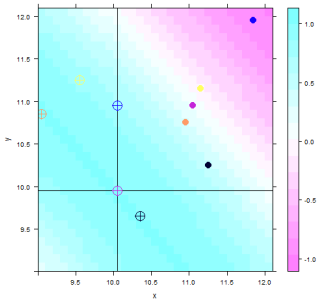
(f) Tiempo=250.



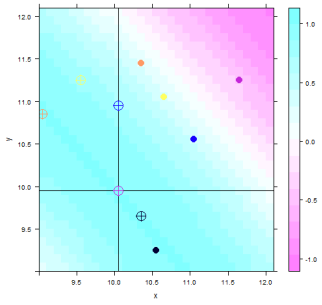
(g) Tiempo=300.



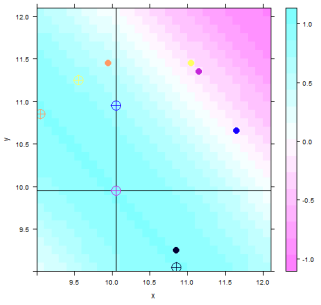
(h) Tiempo=350.



(i) Tiempo=400.



(j) Tiempo=450.



(k) Tiempo=500.

Figura 1: Búsqueda aleatoria del máximo en el plano,el circulo con la cruz dentro representa la mejor opción para cada replica en dado tiempo, el circulo representa la búsqueda actual y el punto de cruce entre la línea vertical y horizontal, representa la mejor opción de todas las replicas.

evaluaron 15 veces y se analizaron estadísticamente, se considera el tiempo en el que se llega al resultado mayor en cada una de las búsqueda (cuadro 1), igualmente se analiza la sumatoria de las 5 replicas, interpretando que a sumatorias mayores, las replicas en conjunto se acercaron más al resultado óptimo (cuadro 2). Para la rutina aleatoria durante 500 pasos no siempre se obtuvo el resultado óptimo y se excluyeron del análisis. Es claro que los tiempos en los que se obtiene el valor óptimo mejora al utilizar la segunda rutina y las replicas en conjunto se acercan a valores mas cercanos al mejor. El cuadro 3 presenta los valores p para ambos análisis, ambas relaciones tienen valores menores a 0,05 por lo que fueron aceptadas como significativas.

Cuadro 1: Datos estadísticos de los tiempos en encontrar el valor máximo de las replicas. El Tiempo 1 hace referencia a las replicas aleatorias y el Tiempo 2 a la simulación de recocido térmico.

Estadistica	N	Mediana	Min	Max
Tiempo 1	12	117.58330	22	230
Tiempo 2	12	19.91667	4	87

Cuadro 2: Datos estadísticos de las sumatorias de las replicas. La Sumatoria 1 hace referencia a las replicas aleatorias y la Sumatoria 2 a la simulación de recocido térmico.

Estadistica	N	Mediana	Min	Max
Sumatoria 1	15	4.76212	3.82853	4.99983
Sumatoria 2	15	4.99990	4.99985	4.99993

Cuadro 3: Valores p entre las sumatorias y los tiempos de las replicas para encontrar el valor máximo.

Variable 1	Variable 2	$p - value$
Sumatoria 1	Sumatoria 2	$7{,}94 \times 10^{-7}$
Tiempo 1	Tiempo 2	$1{,}553 \times 10^{-4}$

5. Conclusiones

La búsqueda aleatoria es muy dependiente de la posición inicial de las replicas, la forma y los limites en los que se evalúa la función determinan el plano donde se realizará la búsqueda. La búsqueda que se rige por la mejor opción entre los vecinos y probabilístico (dependiente de la temperatura) si la selección es peor mejora considerablemente las rutinas.

Referencias

[1] I. Crespo. P7: Búsqueda local. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P7`.

[2] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

P8

Ismael Crespo

6 de abril de 2022

1. Introducción

Esta práctica introduce la simulación de los modelos de urnas para analizar los fenómenos donde nos interesa más conocer cuánto hay de algo pero no su tratamiento individual [2]. Este modelo se utilizará para simular un filtrado de todas aquellas partículas de mayor tamaño a un valor critico c a partir de la agregación y segregación de cúmulos con un número de partículas individuales inicial.

2. Objetivos

1.- Estudiar el efecto de la tasa k/c, usando por lo menos cinco valores distintos para ella, suponiendo que partículas con tamaño mayor a c se filtraran y analizar el porcentaje de las partículas que se lograría filtrar por iteración.

2.-Determinar cómo el momento idóneo de filtrado depende del valor de c. ¿Qué todo cambia y cómo si c ya no se asigna como la mediana inicial sino a un valor menor o mayor?

2.-Estudiar el efecto del parámetro suavizante d en el desempeño de filtrado si la meta es recuperar la mayor cantidad posible de partículas en el proceso, identificar la mejor iteración para realizar el filtrado.

3. Programación en R

La agregación y segregación de las partículas se realiza utilizando las rutinas presentadas por E.Schaeffer [2] a partir de la generación de k cúmulos de diferentes tamaños con distribución normal a partir de un total de n partículas . Una partícula se va a fraccionar con una probabilidad mayor cuando su tamaño este cerca del valor c en una curva sigmoidal y cerca de c en una curva exponencial para la unión de partículas, fomentando la agregación entre cúmulos pequeños. Para la primera parte del trabajo se aceptó la mediana de los tamaños de los cúmulos de partículas como el valor de c.

Para la primera parte del trabajo, se analiza la relación n/k es decir la densidad de cúmulos a partir de una población controlada de partículas. Para este trabajo el valor de n se mantuvo constante y se varió k para trabajar con valores de k/c =0.0015, 0.0050, 0.0100, 0.0175 y 0.0250. Al final de cada iteración de agregación y segregación de partículas se realiza una filtración hipotética para estimar el porcentaje de partículas que se hubieran filtrado manteniendo el valor de c como la mediana, se realizan 10 filtraciones en cada una de la 10 réplicas simuladas para el análisis. En el código 1 se muestra la selección de los valores de k para cada situación.

Código 1: Declaración de los valores de k y n y su uso en el ciclo `for`.

```
1 #valores de k, n se mantiene constante
2 k_i <-c(1500,5000,10000,17500,25000)
3 n_i <- c(1000000)
4 #iteracion con cada valor de k
5  for(jj in 1:length(k_i)){
```

```
for(j in 1:replicas){
k=den$k_i[jj]
n=den$n_i[jj]
originales <- rnorm(k)
cumulos <- originales - min(originales) + 1
cumulos <- round(n * cumulos / sum(cumulos))
}}
```

Código 2: Filtrado de cúmulos de partículas mayor a c

```
 filtrados=numeric()
    for(i in 1:length(freq$tam)){
    if(freq$tam[i]>c){
    filtrados=c(filtrados,(freq$tam[i]*freq$num[i]))
    }}
    filtracion=c(filtracion,(sum(filtrados)/n))
```

En las rutinas presentadas en [2] los cúmulos se separan utilizando el modelo de urnas para agrupar los cúmulos del mismo número de partículas, estos agrupamientos se utilizan para realizar el filtrado como se muestra en el código 2, se multiplica la frecuencia de cada cúmulo por el número de partículas que lo constituyen para obtener el total de partículas filtradas.

Para la segunda parte del trabajo se buscó identificar el mejor paso para filtrar variando el valor de c, se utiliza la mediana de los cúmulos como valor base y posteriormente se utiliza como factor 0,95 y 1,05 para multiplicar por el valor original de c y así obtener valores por encima y por debajo de la mediana utilizada originalmente, véase el código 3. De la misma manera se realiza el tercer análisis, variando el valor de d utilizando como factores divisores de la desviación estándar de los tamaños de cúmulos a $1, 2, 3, 4$. El filtrado para ambos casos se realizó como se muestra en el código 2. Para consulta ampliada de las rutinas consulte los códigos publicados en [1]

Código 3: Variación del valor de c.

```
crit_value=c(0.95,1,1.05)
.....
for(jj in crit_value){
.....
cmediana <- median(cumulos) # tamanio critico de cumulos
c=cmediana*jj
......
}
```

4. Resultados

El porcentaje de partículas recuperadas variando la relación k/c se muestra en la figura 1, se observa un óptimo de filtración con la relación $k/c = 0{,}005$ en el paso 4, manteniéndose constante posterior a este paso. Es apreciable como la relación $k/c = 0{,}0015$ presenta los mejores porcentajes de filtración solamente en las primeras dos iteraciones.

El análisis para la filtración variando el valor de c y d se realiza de la misma manera y se presenta en la figura 2 y 3. Se observa una dependencia importante entre estos valores para mejorar la filtración y depende de en que paso se realice la filtración el valor de d y c funcionará mejor, sin embargo la filtración observa mejoría hasta los pasos 4 y 5 para después mantenerse constantes para cada valor de d y c.

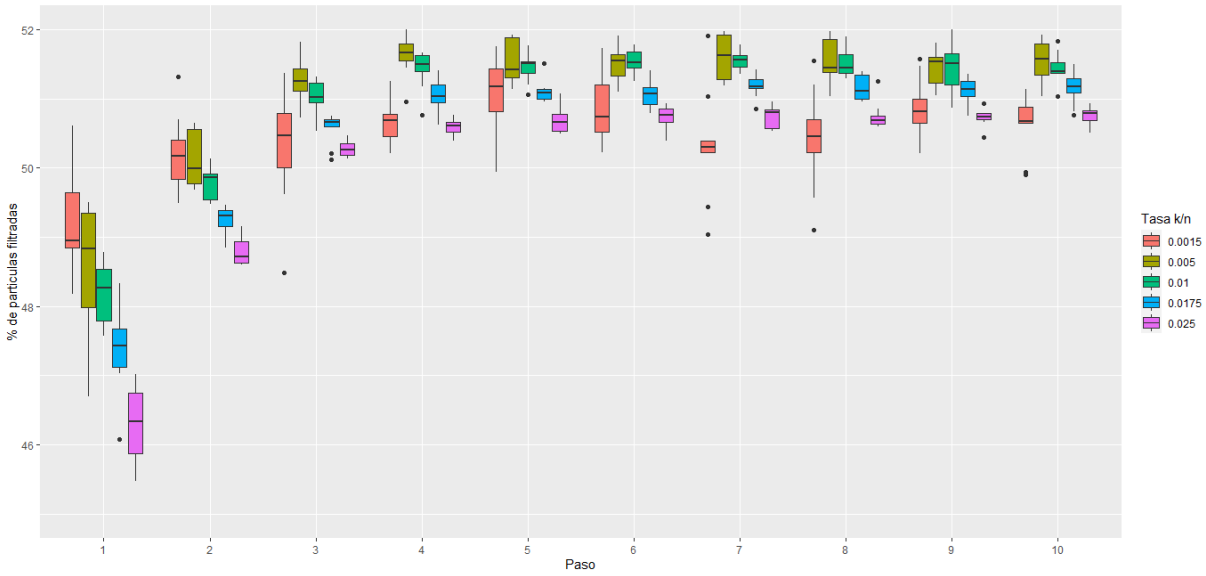


Figura 1: Porcentaje de partículas filtradas en cada paso de la réplica según el valor de k/c

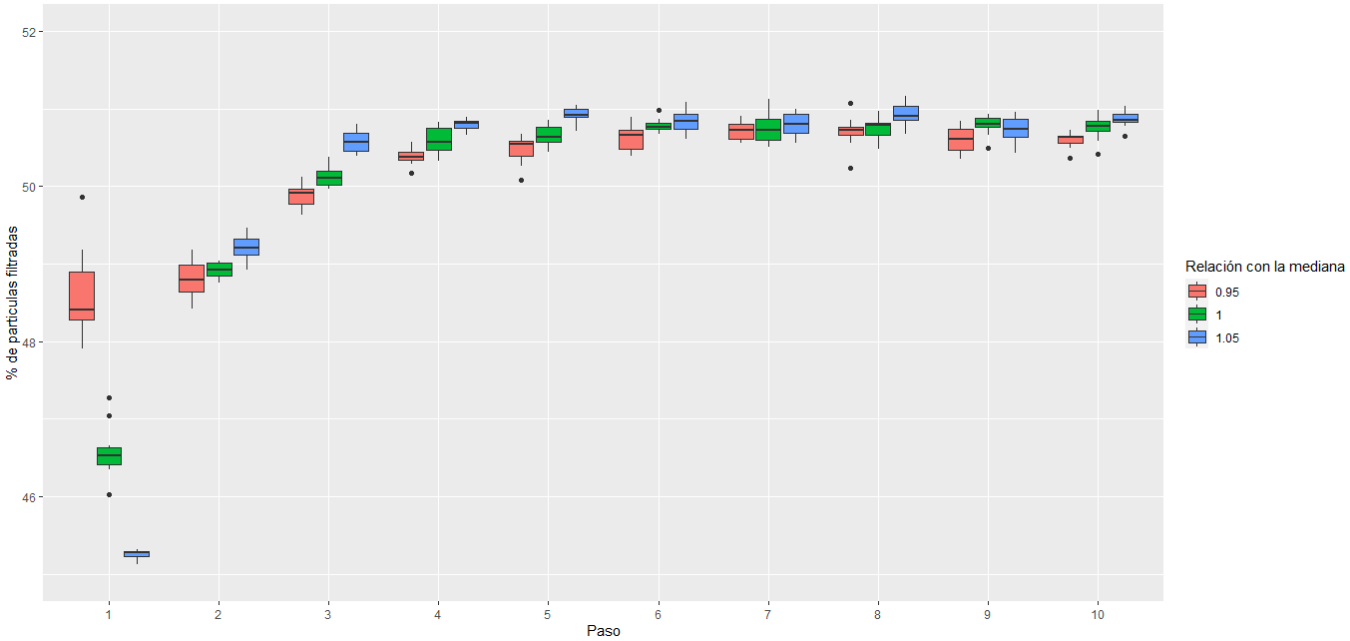


Figura 2: Porcentaje de partículas filtradas en cada paso de la réplica según el valor de c.

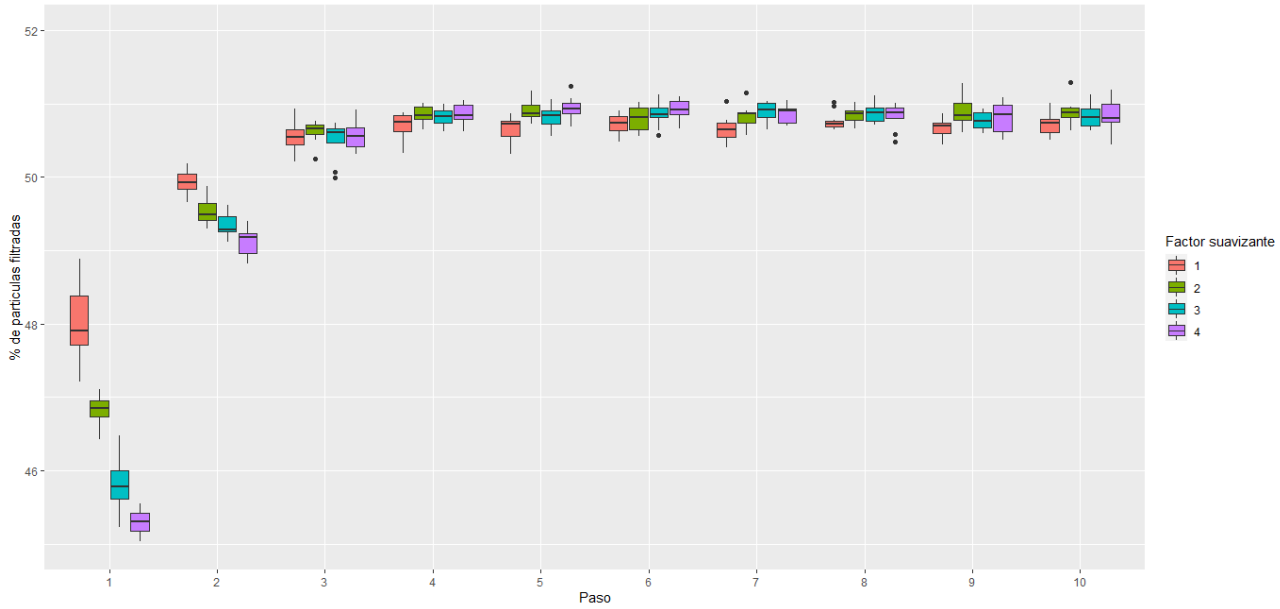


Figura 3: Porcentaje de partículas filtradas en cada paso de la réplica según el valor de d.

5. Conclusiones

La filtración de cúmulos de partículas tiene mejores rendimientos en el paso 4, manteniéndose constantes para los pasos posteriores, se toma como mejor relación $k/c = 0{,}005$ para esta situación, si la filtración se ve obligada a realizarse en tiempos cortos, es necesario utilizar relaciones k/c menores a 0,005. De manera similar los resultados de la filtración en cada paso dependen del valor de d y c, si la filtración se realiza a tiempos cortos se es mejor utilizar valores de c menores a 1, por otra parte valores de d mayores nos darán mejores resultados de filtración si esta se tuviese que realizar en las primeras iteraciones.

Referencias

[1] I. Crespo. P8: Modelo de urnas. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P8`.

[2] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

P9

Ismael Crespo

27 de abril de 2022

1. Introducción

Esta práctica introduce la simulación de fenómenos de atracción y de repulsión entre n partículas que están contenidas en un espacio. 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 euclideana entre las partículas (éstas son reglas inventadas de interacción para efectos de demostración). De igual manera se proponen reglas de atracción y repulsión relacionadas con la magnitud de la masa de cada partícula.

2. Objetivo

Estudiar la distribución de velocidades de las partículas en relación con la magnitud de la carga, y la masa de las partículas.

3. Programación en R

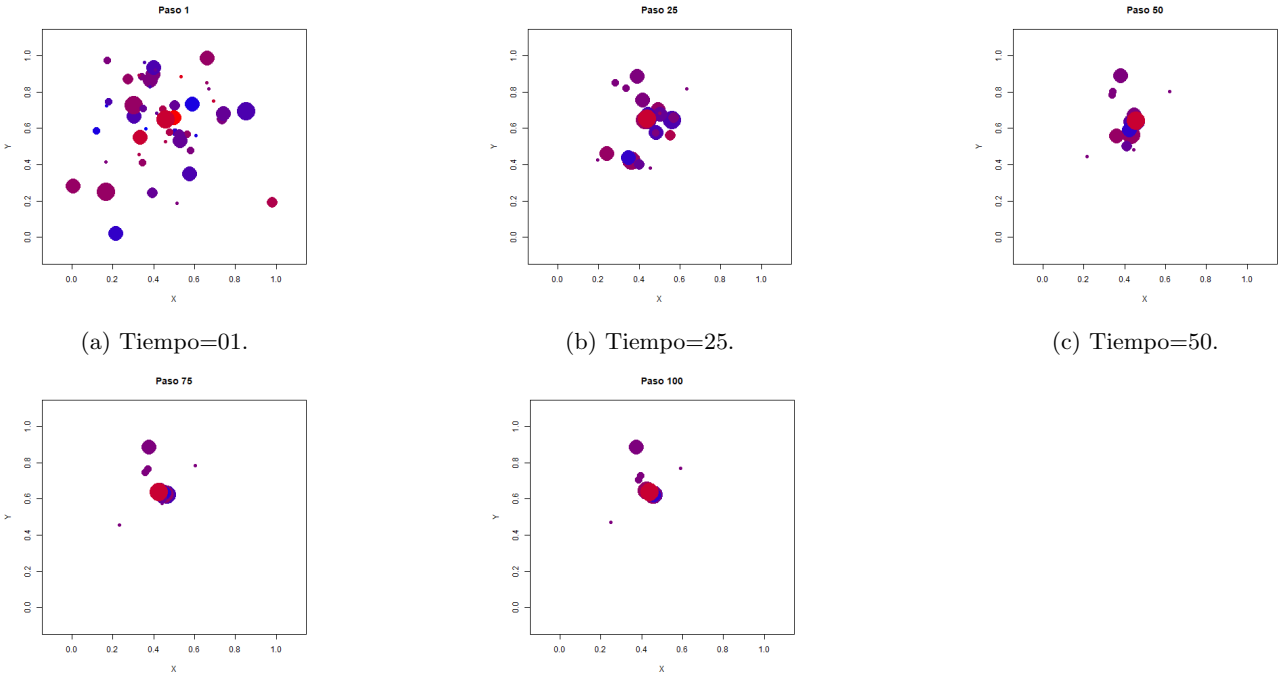
La creación y posicionamiento de partículas con cargas definidas y normalizadas se realiza utilizando las rutinas elaboradas por E.Schaeffer [2], se realizó una modificación a las rutinas originales agregando un valor de masa característico para cada partícula, el valor de la masa se seleccionó aleatoriamente de 1 a 5 y se utilizó la ecuación 1 para definir la magnitud de la fuerza, la dirección de esta fuerza (repulsión o atracción) se mantendrá igual a la definida por el signo de sus cargas.

$$F = \frac{G \times m_1 \times m_2}{r^2} \tag{1}$$

En la ecuación 1, G es la gravedad en la tierra, m_1 es la masa de la partícula en turno y m_2 es la masa de cada una de las otras partículas que existen en el sistema e interactúan con la partícula de masa m_1. En el código 1 se presenta como se agregó a la rutina el parámetro de la masa dentro del cuadro de datos p, la regla presentada en la ecuación 1 para obtener la fuerza del movimiento se ajustó utilizando la variable `suav` para controlar que no existieran desplazamientos fuera del rango en el que se realizó la simulación. Por medio de un ciclo `for.each` se realizó el análisis en paralelo para la interacción de cada partícula con todas las demás durante 100 pasos (código 2). Para consulta a detalle de la rutina véase los códigos publicados por I.Crespo [1].

Código 1: Repulsión y atracción de las partículas debido a la masa.

```
1 p <- data.frame(x = rnorm(n), y=rnorm(n),
2 c=rnorm(n),m=sample(1:5,n,replace=T) )
3
4 masa=(gra*mi*mj)*(suav)/(((sqrt(dx^2 + dy^2))/2)^2 + eps)
5         carga=abs(ci - cj) / (sqrt(dx^2 + dy^2) + eps)
6         factor <- dir * (carga+masa)
```



(a) Tiempo=01. (b) Tiempo=25. (c) Tiempo=50.

(d) Tiempo=75. (e) Tiempo=100.

Figura 1: Repulsión y atracción de partículas gobernado por la masa y la carga de cada una.

Código 2: Ciclo `for` para el número de pasos a simular y `foreach` para cada partícula.

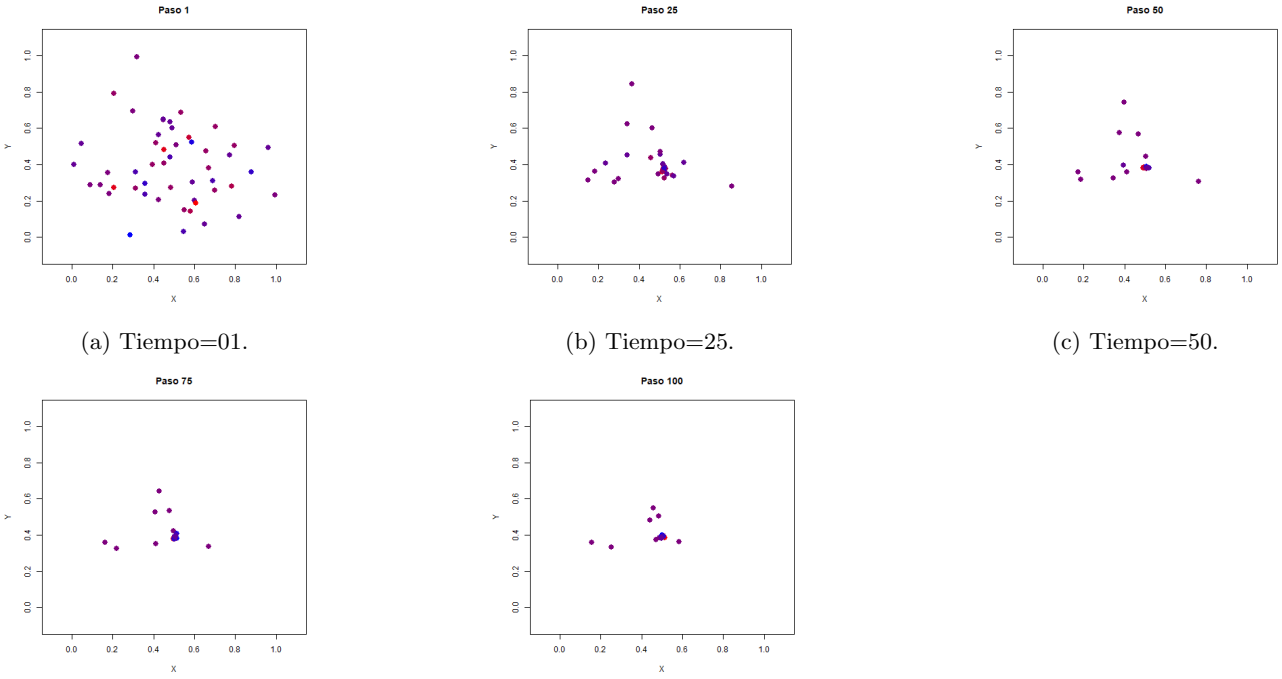
```
for (iter in 1:tmax) {
    f <- foreach(i = 1:n, .combine=c) %dopar% fuerza(i)
    delta <- 0.02 / max(abs(f)) # que nadie desplace una paso muy largo
    p$x <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$x
    + delta * f[c(TRUE, FALSE)][i], 1), 0)
    p$y <- foreach(i = 1:n, .combine=c) %dopar% max(min(p[i,]$y
    + delta * f[c(FALSE, TRUE)][i], 1), 0)
    }
```

Para realizar las gráficas se le atribuyó un color a cada carga desde -5 a 5 y la masa de cada partícula se distingue con el tamaño del punto de cada partícula. El análisis del impacto de la fuerza de la masa en la velocidad de las partículas se realizó considerando la posición inicial y final de cada partícula y se dividió entre el número total de pasos.

4. Resultados

La secuencia gráfica de la repulsión y atracción causada por la masa y la carga se muestra en la figura 1, véase el repositorio de I.Crespo [1] para la animación completa. Se observa la agregación de todas las partículas en una misma zona, y se observa un estado de estacionario a partir del paso 75. En la imagen 2 se presenta la secuencia gráfica de la interacción de partículas solamente gobernadas por la carga de cada una de estas. Véase el repositorio de I.Crespo [1] para la animación completa.

Cada interacción se replicó un total de 15 veces, la media de las velocidades cuando las interacciones están gobernadas solamente por la carga es de 0,0696 *unidades/paso* y cuando la interacción es debido a la masa y a la carga la media de las velocidades resultó 0,0605 *unidades/paso*. La velocidad resultó mayor cuando la interacción era determinada



(a) Tiempo=01. (b) Tiempo=25. (c) Tiempo=50.

(d) Tiempo=75. (e) Tiempo=100.

Figura 2: Repulsión y atracción de partículas gobernado solamente por la carga de cada una.

solamente por la carga ($pvalue = 0{,}019$).

5. Conclusiones

La integración de la masa dentro de las interacciones entre partículas ocasionó una disminución en la velocidad de las partículas cuando la simulación duró 100 pasos y utilizando un factor suavizante `suav`=0.005, sin embargo las partículas se agregan en un mismo punto en menos pasos cuando se integra la componente de la masa en la interacción.

Referencias

[1] I. Crespo. P9: interacciones entre partículas. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P9`.

[2] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

P10

Ismael Crespo

4 de mayo de 2022

1. Introducción

Este trabajo presenta la aplicación del modelo de mochilas para simular mutaciones genéticas y reproducciones para obtener contenidos óptimos de peso en relación con el valor de cada miembro de la población a partir de la generación de un vector característico para cada miembro. E. Schaeffer [2] describe el modelo de mochilas como un problema que consiste en seleccionar un subconjunto de objetos de tal forma que (i) no se exceda la capacidad de la mochila en términos de la suma de los pesos de los objetos incluidos, y que (ii) el valor total de los objetos incluidos sea lo máximo posible.

2. Objetivo

Generar tres instancias con tres distintas reglas:

1.-El peso y el valor de cada objeto se generan independientemente con una distribución uniforme.

2.-El valor de cada objeto se generan independientemente con una distribución exponencial y su peso es inversamente correlacionado con el valor.

3.-El peso de cada objeto se generan independientemente con una distribución normal y su valor es (positivamente) correlacionado con el cuadrado del peso, con un ruido normalmente distribuido de baja magnitud.

Determinar para cada uno de los tres casos si variar la probabilidad de mutación, la cantidad de cruzamientos y el tamaño de la población tienen un efecto estadísticamente significativo en la calidad de resultado, manteniendo el tiempo de ejecución fijo.

3. Programación en R

La obtención del valor óptimo del peso y valor de cada miembro de la población se realiza utilizando las rutinas elaboradas por E.Schaeffer [2], este valor óptimo se fija como un limite en el que valores mayores a este no serán aceptados dentro de la población y los valores mas cercanos a este se consideraran de mejor calidad. Tomando como base el algoritmo genético presentado por E.Schaeffer [2] se agregaron las tres reglas para la generación de los valores y pesos, la generación de los genes se mantuvo con la misma lógica, y se generaron los vectores `PM` (Probabilidad de Mutación) '`POB` (Población) y `REP` (Reproducción) para estudiar el impacto de estas variables en la simulación.

En el código 1 se presenta como se agregó a la rutina las reglas para la generación de los valores y el peso presentadas como `generacion.peso_1`, `generacion.peso_2` y `generacion.peso_3` y de manera similar para la generación de valores.

Cada regla se estudio variando los valores de `PM`, `POB` y `REP` con dos valores diferentes para cada uno, para realizar un análisis estadístico cada experimento se replico tres veces. Para consulta a detalle de los códigos consulte el repositorio de I. Hernández [1]

Código 1: Reglas para la generación de los pesos y valores de los posibles componentes del gen.

```
generador.pesos_1<- function(cuantos, min, max) {
    return(sort(round(normalizar(runif(cuantos)) * (max - min) + min)))
}

generador.valores_1 <- function(cuantos, min, max) {
    return(sort(round(normalizar(runif(cuantos)) * (max - min) + min)))

}

generador.valores_2 <- function(cuantos, min, max) {
    return(sort(round(normalizar(rexp(cuantos)) * (max - min) + min)))
}

generador.pesos_2 <- function(pesos, min, max) {
    n <- length(valores)
    pesos <- double()
    for (i in 1:n) {
        media <- valores[n-i+1]
        desv <- runif(1)
        pesos<- c(pesos, rnorm(1, media, desv))
    }
    pesos <- normalizar(pesos) * (max - min) + min
    return(pesos)
}

generador.pesos_3<- function(cuantos, min, max) {
    return(sort(round(normalizar(rnorm(cuantos)) * (max - min) + min)))
}

generador.valores_3<- function(pesos, min, max) {
    n <- length(pesos)
    valores <- double()
    for (i in 1:n) {
        media <- (pesos[i])^2
        desv <- runif(1)
        valores <- c(valores, rnorm(1, media, desv))
    }
    valores <- normalizar(valores) * (max - min) + min
    return(valores)
}
```

4. Resultados

Para medir que tan cerca se esta del valor óptimo se utilizó la ecuación 1

$$Desviación = \frac{óptimo - mejor}{óptimo} \tag{1}$$

Donde mejor es el valor mas grande encontrado al tiempo final sin que sea mayor al óptimo. La figura 1 presenta los resultados para las tres reglas variando la probabilidad de mutación, la figura 2 presenta los resultados para las tres reglas variando las reproducciones y en la figura 3 se presentan los resultados de las tres reglas variando la población. En la figura 4 se presenta los resultados del algoritmo utilizando la primer regla y utilizando el

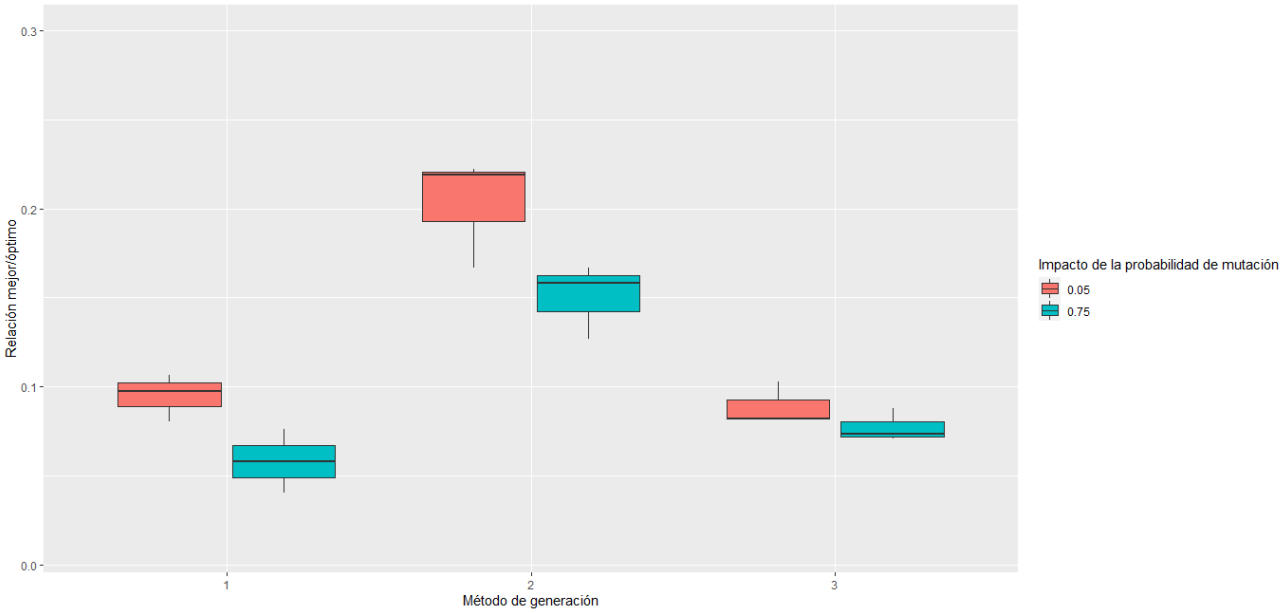


Figura 1: Resultados de la aplicación de un algoritmo genético aplicando las reglas descritas y variando la probabilidad de mutación.

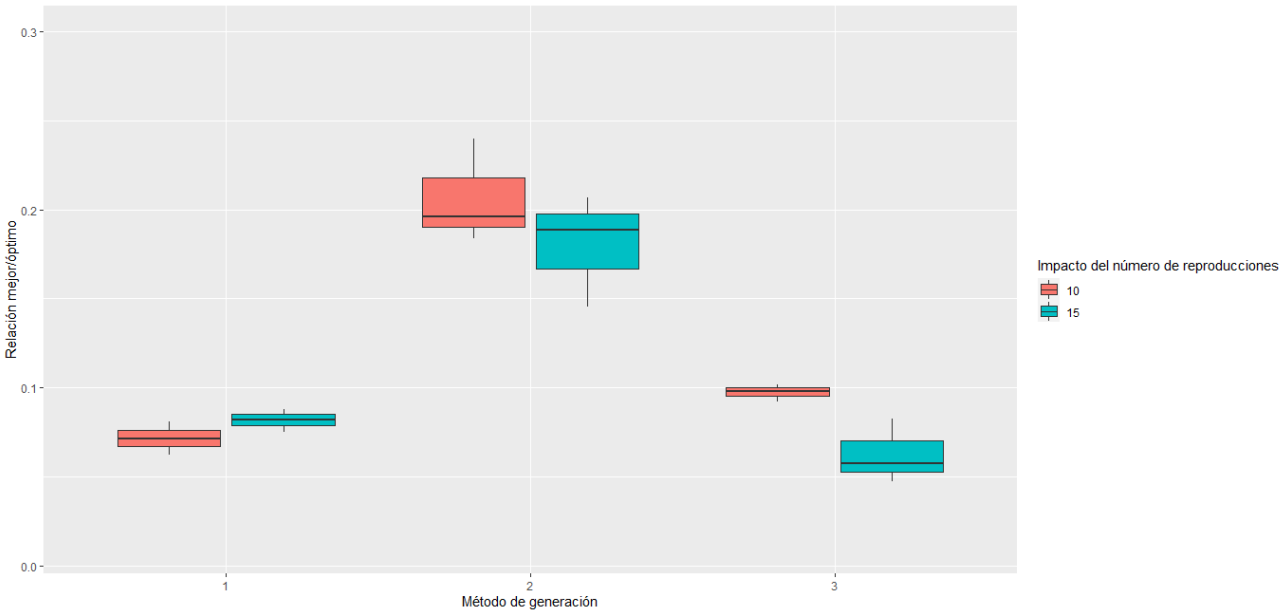


Figura 2: Resultados de la aplicación de un algoritmo genético aplicando las reglas descritas y variando las reproducciones para generar hijos e hijas.

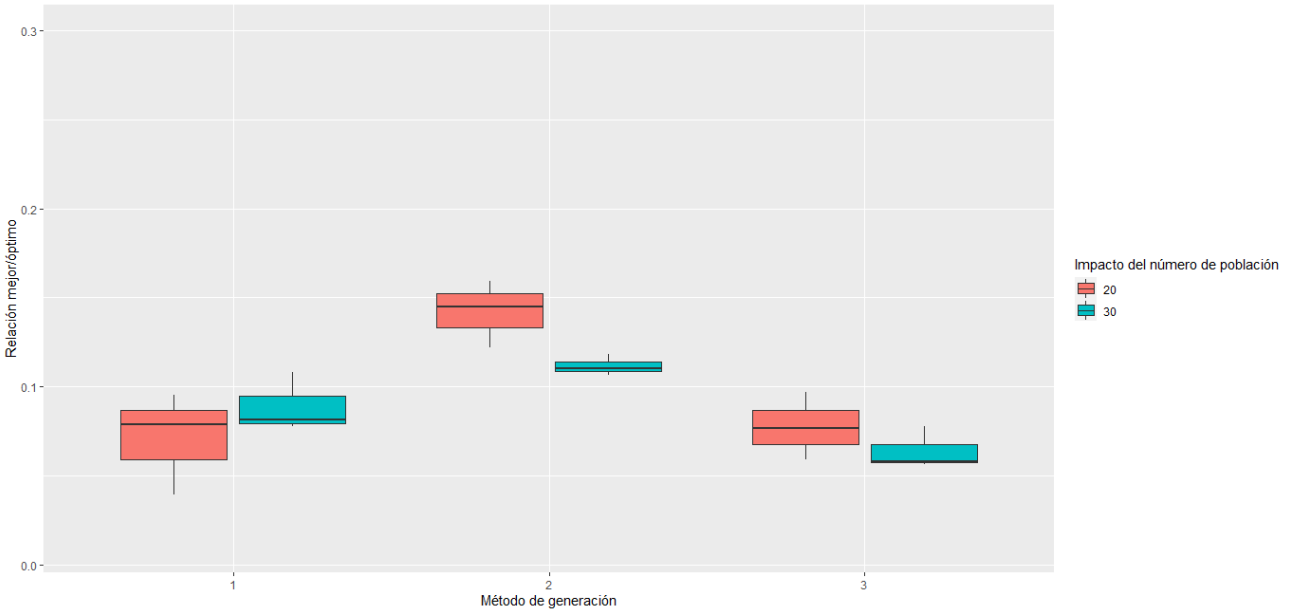


Figura 3: Resultados de la aplicación de un algoritmo genético aplicando las reglas descritas y variando la población permisible.

total de las variables para estudiar su impacto en los resultados. Cada simulación se replicó un total de 3 veces, el cuadro 1 presenta un análisis estadístico de cada experimento, por medio del `wilcox.test` se obtuvo el *p.value* de la relación entre métodos, los resultados no presentan significancia (*p.value* >0.05) cuando se varía `PM`, `POB` y `REP` entre las reglas 1 Y 3, solo tiene significancia utilizando la regla 2. Al comprar las reglas 3 y 1 en todos los casos tampoco se obtuvo significancia (*p.value* >0.05), la regla 2 si presentó significancia al compararse con la regla 1 y 3 (*p.value* <0.05).La figura 4 presenta la combinación de todos los valores de las variables utilizando la regla 1.

Cuadro 1: Estadística de cada simulación.

	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
REGLA 1 PM=0.50	3	0.095	0.013	0.080	0.089	0.102	0.107
REGLA 1 PM=0.75	3	0.058	0.018	0.040	0.049	0.067	0.076
REGLA 2 PM=0.50	3	0.203	0.031	0.167	0.193	0.221	0.222
REGLA 2 PM=0.75	3	0.150	0.021	0.127	0.142	0.162	0.167
REGLA 3 PM=0.50	3	0.089	0.012	0.082	0.082	0.092	0.103
REGLA 3 PM=0.75	3	0.077	0.009	0.071	0.072	0.080	0.088
REGLA 1 POB=20	3	0.071	0.029	0.039	0.059	0.087	0.095
REGLA 1 POB=30	3	0.089	0.017	0.077	0.079	0.095	0.108
REGLA 2 POB=20	3	0.142	0.019	0.122	0.133	0.152	0.159
REGLA 2 POB=30	3	0.112	0.006	0.107	0.108	0.114	0.118
REGLA 3 POB=20	3	0.077	0.019	0.059	0.068	0.087	0.097
REGLA 3 POB=30	3	0.064	0.012	0.056	0.057	0.068	0.078
REGLA 1 REP=10	3	0.071	0.009	0.062	0.067	0.076	0.081
REGLA 1 REP=15	3	0.082	0.006	0.075	0.079	0.085	0.088
REGLA 2 REP=10	3	0.207	0.029	0.184	0.190	0.218	0.240
REGLA 2 REP=15	3	0.180	0.032	0.145	0.167	0.198	0.207
REGLA 3 REP=10	3	0.097	0.005	0.092	0.095	0.100	0.101
REGLA 3 REP=15	3	0.062	0.018	0.047	0.052	0.070	0.083

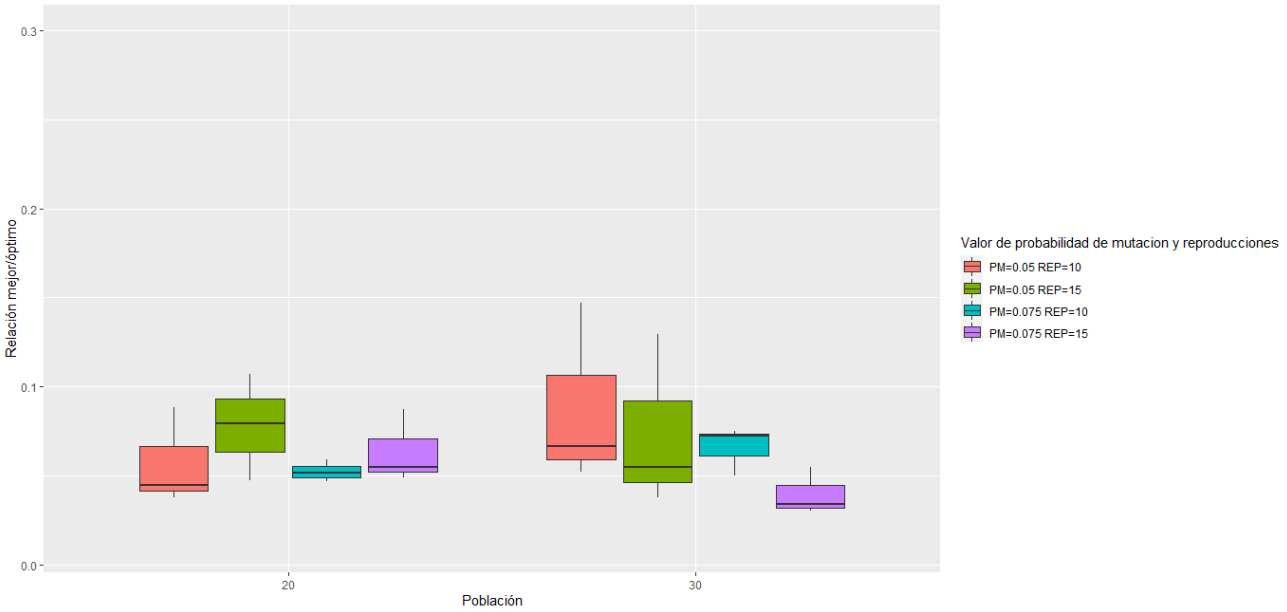


Figura 4: Resultados de la aplicación de un algoritmo genético aplicando la primera regla e intercambiando todas las variables.

El cuadro 2 presenta el análisis estadístico para el algoritmo utilizando la regla 1 y combinando todos los valores de las variables. Los resultados obtenidos al combinar los valores de las variables no representan significancia ($p.value$ >0.05).

Cuadro 2: Estádistica de la aplicación del algortimo con la regla de generación de valores y peso 1 y la combinaicón de todas las variables.

	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
PM=0.05 REP=10 POB=20	3	0.057	0.027	0.038	0.041	0.066	0.088
PM=0.075 REP=10 POB=20	3	0.052	0.006	0.047	0.049	0.055	0.059
PM=0.05 REP=15 POB=20	3	0.078	0.030	0.047	0.063	0.093	0.107
PM=0.075 REP=15 POB=20	3	0.064	0.021	0.049	0.052	0.071	0.087
PM=0.05 REP=10 POB=30	3	0.088	0.051	0.052	0.059	0.107	0.147
PM=0.075 REP=10 POB=30	3	0.066	0.014	0.050	0.061	0.074	0.075
PM=0.05 REP=15 POB=30	3	0.074	0.049	0.038	0.046	0.092	0.129
PM=0.075 REP=15 POB=30	3	0.040	0.013	0.030	0.032	0.044	0.055

5. Conclusiones

La regla 2 para la generación de valores y pesos presenta los peores resultados en relación con el óptimo. El cambio de las reglas tiene significancia, sin embargo el uso de los diferentes valores para cada variable con la misma regla no presenta cambios importantes en el desarrollo del algoritmo.

Referencias

[1] I. Crespo. P10: algoritmo genético. 2022. URL https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P10.

[2] E. Schaeffer. Simulación. 2022. URL https://satuelisa.github.io/simulation.

P11

Ismael Crespo

11 de mayo de 2022

1. Introducción

En optimización multicriterio, a un mismo conjunto de variables ocupa asignarse valores de tal forma que se optimizan dos o más funciones objetivo, que pueden contradecir una a otra — una mejora en una puede corresponder en una empeora en otra. E. Schaeffer [2] describe a la dominancia de Pareto a ,cuando una solución no empeora ninguno de los objetivos y mejora a por lo menos uno. Estas soluciones dominantes se interpretan como un frente que ofrece las mejores opciones entre todas las soluciones. En este trabajo utilizaremos el concepto de soluciones y frente de Pareto para encontrar las mejores soluciones polinómicas.

2. Objetivo

1. Analizar las diferencias observada del porcentaje de soluciones de Pareto como función del número de funciones objetivo para $k = [2, 3, 4, 5]$.
2. Generar un frente de Pareto a partir del original en función de la distancia entre soluciones.

3. Programación en R

La creación de la población de polinomios y la elección de las mejores opciones se realiza utilizando como base las rutinas elaboradas por por E. Schaeffer [2], agregando el vector $k = 1, 2, 3, 4, 5$ que contiene el número de funciones objetivo a simular para cada experimento.

En el código 1 se presenta como se agregó a la rutina el vector k para utilizarse dentro de las funciones generadas por E. Schaeffer [2], y como las dimensiones de la matriz *val* estará en función de el valor de k y del número de soluciones n que se simularon. Cada rutina se replicó 15 ocasiones para un análisis estadístico.

El código 2 presenta la selección de las soluciones en la frontera de Pareto, esta selección pretende eliminar aglomeraciones de soluciones en un mismo lugar, y somete a valoración soluciones cercanas y selecciona las mejores entre estas en función de su diferencia con las mejores soluciones en cada uno de los dos ejes. Para consulta a detalle de los códigos consulte el repositorio de I. Hernández [1].

Código 1: Integración del vector k para la creación de los polinomios utilizando las funciones elaboradas previamente.

```
kvec <- c(2,3,4,5)#cuantas funciones objetivo
replicas=15
obj <- list()
for(k in kvec){
for (rep in 1:replicas){
for (i in 1:k) {
    obj[[i]] <- poli(vc, md, tc)
}
minim <- (runif(k) < 0.5)
sign <- (1 + -2 * minim)
n <- 200 #cuantas soluciones aleatorias
sol <- matrix(runif(vc * n), nrow=n, ncol=vc)
val <- matrix(rep(NA, k * n), nrow=n, ncol=k)
for (i in 1:n) { #evaluamos las soluciones
    for (j in 1:k) { #para todos los objetivos
        val[i, j] <- eval(obj[[j]], sol[i,], tc)
    }
}
```

Código 2: Selección de la frontera de Pareto.

```
for(i in 1:(length(frente)/2)){
for(j in 1:(length(frente)/2)){
if(frente[i,1]>0&&frente[j,1]>0&&abs(frente[i,1]-frente[j,1])<(valmejor1*0.05)
&&frente[i,1]<frente[j,1]&&abs(frente[i,1]-frente[j,1])>0){
frente[j,]=FALSE
if(frente[i,2]>0&&frente[j,2]>0&&abs(frente[i,2]-frente[j,2])<(valmejor2*0.05)
&&frente[i,2]>frente[j,1]&&abs(frente[i,2]-frente[j,2])>0){
frente[j,]=FALSE
}}}}
```

4. Resultados

4.1. Variación del número de funciones objetivo.

La figura 1 presenta los porcentajes de soluciones de Pareto para cada variación de k, se observa que cuando el valor de $k = 2$ el porcentaje de las soluciones de Pareto es menor, y que al aumentar el valor de $k = 5$ es posible que todas las soluciones cuenten con dominanica de Pareto en algunos casos, esto se entiende a que a mayor funciones objetivo es mayor la posibilidad de que una solución en al menos una función objetivo tenga dominancia, en cuando menos una de estas funciones. La tabla 1 presenta datos estadísticos importantes para el número de funciones objetivo, Entre todos los experimentos se obtuvieron valores $p < 0{,}05$ utilizando la función `wilcox.test`.

4.2. Modificación de la frontera de Pareto.

Utilizando el código 2 se modifico la frontera de Pareto en un ejemplo en dos dimensiones para descartar soluciones que se encontrar en una misma zona, seleccionando, cuando hubiera dos muy cerca, la mejore de estas en relación con la mejor en cada eje. La figura 2 presenta la comparación de una frontera de Pareto sin la aplicación de una selección (figura 2a) y después de aplicarse (figura 2b).

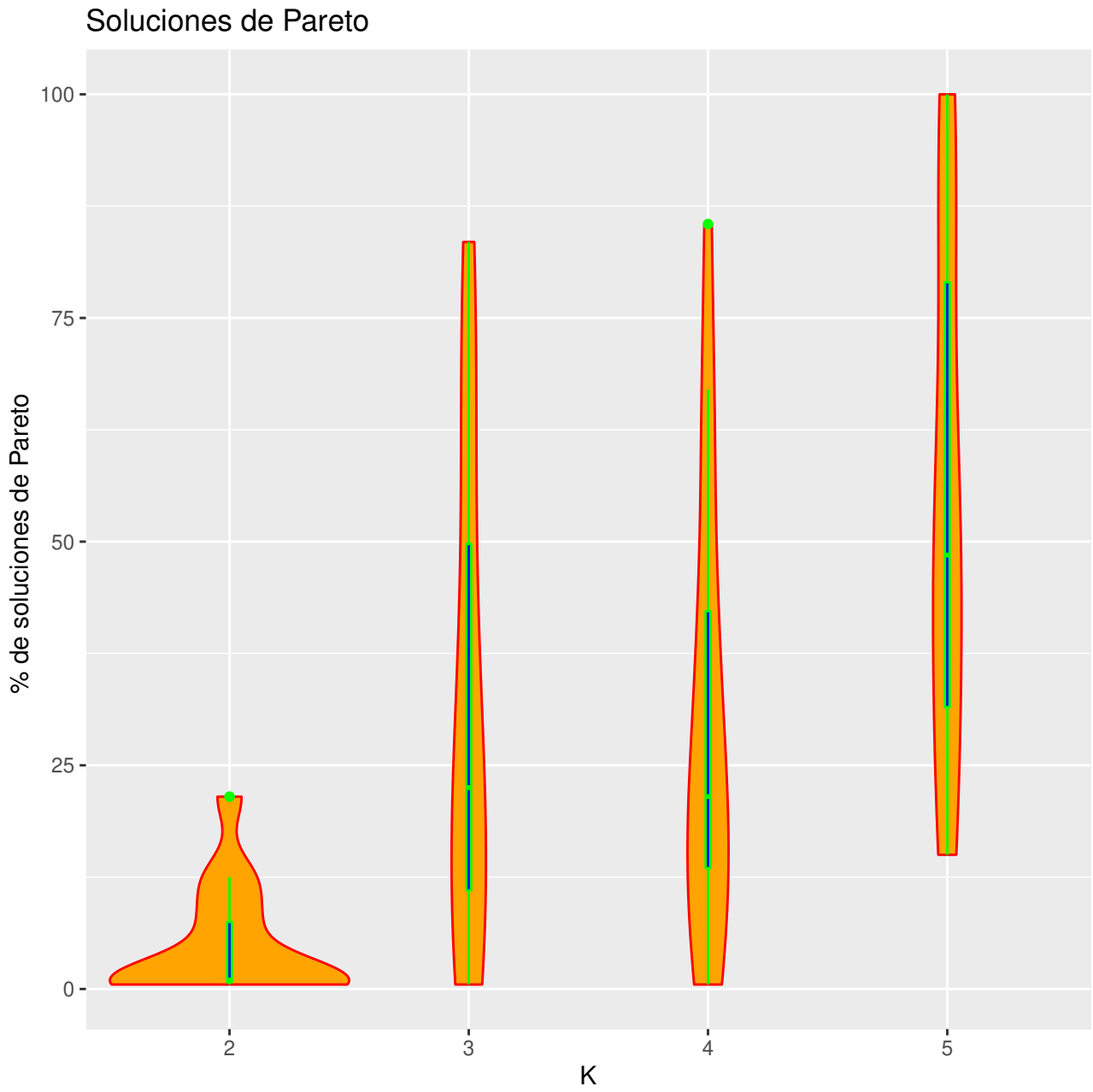
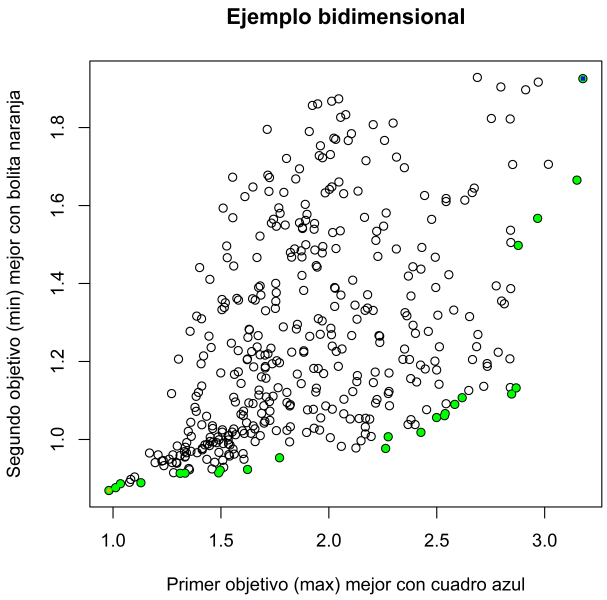


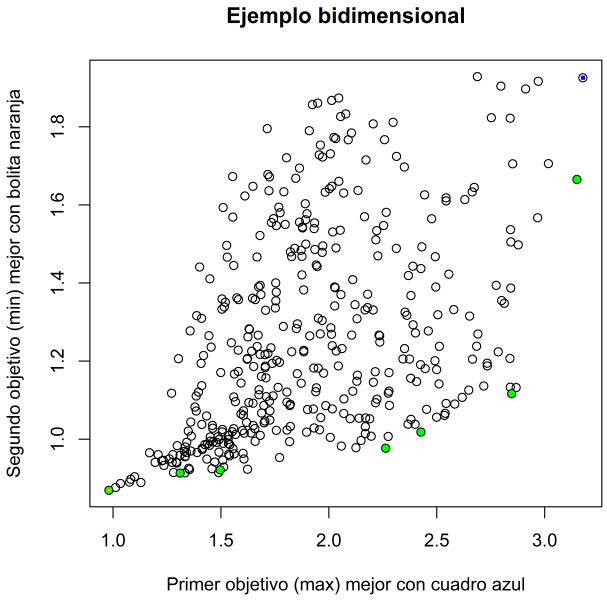
Figura 1: Diagramas caja-bigote y violín del porcentaje de soluciones de Pareto para cada valor de k.

Cuadro 1: Datos estadísticos del porcentaje de las soluciones de Pareto según las funciones objetivos.

k	N	Mean	St. Dev.	Min	Pctl(25)	Pctl(75)	Max
2	15	6.833	6.795	0	2	10.5	22
3	15	21.800	22.896	1.000	2.000	37.000	67.500
4	15	46.433	31.699	2	24	68.5	100
5	15	57.433	26.758	14.000	38.750	80.000	100.000



(a) Frontera de Pareto original.



(b) Frontera de Pareto modificada

Figura 2: Repulsión y atracción de partículas gobernado solamente por la carga de cada una.

5. Conclusiones

El concepto de dominancia y fronteras de Pareto, son muy útiles para la selección de mejores opciones entre materiales, equipos de trabajo, metodologías, por nombrar algunos ejemplos prácticos. Cuando las funciones objetivo son menos estas resultan mas útiles en la selección de una opción, sin embargo al aumentar la longitud del polinomio las soluciones dominantes son muchas más ya que la probabilidad de ser dominantes en alguno de los objetivos aumenta.

Referencias

[1] I. Crespo. P11: frentes de pareto. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P11`.

[2] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

P12

Ismael Crespo

18 de mayo de 2022

1. Introducción

En esta práctica se presenta una demostración básica de aprendizaje a máquina: se busca 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 híperplano (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. Para responder a una salida proporcionada a ello, el perceptrón calcula el producto interno de $x * w$, es decir $\sum_{i=1}^{d} x_i * w_i$, y si esta suma es positiva, la salida del perceptrón es verdad, en otro caso es falso (E. Schaeffer [2]).

2. Objetivo

Estudiar de manera sistemática el desempeño de la red neuronal en términos de su puntaje F (F-score en inglés) para los diez dígitos en función de las tres probabilidades asignadas a la generación de los dígitos (`ngb`), variando a las tres en un experimento factorial adecuado.

3. Programación en R

La creación de los dígitos dentro de una pantalla (figura 1), por medio del encendido probabilístico a negro, gris y blanco (`ngb`), así como la aplicación de una red neuronal se realiza por medio de las rutinas elaboradas por E. Schaeffer [2], utilizando estas rutinas se añade una evaluación de tipo F. Score al desempeño de la red neuronal y se cambian las probabilidades de encendido `ngb` para elaborar un experimento facotrial analizando estas tres instancias como factores y generando 5 replicas para cada experimento. La adición de la evaluacón F. Score (ecuaciones 1,2 y 3) y la varación de los tres factores se presentan en los codigos 1 y 2. Para consulta a detalle de los códigos consulte el repositorio de I. Hernández [1].

$$precision = \frac{TP}{TP + FP} \tag{1}$$

$$rellamado = \frac{TP}{TP + FN} \tag{2}$$

$$F_1 = 2 \times \frac{precision \times rellamado}{precision + rellamado} \tag{3}$$

Donde TP son los verdaderos positivos, FP los falsos positivos y FN los falsos negativos obtenidos del vector `contadores`.

Código 1: Variación de factores ngb.

```
neg=c(0.990,0.999,0.980)
gris=c(0.920,0.930,0.910)
blan=c(0.002,0.001,0.003)
n <- floor(log(k-1, 2)) + 1
neuronas <- matrix(runif(n * dim), nrow=n, ncol=dim) # perceptrones
for(ne in neg){
 for(gr in gris){
  for(bl in blan){
fscore=numeric()
modelos <- read.csv("digits.txt", sep="␣", header=FALSE, stringsAsFactors=F)
modelos[modelos=='n'] <- ne
modelos[modelos=='g'] <- gr
modelos[modelos=='b'] <- bl
for(i in 1:replicas){
........
}}}}
```

Código 2: Calculo de F. Score

```
tp=numeric()
for (i in 1:10){
tp=c(contadores[i,i],tp)
}
tp=sum(tp)
fn=sum(contadores[,11])
tn=0
fp=abs(tp+fn-sum(contadores))
acuraccy=(tp+tn)/(tp+tn+fp+fn)
precision=tp/(tp+fp)
recall=tp/(tp+tn)
fscore=c(fscore,2*(precision*recall)/(precision+recall))
}
fscoremean=c(fscoremean,mean(fscore))
```

4. Resultados

4.1. Variación del número de funciones objetivo.

La tablas 1, 2 y 3, presentan las medianas de los F. Score de cada experimento, separado para la combinación de cada factores, las variaciones de cada probabilidad se muestran en el código 1, el factor 1 ($f1$) se refiere a la probabilidad de negro, el factor 2 ($f2$) a la probabilidad de ser gris y el factor 3 ($f3$) la probabilidad de ser blanco y los subíndices (1,2 y 3) a cada valor que toma cada factor.

Cuadro 1: $f1_1$ media= 0.8952291.

	$f2_1$	$f2_2$	$f2_3$
$f3_1$	0.8930	0.9061	0.8875
$f3_2$	0.8984	0.9015	0.8913
$f3_3$	0.8959	0.8940	0.8895

Cuadro 2: $f1_2$ media=0.9094911.

	$f2_1$	$f2_2$	$f2_3$
$f3_1$	0.9088	0.9126	0.9021
$f3_2$	0.9058	0.9135	0.9066
$f3_3$	0.9147	0.9095	0.9117

Cuadro 3: $f1_3$ media= 0.8846223.

	$f2_1$	$f2_2$	$f2_3$
$f3_1$	0.8826	0.8888	0.8711
$f3_2$	0.8759	0.8914	0.8950
$f3_3$	0.8774	0.8908	0.8886

La tabla 4 presenta la prueba ANOVA, realizada por medio de la función `aov`, la columna $Pr(> F)$ indica la signifcancia para cada factor y entre estos $f1$ resulto tener la mayor signifciancia seguido de $f2$ no se mostró significancia entre los factores.

Cuadro 4: Prueba ANOVA.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
f1	1.0000000	0.0027662	0.0027662	74.6404832	0.0000001
f2	1.0000000	0.0002326	0.0002326	6.2774170	0.0214907
f3	1.0000000	0.0000028	0.0000028	0.0750199	0.7871188
f1:f2	1.0000000	0.0000000	0.0000000	0.0006303	0.9802320
f1:f3	1.0000000	0.0000187	0.0000187	0.5055443	0.4857119
f2:f3	1.0000000	0.0000069	0.0000069	0.1856300	0.6714272
f1:f2:f3	1.0000000	0.0000288	0.0000288	0.7778320	0.3888266
Residuals	19.0000000	0.0007041	0.0000371		

5. Conclusiones

La utilización de una red neuronal para identificar los dígitos generados probabilísticamente a través de pixeles, sgún los valores de `ngb`, funciona con valores de F. score mayores a 0.9 para los mejores casos.

El análisis por medio de experimentos factoriales nos indica las mejores opciones para implementar el experimento de la red neuronal.

Referencias

[1] I. Crespo. P12: red neuronal. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/P12`.

[2] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

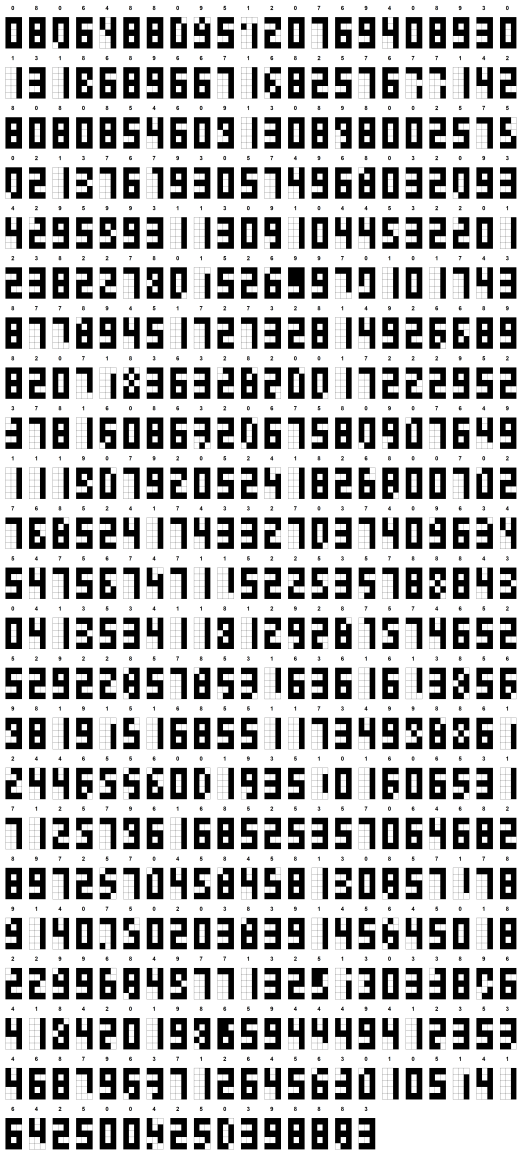


Figura 1: Generación de dígitos con encendido de pixeles.

Separación de moléculas de gas mediante una estructura nanoporosa con propiedades catalizadoras: Simulación y efectos de la porosidad.

Ismael Crespo

1 de junio de 2022

Resumen

En el desarrollo de este trabajo se simuló el flujo de un gas con una composición inicial B_xC_y a través de un medio nanoporoso con propiedades catalizadas para lograr la separación de ambos elementos y tener un flujo final con una fracción de B_x y otra fracción de C_y.Se analiza el impacto del volumen poroso del medio en el desempeño de la separación del gas en las fracciones separadas.

Abstract

In this work, the flow of a gas with an initial composition B_xC_y was simulated through a nanoporous medium with catalyzed properties to achieve the separation of both elements and have a final flow with a fraction of B_x and another fraction of C_y. The impact of the porous volume of the medium on the gas separation performance in the separated fractions is analyzed.

1. Introducción

Disminuir el uso de combustibles fósiles es una de las principales preocupaciones de la agenda mundial para combatir el cambio climático, los reportes anuales 2022 de British Petroleum [6] y la OPEC [4] coinciden en una tendencia a la alta del consumo de hidrocarburos y en algunos países el uso de carbón se mantiene constante desde hace varios años. Esta quema de combustibles generadores de grandes cantidades de CO_2 es uno de las principales actividades que contribuyen a la acumulación de gases de efecto invernadero y al aumento en la temperatura de nuestro planeta (F.Skripnuk et al, 2018 [11]).

Entre las alternativas que han surgido para atacar esta problemática se encuentra la producción de biogas por medio de la combustión anaerobia de desechos orgánicos (K. Pramanik et al, 2018 [7], A. Parsaee et al, 2019 [5]) y la generación de hidrógeno a través de la separación de la molécula de H_2 a partir de CH_4 o H_2O en la producción de hidrógeno destacan el uso de electrolizadores y el reformado de vapor como mecanismos de seperación(M. Kayfeci et al, 2019 [3]).

El reformado de gas natural es una proceso endotérmico en donde por medio de la combustión de metano provocado de la inyección de vapor de H_2O o CO_2 en un reactor. Las principales reacciones que se llevan acabo en estos procesos son las siguientes (R. Julian et al, 2005 [8])

$$CH_4 + H_2O = CO + 3H_2$$
$$CH_4 + CO_2 = 2CO + 2H_2$$

El uso de nanocatalizadores en los procesos de reformado de gas natural ha sido investigado como una método para mejorar la eficiencia de estos procesos y disminuir la temperatura de reacción para la separación del hidrógeno, los

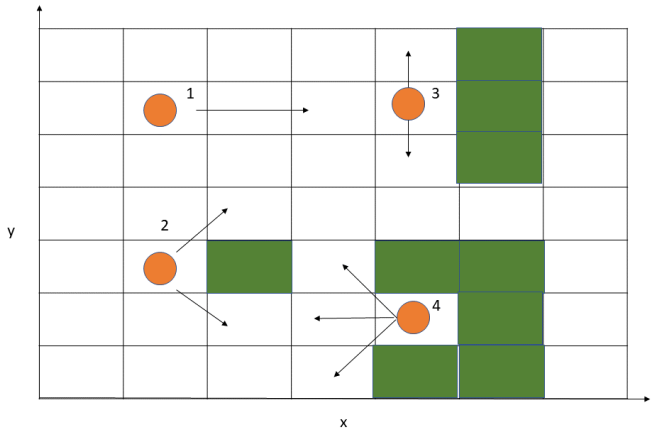


Figura 1: Dinámica de movimiento, el número en cada caso representa el orden donde se intentará mover cada molécula de gas, si no hay espacio vacío en la iteración la molécula permanecerá en la posición en la que se encontraba al inicio de la iteración.

prinicpales nanocatalizadores estudiados son metales como el Co, Ni y Mo debido a su importante actividad reactiva (F. Sharifianjazi et al, 2021 [10], S. Youngdong et al 2021 [12]). Trabajos previos han simulado numericamente el reformado de gas natural en un medio poroso considerando las temperaturas de reacción y la velocidad de flujo en relación con el rendimiento en la producción de hidrógeno (R. Dhamrat et al, 2006 [2]). H, Zeng et al, 2019 [13], estudió el reformado de gas natural en un medio poroso combinado con catalizadores que ayudaban a las reacciones de oxidación involucrados en el proceso.

El objetivo principal de este trabajo es estudiar el flujo de un gas en un medio poroso con propiedades catalizadoras para provocar la separación de este gas en nuevas composiciones o en gases de composición elemental. Se busca analizar el impacto del volumen de los granos en la reacción de separación y el impacto de la porosidad y el poder catalizador del medio en el proceso.

2. Método numérico y experimental para la simulación

La primera parte del modelo genera un medio poroso a partir de la ubicación de granos en la matriz de $n \times n$ y su crecimiento, las moléculas de gas no podrán ocupar estas posiciones y cuando estén en contacto con estos granos en repetidas ocasiones se generará la conversión de esta a otra. Este trabajo utiliza como base la dinámica de movimiento de partículas en 2 dimensiones (x,y) que utiliza en trabajos previos E. Schaeffer [9] donde se considera los espacios vacíos en sus alrededores como posibles posiciones futuras dentro de un espacio matricial de $n \times n$. En este trabajo se considera la posición $(x+1, y)$ como la primer opción de movimiento para provocar un flujo de moléculas de gas orientado en una dirección, si esta posición en el plano esta ocupado por un grano catalizado le dirección a tomar aleatoriamente sería la posición $(x+1, y+1)$ ó $(x+1, y-1)$ como se muestra en la figura 1. Para simular una inyección de flujo constante en cada iteración se generan nuevas moléculas de gas en $x = 1$, $y = 1 : n$, el desempeño de la separación se evalúa con la medición de moléculas separadas al final del reactor (figura 2).

Se simulo cada experimento 5 ocasiones iterando 250 veces la inyección constante de 10 moléculas de gas y el movimiento antes descrito para cada molécula. Cuando una molécula estuvo en contacto con material catalizado durante 50 ocasiones en alguno de sus ejes esta molécula cambiará de composición. Consúltese la referencia [1] para ver el código del modelo.

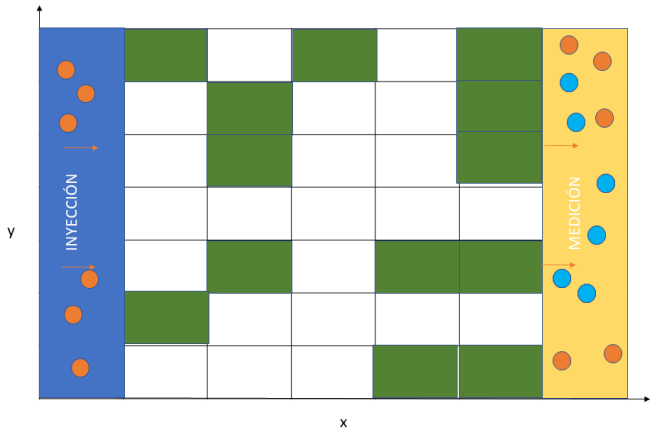
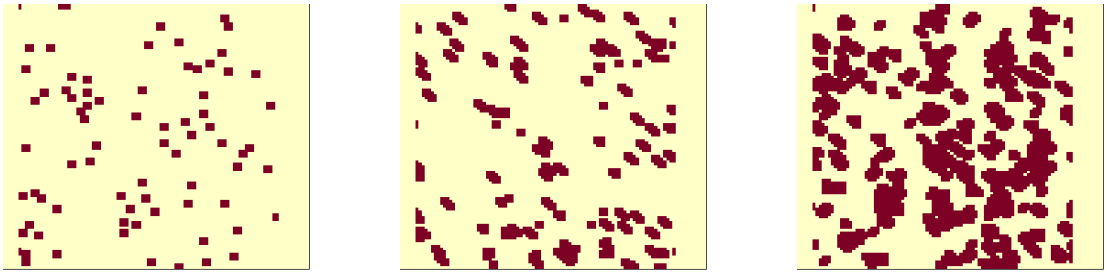


Figura 2: Diseño experimental de inyección constante y medición.



(a) Porosidad=1. (b) Porosidad=2. (c) Porosidad=3.

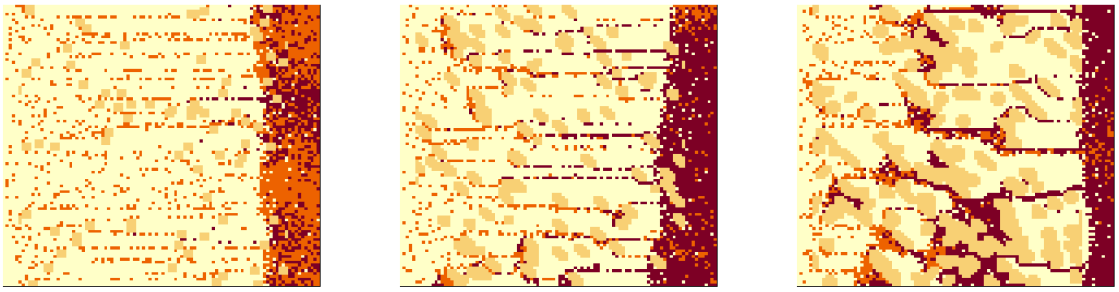
Figura 3: Medios porosos para flujo de gas, las zonas huecas al inicio y al final son las zonas de inyección y medición.

2.1. Medio poroso

Se generaron 3 diferentes medios porosos con fracciones porosas adimensionales 1,2 y 3, que indican el tiempo de crecimiento de los granos iniciales, en la figura 3 se presentan los tres medios porosos generados. La composición de los granos del medio es tal que fomenta la separación del gas, por lo que al estar en contacto el gas con la superficie de cada grano este comenzara a separarse.

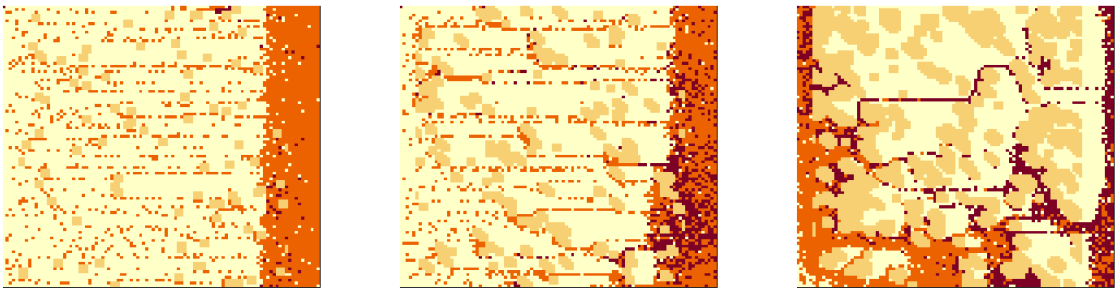
3. Resultados

La figura 4 presenta las condiciones del reactor en T=250 cuando el tiempo en contacto necesario para la separación del gas es $r = 50$, en la figura 5 se presenta el termino de la simulación para la consúltese el material adicional disponible [1] para ver las secuencias gráficas de cada simulación. El análisis del desempeño de cada combinación de porosidad y capacidad catalítica se evalúa con la medición de los gases en la zona destinada a esta al final del reactor poroso, tomando la medición del volumen de B y A respectivamente y la relación del total recuperado, el volumen total inyectado al $T = 250$ es de 2510 unidades de gas A_xB_y. En las figura 6 se observa el volumen recuperado de A_xB_y y B_y al final de cada experimento. La figura 7 presenta el total de gas recuperado en la zona



(a) t=250, Porosidad=1. (b) T=250, Porosidad=2. (c) T=250, Porosidad=3.

Figura 4: Final de la simulación para cada medio poroso con un valor de catalizador $r = 50$.



(a) t=250, Porosidad=1. (b) T=250, Porosidad=2. (c) T=250, Porosidad=3.

Figura 5: Final de la simulación para cada medio poroso con un valor de catalizarod $r = 100$.

de medición(fig 7a) y la fracción de B_y (fig 7b) sobre el volumen total de gas recuperado:

$$FB_y = \frac{VB_y}{VB_y + VA_xB_y} \tag{1}$$

Evaluando la significancia de la modificación del poder catalizador por medio de un `wilcox.test` se encontró que para porosidades 1 y 2 si influye el volumen de B_y recuperado (P. value<0.05) pero para la porosidad de 3 no mostró significancia (P. value>0.05). El valor de la porosidad si mostró significancia en todos los casos (P. value<0.05) en el volumen de B_y recuperado, los mismos resultados se obtuvieron en cuanto al volumen de A_x recuperado y el impacto de la modificación de la porosidad y el catalizador. En cuanto al volumen total de gas recuperado se observa que este no se ve afectado por la porosidad 1 y 2 (P. value>0.05), sin embargo para la porosidad 3 se observo significancia con relación a la porosidad 1 y 2, pero se observó una desviación estándar grande (159.20) debido a la geometría del medio poroso y la afectación a la permeabilidad.Los cuadros 1, 2, 3 y 4 presentan los datos estadísticos para cada análisis del desempeño del proceso variando la porosidad y el catalizador.

Cuadro 1: Volumen de A_xB_y recuperado según la porosidad (P) y el catalizador (r).

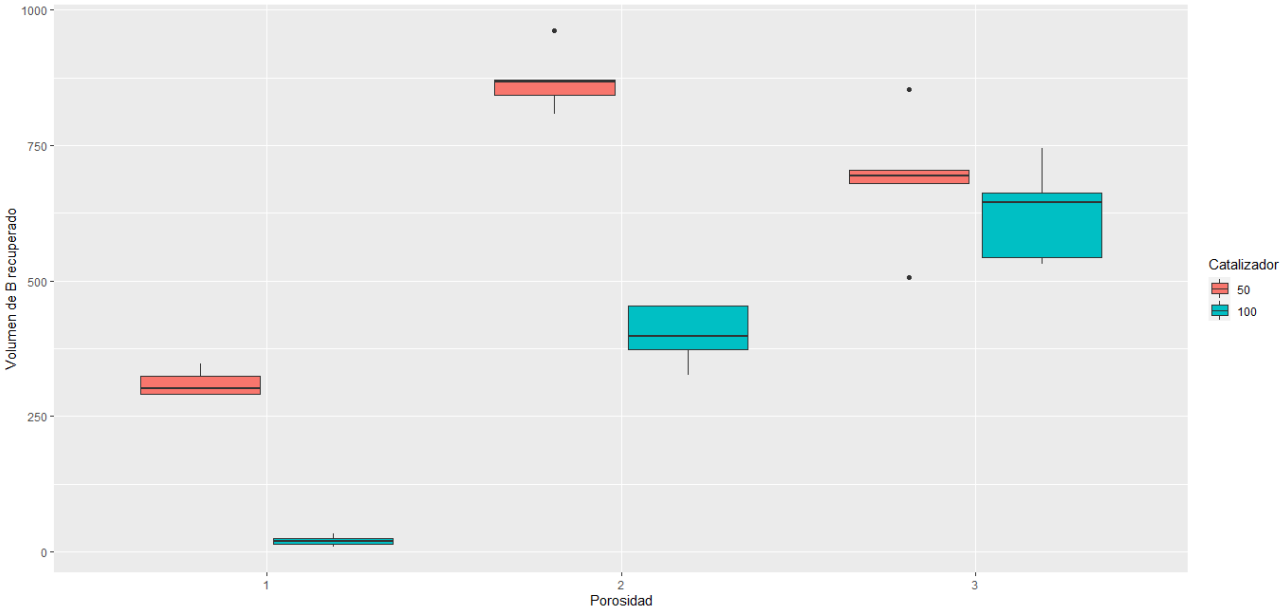
Statistic	N	Mean	St. Dev.	Min	Max
P=1 r=50	5	669	27	632	691
P=1 r=100	5	958	11	943	972
P=2 r=50	5	108	60	12	171
P=2 r=100	5	577	58	519	651
P=3 r=50	5	27	35	0	68
P=3 r=100	5	120	100	39	274

Cuadro 2: Volumen de B_y recuperado según la porosidad (P) y el catalizador (r).

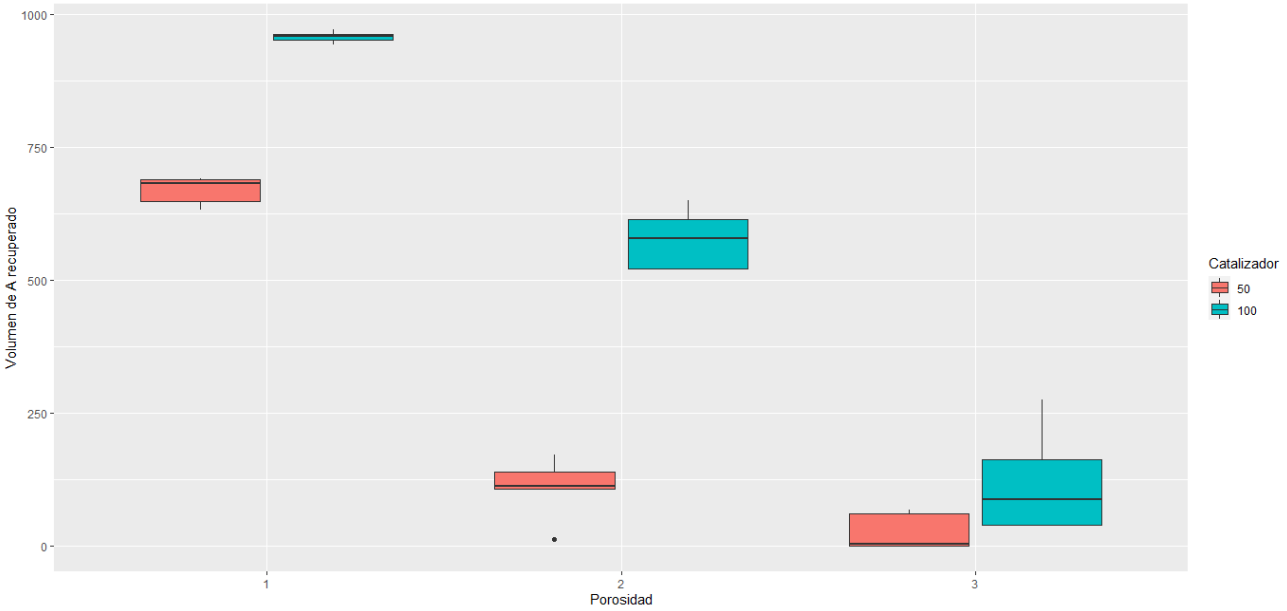
Statistic	N	Mean	St. Dev.	Min	Max
P=1 r=50	5	311	24	290	347
P=1 r=100	5	20	9	9	33
P=2 r=50	5	870	57	807	962
P=2 r=100	5	400	54	326	453
P=3 r=50	5	687	123	507	853
P=3 r=100	5	625	89	530	745

Cuadro 3: Volumen total recuperado de ambos gases según la porosidad (P) y el catalizador (r).

Statistic	N	Mean	St. Dev.	Min	Max
P=1 r=50	5	979	5	972	985
P=1 r=100	5	978	2	976	981
P=2 r=50	5	978	3	974	981
P=2 r=100	5	977	6	972	987
P=3 r=50	5	714	147	507	921
P=3 r=100	5	745	159	582	918

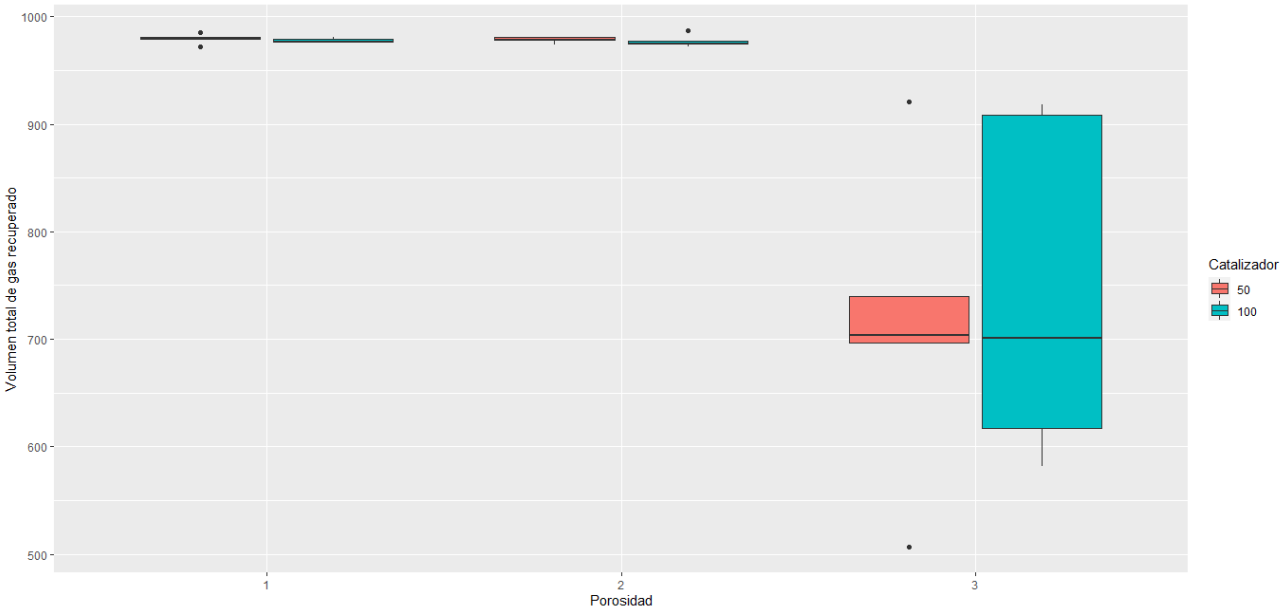


(a) Volumen de B_y.

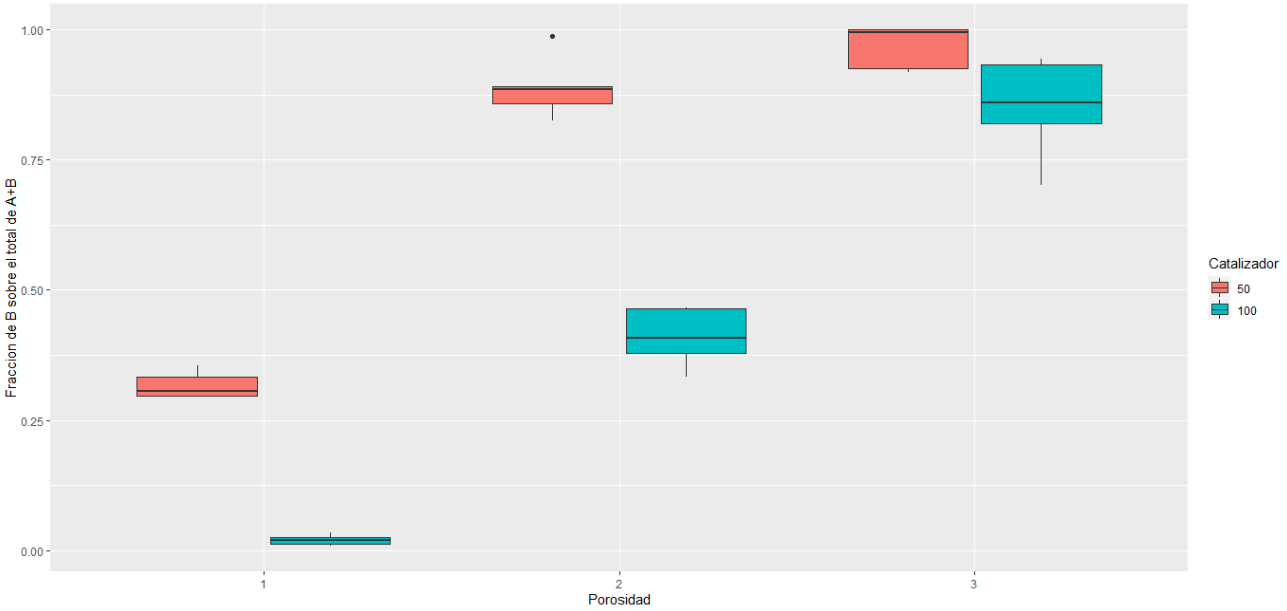


(b) Volumen de A_xB_y

Figura 6: Volúmenes recuperados de la composición original de gas y la composición separada.



(a) Total de gas recuperado A_xB_y.



(b) Fracción de B_y

Figura 7: Volúmenes recuperados de la composición original de gas y la composición separada.

Cuadro 4: Fracción de B_y según la porosidad (P) y el catalizador (r)

Statistic	N	Mean	St. Dev.	Min	Max
P=1 r=50	5	0.32	0.03	0.30	0.35
P=1 r=100	5	0.02	0.01	0.01	0.03
P=2 r=50	5	0.89	0.06	0.83	0.99
P=2 r=100	5	0.41	0.06	0.33	0.47
P=3 r=50	5	0.97	0.04	1	1
P=3 r=100	5	0.85	0.10	0.70	0.94

4. Conclusiones

1. El poder del catalizador tiene significancia mejorando la separación y recuperación del gas B_y para las porosidades 1 y 2.

2. El volumen mas grande de B_y recuperado se logró con la porosidad 1 y el catalizador 50.

3. Es claro observar que a el catalizador 50 y el medio poroso 3 muestran el mejor desempeño en relación a la fracción de B_y, sin embargo el volumen total recuperado no es el mejor, esto se debe a que la estructura porosa genera tortuosidad y disminuye la permeabilidad del medio atrapando grandes cantidades de gas entre sus poros.

4. Para la porosidad 3 el poder del catalizador disminuye la significancia debido a que se garantiza un prolongado contacto entre el gas y la estructura porosa a lo largo del flujo.

5. Se propone, para trabajos futuros, realizar modificaciones de la permeabilidad, como fracturas a lo largo del medio para mejorar el volumen de B_y recuperado.

Referencias

[1] I. Crespo. Separación de moléculas de gas mediante una estrucutra nanoporosa con propiedades catalizadoras: Simulación y efectos de la porosidad. 2022. URL `https://github.com/IsmaelHC/Simulacion-NANO-2022/tree/main/PF`.

[2] Raviraj S. Dhamrat and Janet L. Ellzey. Numerical and experimental study of the conversion of methane to hydrogen in a porous media reactor. *Combustion and Flame*, 144(4):698–709, 2006. ISSN 0010-2180. doi: https://doi.org/10.1016/j.combustflame.2005.08.038.

[3] Muhammet Kayfeci, Ali Keçebaş, and Mutlucan Bayat. Chapter 3 - hydrogen production. In Francesco Calise, Massimo Dentice D'Accadia, Massimo Santarelli, Andrea Lanzini, and Domenico Ferrero, editors, *Solar Hydrogen Production*, pages 45–83. Academic Press, 2019. ISBN 978-0-12-814853-2. doi: https://doi.org/10.1016/B978-0-12-814853-2.00003-5.

[4] OPEC. Annual statistical bulletin 2021. page 211, 2021. URL `https://asb.opec.org/`.

[5] Mostafa Parsaee, Mostafa Kiani Deh Kiani, and Keikhosro Karimi. A review of biogas production from sugarcane vinasse. *Biomass and bioenergy*, 122:117–125, 2019. doi: https://doi.org/10.1016/j.fuel.2019.03.065.

[6] British Petroleum. Energy outlook 2022 editions. page 189, 2021. URL `https://www.bp.com/en/global/corporate/energy-economics/energy-outlook.html`.

[7] Sagor Kumar Pramanik, Fatihah Binti Suja, Shahrom Md Zain, and Biplob Kumar Pramanik. The anaerobic digestion process of biogas production from food waste: Prospects and constraints. *Bioresource Technology Reports*, 8:100310, 2019. ISSN 2589-014X.

[8] Julian R.H. Ross. Natural gas reforming and co2 mitigation. *Catalysis Today*, 100(1):151–158, 2005. ISSN 0920-5861. doi: https://doi.org/10.1016/j.cattod.2005.03.044. 100th Anniversary Issue.

[9] E. Schaeffer. Simulación. 2022. URL `https://satuelisa.github.io/simulation`.

[10] Fariborz Sharifianjazi, Amirhossein Esmaeilkhanian, Leila Bazli, Sara Eskandarinezhad, Samad Khaksar, Parisa Shafiee, Mohammad Yusuf, Bawadi Abdullah, Peyman Salahshour, and Farnaz Sadeghi. A review on recent advances in dry reforming of methane over ni- and co-based nanocatalysts. *International Journal of Hydrogen Energy*, 2021. ISSN 0360-3199. doi: https://doi.org/10.1016/j.ijhydene.2021.11.172.

[11] D F Skripnuk and E A Samylovskaya. Human activity and the global temperature of the planet. *IOP Conference Series: Earth and Environmental Science*, 180:012021, aug 2018. doi: https://doi.org/10.1088/1755-1315/180/1/012021.

[12] Youngdong Song, Ercan Ozdemir, Sreerangappa Ramesh, Aldiar Adishev, Saravanan Subramanian, Aadesh Harale, Mohammed Albuali, Bandar Abdullah Fadhel, Aqil Jamal, Dohyun Moon, Sun Hee Choi, and Cafer T. Yavuz. Dry reforming of methane by stable ni mo nanocatalysts on single-crystalline mgo. *Science*, 367(6479): 777–781, 2020. doi: http://doi:10.1126/science.aav2412.

[13] Hongyu Zeng, Yuqing Wang, Siqi Gong, Yixiang Shi, and Ningsheng Cai. Catalytically enhanced methane-rich combustion by porous media reactor. *Fuel*, 248:65–75, 2019. ISSN 0016-2361. doi: https://doi.org/10.1016/j.fuel.2019.03.065.