Potafolio de Evidencias

Oscar Quiñonez

16 de diciembre de 2020

1. Datos

Clase: Simulación computacional de nanomateriales
Semestre: Agosto 2020 - Enero 2021
Profesora: Satu Elisa Schaeffer
Matrícula: 2033891

2. Instrucciones

En este documento se adjuntan las doce tareas llevadas a cabo durante el semestre.

Movimiento Browniano

Oscar Quiñonez

23 de septiembre de 2020

1. Objetivo

A través de una simulación se intenta representar la manera en que el movimiento browniano afecta en el regreso de una partícula a su punto origen.

2. Metodología

Para llevar a cabo la simulación del movimiento browniano se utilizó el programa R 4.0.2 para representar una caminata desde el origen, fueron simuladas hasta 8 dimensiones con los incrementos exponenciales entre 5 y 10, a cada incremento se le hicieron 50 repeticiones para conocer que tan probable podría ser el retorno al origen de la partícula. [1] Fueron utilizadas las instrucciones de la tarea 1, [2] además del apoyo en el repositorio de Tellez, C.

3. Resultados y Discusión

Al realizar la simulación en R 4.0.2 se generó una serie de datos que explican el regreso de la partícula al origen en cada una de las 8 dimensiones. A continuación se muestra la gráfica en la que se pueden ver las 8 dimensiones y como cada una de ellas esta representada por una columna. Ver figura 1.

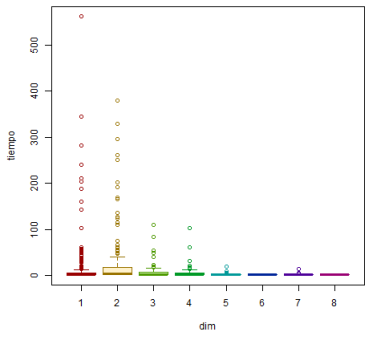


Figura 1: Tiempo para regresar al origen en cada una de las 8 dimensiones.

4. Conclusión

La simulación de modelos matemáticos usando R 4.0.2 nos ayudó a entender como una partícula regresa a su punto de origen en 8 dimensiones, se puede observar que mientras mas dimensiones recorra más rápido regresa, pero al mismo tiempo es más improbable que esto suceda.

Referencias

[1] E. Schaeffer. Práctica 1: Movimiento browniano, September 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p1.html`.

[2] C Tellez. Práctica 1, September 2020. URL `https://github.com/claratepa/Simulacion/tree/master/Practica1`.

Autómata Celular

Oscar Quiñonez

30 de septiembre de 2020

1. Objetivo

Mediante la simulación de una matriz de 20 X 20 celdas, se intenta observar el comportamiento de esta a través del "juego de la vida" durante 50 iteraciones.

2. Metodología

En esta simulación fue necesario el uso del programa Python 3, donde se representó una matriz boolena, es decir que solamente contiene los números 1 y 0. Fueron utilizadas las instrucciones [1] de la tarea 2, donde se especifica que una celda se mantien viva únicamente si está rodeada de otras 3 celdas vivas, además del apoyo en el repositorio [2] como base de l código .

3. Resultados y Discusión

Al realizar la simulación en Python 3 se obtuvó una matriz en la que cada punto representa una celda viva y cada una muestra la cantidad de iteraciones que "sobrevivió" . Se puede apreciar que las últimas celdas vivas estan en la esquina de la matriz resaltado en color negroy entre ellas mismas cumplen con la condición para vivir. A continuación se muestra la matriz. Ver figura 1 .

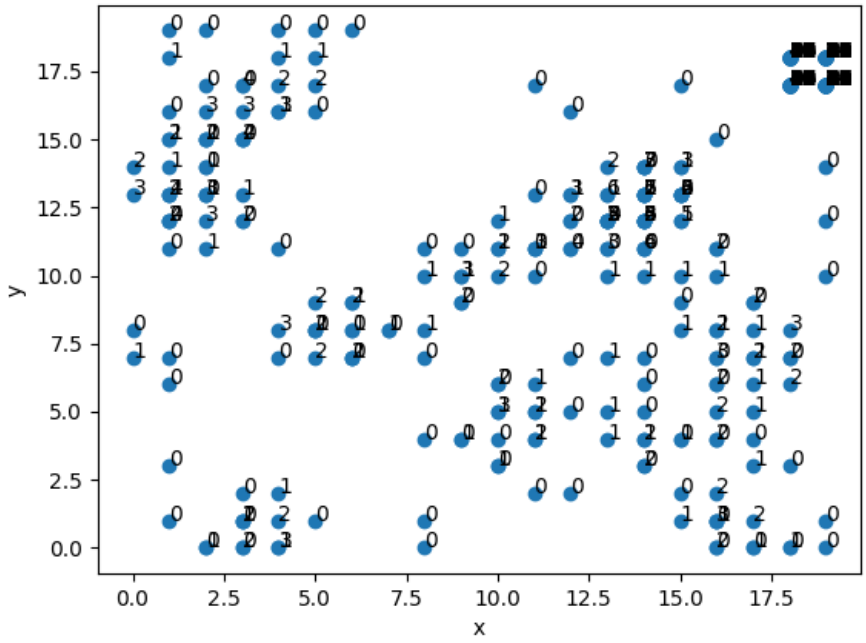


Figura 1: Se muestra un mapeo de la matriz.

4. Conclusión

La simulación del "juego de la vida" nos muestra como una matriz se va desarrollando a lo largo de 50 iteraciones en las que existe un declive en el número de celdas que permanecen activas, ya que, como se mencionaba al principio del experimento, la condición era estar rodeada de otras 3 celdas activas, por lo que, cada iteración reduciá las probabilidades de reunirlas y por lo tanto iban eliminándose entre ellas. En el repositorio[3] se puede encontrar el archivo en formato gif del comportamiento de la matriz.

Referencias

[1] E. Schaeffer. Práctica 2: Autómata celular, September 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p2.html`.

[2] E. Schaeffer. Práctica 2, September 2020. URL `https://github.com/satuelisa/Simulation/blob/master/CellularAutomata/gameOfLife.py`.

[3] O. Quiñonez. tareados, September 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareados`.

Teoría de Colas

Oscar Quiñonez

8 de octubre de 2020

1. Objetivo

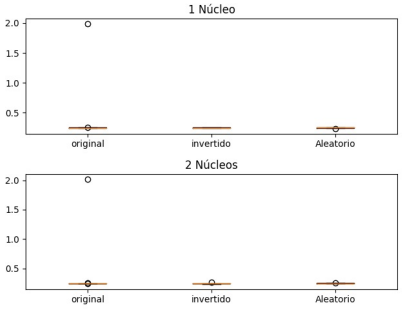
En la presente simulación, se intenta reproducir la llamada"Teoría de colas" utilizando un archivo que contiene números primos previamente generados.

2. Metodología

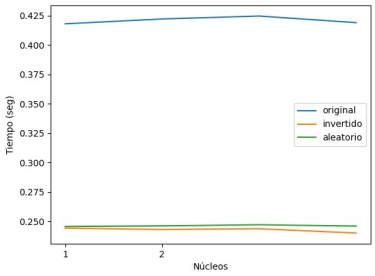
Esta simulación fue realizada usando como herrameinta el programa Python 3, donde se examinó el archivo que contenía miles de números primos y que la ser ejecutado, se varío el uso de los núcleos que tiene el procesador instalado en el ordenador. Para realizar esta simulación fue necesario el uso de las instrucciones [3] de la tarea 3, donde se menciona que los números primos deben de tener por lo menos 8 dígitos, además del apoyo en el repositorio Schaeffer [2] como base de l código utilizado.

3. Resultados y Discusión

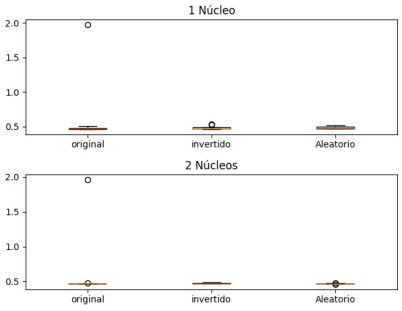
Al realizar la simulación en Python 3 se obtuvieron una serie de datos para 1000, 2000, 3000 y hasta 4000 números del archivo auxiliar, a partir de los cuales se pudieron generar dos tipos de gráficas, una con respecto al uso de los núcleos contra la cantidad de datos y otra de núcleos contra tiempo. Se pueden observar estas gráficas a continuación.



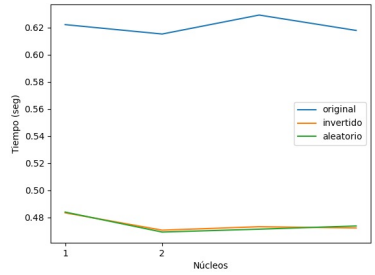
(a) Gráfica para mil números usando dos procesadores.



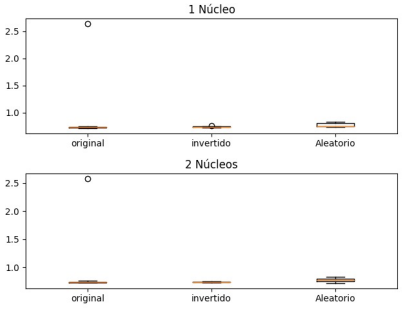
(b) Gráfica de núcleos contra tiempo.



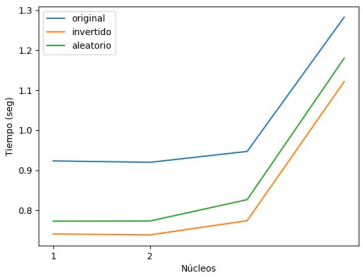
(c) Gráfica para dos mil números usando dos procesadores



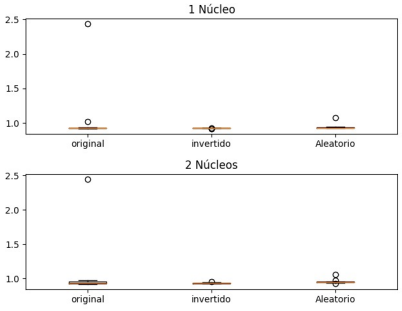
(d) Gráfica de núcleos contra tiempo.



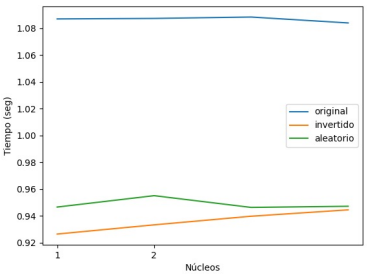
(e) Gráfica para tres mil números usando dos procesadores



(f) Gráfica de núcleos contra tiempo



(g) Gráfica para cuatro mil números usando dos procesadores



(h) Gráfica de núcleos contra tiempo

Figura 1: Magnitudes del vector

4. Conclusión

La simulación de la "Teoría de colas" usando números primos usando 8 dígitos nos ayudó a variar el uso de los núcleos en el procesador y con ello, la diferencia del tiempo que se pude ver en las gráficas. En el repositorio[1] se puede encontrar las gráficas y las características del procesador.

Referencias

[1] O. Quiñonez. tareados, September 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareatres`.

[2] E. Schaeffer. Práctica 3, September 2020. URL `https://github.com/satuelisa/Simulation/blob/master/QueuingTheory/ordering.py`.

[3] E. Schaeffer. Práctica 3: Teoría de colas, September 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p3.html`.

Diagramas de Voronoi

Oscar Quiñonez

14 de octubre de 2020

1. Objetivo

En el presente trabajo se realiza la representación de una celda en la cual se siembran semillas, a través de la simulación usando Python, se intenta ver el comportamiento de las grietas formadas en esta celda, así también, determinar la mayor distancia euclidiana en una grieta.

2. Metodología

Dadas las instrucciones [1] en la tarea 4, y el uso del código [2]como referencia, se nos indica el uso de los diagamas de Voronoi, mediante la cual se generó un código que simulara la celda con variaciones en la cantidad de semillas, las cuales fueron, 40, 60, y 80 semillas.

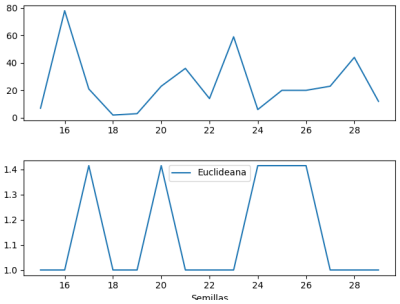
3. Resultados y Discusión

Al usar el código generado, y variando la cantidad de semillas que se simularon en el diagrama de Voronoi con las cantidades dadas anteriormente, nos da como resultado una serie de imágenes de que representan la celda y cada color representa un espacio que ocupa la semilla, entre ellas, se forma una grieta que se puede apreciar en color negro y a medida que va creciendo, se va acercando al borde de la celda (figuras 1, 2 y 3). En la figura 4 se pude apreciar que las celdas en las que se llega al borde son: 17, 21, 24, 25 y 26. Y que las distancias máximas de recorrido fueron aproximadamente 80 celdas. A continuación se muestran las figuras 5 y 6 en las que las distancias máximas fueron mayor a 100 y aproximado a 70 respectivamente.

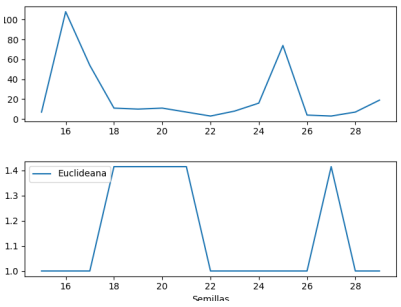
(a) Figura 1: celda con 40 semillas.

(b) Figura 2: celda con 60 semillas.

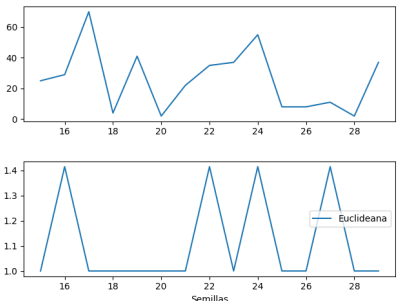
(c) Figura 3: celda con 80 semillas.



(d) Figura 4: celda con 40 semillas.



(e) Figura 5: celda con 60 semillas.



(f) Figura 6: celda con 80 semillas.

4. Conclusión

En las figura 7 y 8 vemos de mejor manera la diferencia entre el comienzo de una grieta y el final de la misma, lo que nos muestra claramente la distancia que recorrió. Mediante el uso de los diagramas de Voronoi, se pudo determinar visualmente el comportamiento de las semillas dentro de una celda en la que va desarrollándose una grieta, por esta razón es muy útil este tipo de diagramas para la representación de un plano euclídeo. Todas las imágenes se pueden ver [3] en las carpetas destinadas para las distintas cantidades de semillas.

(a) Figura 7: inicio de la grieta.

(b) Figura 8: fin de la grieta.

Referencias

[1] E. Schaeffer. Práctica 4: diagramas de voronoi, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p4.html`.

[2] E. Schaffer. Voronoidiagrams, 2020. URL `https://github.com/satuelisa/Simulation/tree/master/VoronoiDiagrams`.

[3] O. Quiñonez. tareados, September 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareacuatro`.

Método Monte-carlo

Oscar Quiñonez

21 de octubre de 2020

1. Objetivo

Como parte del proyecto 5 se utiliza el método Monte-carlo en el que se busca el tamaño de la muestra que se requiere para conocer el lugar que ocupan hasta siete decimales generados por una integral que previamente fue ejecutada en el software Wolfram Alpha.

2. Metodología

Para esta simulación se requirió el uso de R en su versión 4.0.2, además del uso de un código [1] como base para generar distintas variaciones de este y obtener así una rutina de 100 repeticiones para valores de 100, 1000, 10000, 100000 y 1000000 como tamaño de muestra. Mediante el uso del método Monte-carlo se calculó la diferencia entre el valor real y el estimado para conocer así el margen de error que existe entre ellos.

3. Resultados y Discusión

Al obtener los resultados de la rutina [2] se puede deducir que mientras mayor sea el valor en el tamaño de la muestra mas nos acercamos al valor real de la integral que tomamos como referencia en Wolfram-Alpha, es decir la precisión esta absolutamente relacionada con la cantidad de dígitos en la muestra pues se pueden obtener hasta 4 decimales para 1000000, lo que supone una mayor aproximación al valor real. En la figura 1 que se muestra a continuación, se presenta un diagrama de caja-bigote en la que la precisión se ve representada como el área que se encuentra por debajo de cada una de las líneas de colores, como indica la leyenda ubicada en el extremo superior derecho. Esta nos indica que efectivamente mientras mayor sea el valor de la muestra, el margen de error se verá reducido y por eso las líneas roja y verde están muy cercanas entre ellas.

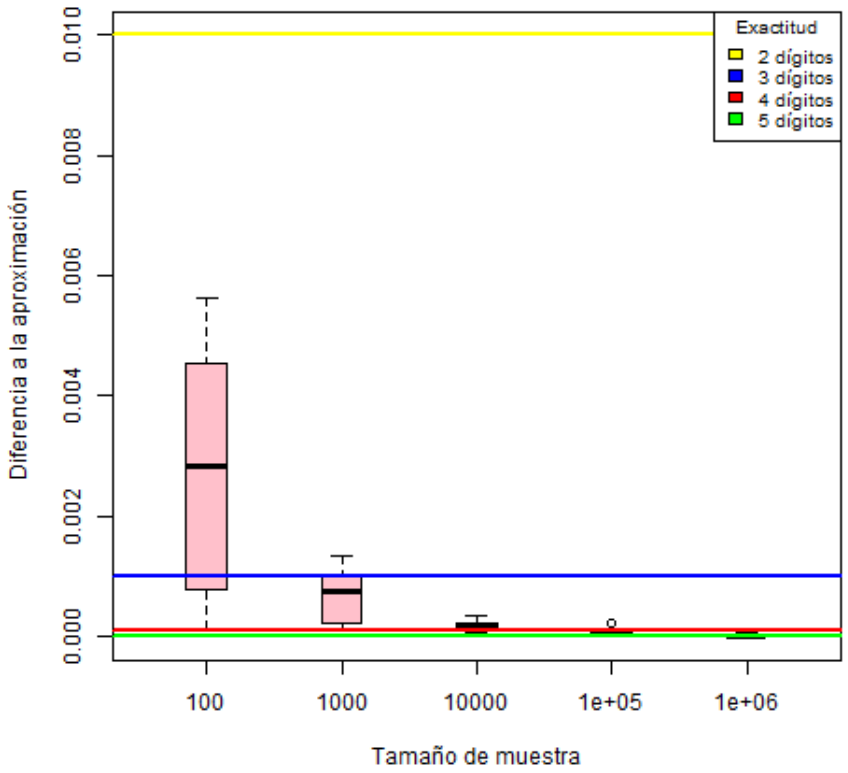


Figura 1: Diagrama de caja-bigote en la que se observan las aproximaciones al valor real.

4. Conclusión

Existe una relación directa entre la aproximación al valor real de la integral y el valor de la muestra con la que se calculó, pues mientras mayor sea este último, más decimales nos generará y mayor será la cercanía de los valores, como se vio representada en las líneas de la figura1. El método Monte-carlo es muy útil en este tipo de experimentos pues nos permite usar expresiones matemáticas complejas (en este caso una integral) para aproximarnos a valores exactos que de otra manera serían demasiado tardados para valores muy grandes sin el uso de una computadora.

5. Reto1

El reto consiste en usar la técnica de Kurt [3] para estimar el valor de π mas exacto posible, sabiendo que la precisión en los decimales está relacionada al tamaño de la muestra, como fue demostrado en la tarea base. Este método se basa en la determinación del valor de π a través del área de un círculo y un cuadrado en la que la relación puede expresarse como $\pi/4$. En la figura 2 se muestran dos diagramas para el representar el valor exacto de π, donde se indican los valores generados y comparados con los esperados (los que caen en la banda de color azul), así como los valores generados en comparación con los decimales utilizados, pues como ya se mencionó anteriormente mientras mas valores existan mas se aproxima al valor exacto.

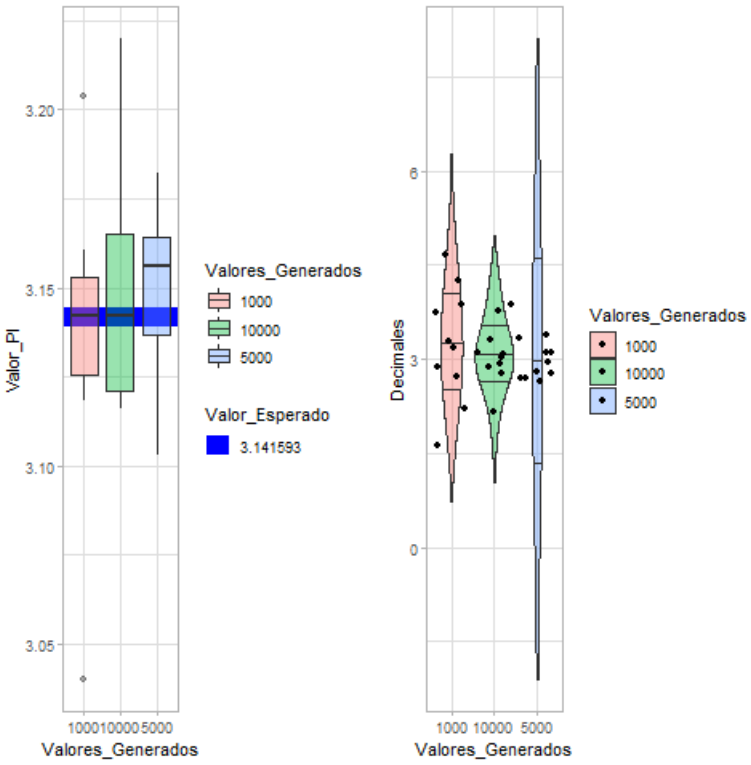


Figura 2: Representacion entre los valores generados contra el valor esperado.

Referencias

[1] E. Schaeffer. Práctica 5: Método monte-carlo, 2020. URL https://github.com/satuelisa/Simulation/blob/master/MonteCarlo/rng.R.

[2] O. Quiñonez. tareacinco, September 2020. URL https://github.com/OscarNANO/OscarNANO/tree/master/tareacinco.

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

Sistema Multiagente

Oscar Quiñonez

28 de octubre de 2020

1. Objetivo

Para realizar el proyecto 6, se implementa un sistema multiagente [1], en el que se simula el contagio entre un grupo de personas que tienen contacto entre ellas, se busca entonces determinar la probabilidad de contagio de las personas vacunadas y la manera en la que se propagaría el contagio entre las no vacunadas.

2. Metodología

En esta simulación se requirió del uso del programa R 4.0.2, tomando como base el código [2] para la modificación en cuanto al número de agentes, dado que pv representa a la parte de la muestra que ya había sido vacunada previamente y por lo tanto no se contagian ni pueden contagiar a los demás miembros de la muestra total. Los datos más relevantes, y por lo tanto mas importantes para nosotros son el porcentaje máximo de contagio que puede haber en la muestra y el momento en el que esto ocurre.

3. Resultados y Discusión

Al variar el valor de pv entre 0 y 1 con pasos de 0.1 se muestra un cambio en el comportamiento de los infectados, pues recordemos que algunos ya estaban vacunados pero la mayoría seguían sin estarlo y por lo tanto eran mas vulnerables a contraer el contagio y transmitirlo entre el resto de la muestra que tampoco había sido vacunada con anterioridad. El porcentaje máximo se ve representado en la figura 1 en la que cada barra representa una variación de 0.1 entre los valores de 0 y 1, es decir que cada una es un avance del 10 %; por lo tanto, se puede apreciar que mientras mayor sea la cantidad de vacunados, menor es la probabilidad de contagio entre el grupo y así se ve reflejada la efectividad que lleva para la población total que todos o casi todos estén vacunados.

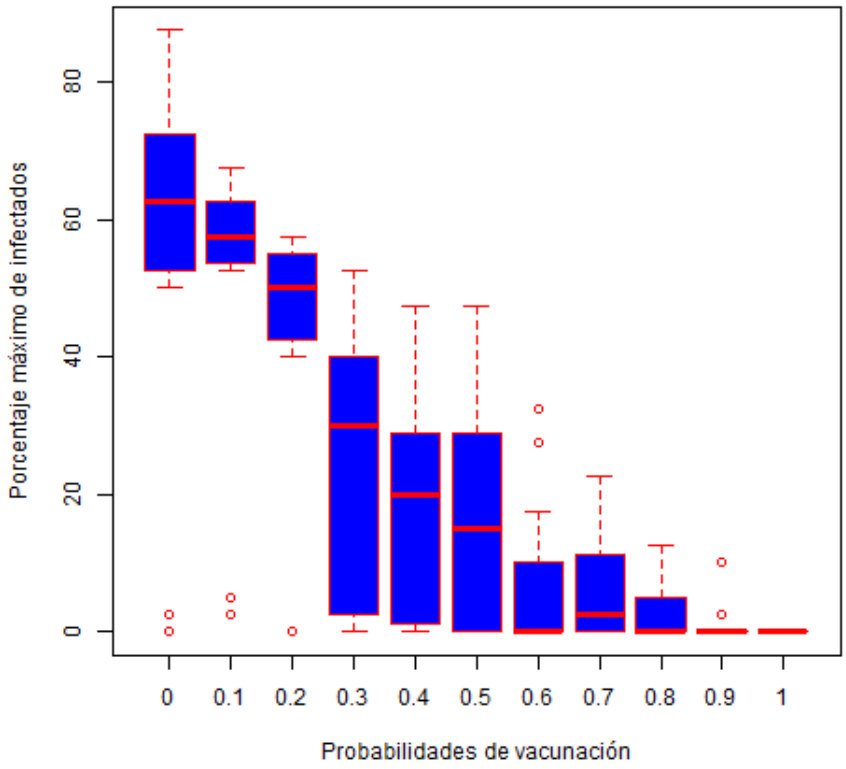


Figura 1: Variación en la probalilidad de vacunación.

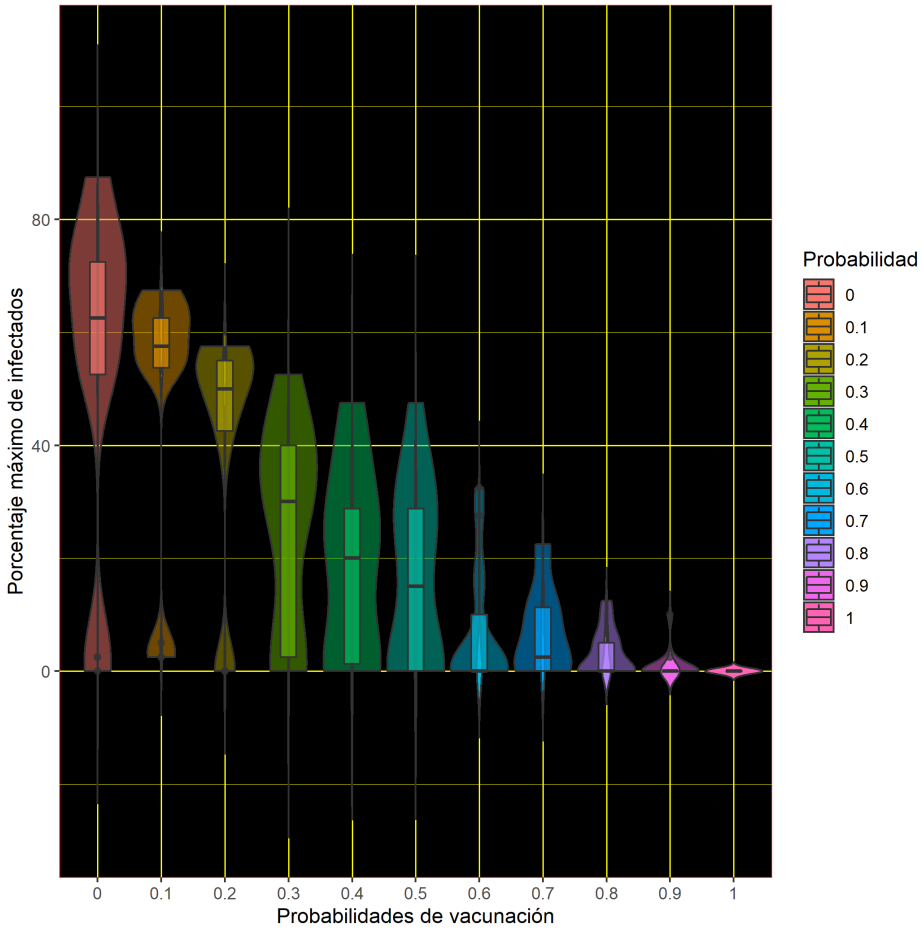


Figura 2: Porcentaje máximo de contagio.

En la figura 2 mostrada anteriormente nos indica que el porcentaje máximo de contagios es de 87.5 % de los 40 agentes estudiados, que equivalen a 35 de ellos, además de que también se muestra una disminución mientras van aumentando los vacunados. Los datos también se muestran en la tabla 1 obtenida del archivo de texto generado por el código [3].

Cuadro 1: Máximos infectados respecto a la probabilidad de vacunación

Repetición	Probabilidad	Max Infectados	Porcentaje
9	0	35	87.5 %
29	0.1	27	67.5 %
38	0.2	23	57.5 %
46	0.3	21	52.5 %
64	0.4	19	47.5 %
76	0.5	19	47.5 %
98	0.6	13	32.5 %
113	0.7	9	22.5 %
130	0.8	5	12.5 %
148	0.9	4	10 %
151	1	0	0 %

4. Conclusión

Después de haber realizado la simulación se muestra en ambas graficas que la disminución en el ritmo de contagio depende de la previa vacunación de los agentes pues de esta manera se evita tanto el contagio como la propagación por parte de un agente hacia el resto de la muestra; debido a que así no arriesga a todos los que pueden ser vulnerables.

Referencias

[1] E. Schaeffer. Práctica 6: Sistema multiagente, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p6.html`.

[2] E. Schaffer. Sistema multiagente, 2020. URL `https://github.com/satuelisa/Simulation/tree/master/MultiAgent`.

[3] O. Quiñonez. tareaseis, October 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareaseis`.

Búsqueda Local

Oscar Quiñonez

4 de noviembre de 2020

1. Objetivo

Utilizando el método heurístico se busca representar la posición de un objeto mediante el uso de diferentes rutas, al encontrar la mas corta, conoceremos lógicamente, la mas cercana, que será representada en una proyección tridimensional en forma de un mapa topográfico.

2. Metodología

Partiendo del código base [1] proporcionado anteriormente en clase, se busca desarrollar la simulación mediante el uso del programa R 4.0.3, en el cual se variaron los valores entre los ejes $g(x, y)$, que van de $-3 \leq x, y \leq 3$, de esta forma hacer que se recorran las rutas con pasos de 1.5 hasta completar una ruta de 100 pasos.

3. Resultados y Discusión

Se obtuvo una gráfica que se muestra como figura 1, en la que se puede observar una notable diferencia con la figura 2, que fue la figura obtenida con el código original ya antes mencionado, esta diferencia en la cantidad de intersecciones se debe a la aproximación al origen con respecto al número de repeticiones. Así también se observa una diferencia en los planos mostrados en las figuras 3 y 4, pues las intersecciones se dan en una menor cantidad de repeticiones y nos damos cuenta como se deshace la simetría de los planos. Todas las figuras estan disponibles [2] en el repositorio de la clase.

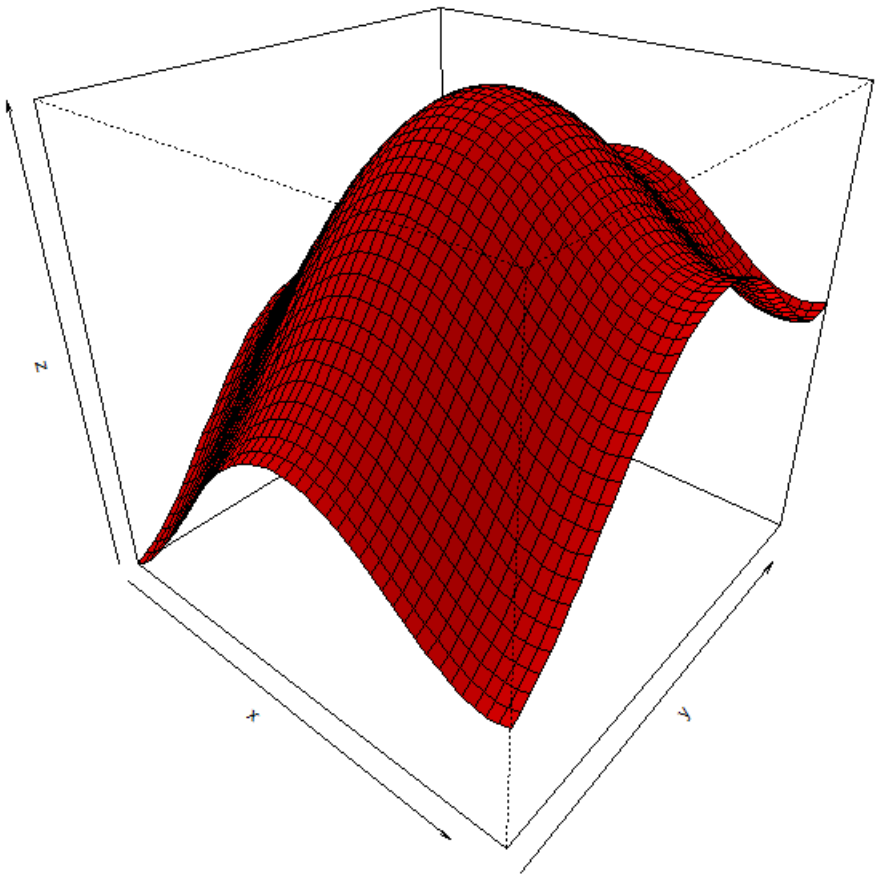


Figura 1: Gráfica con el código modificado.

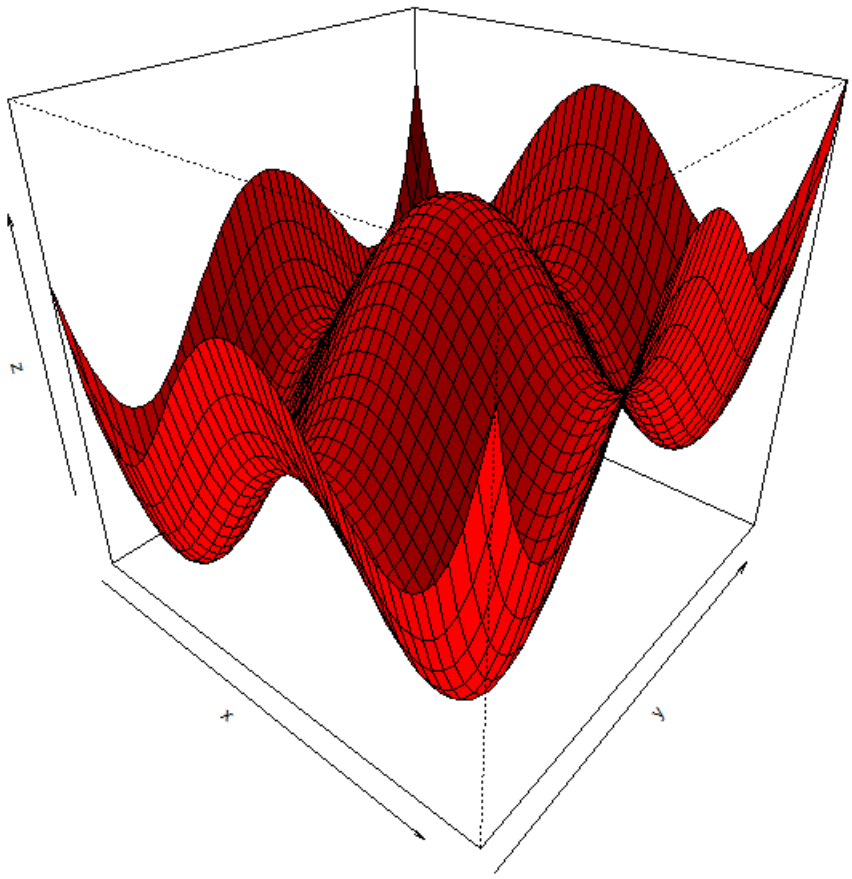
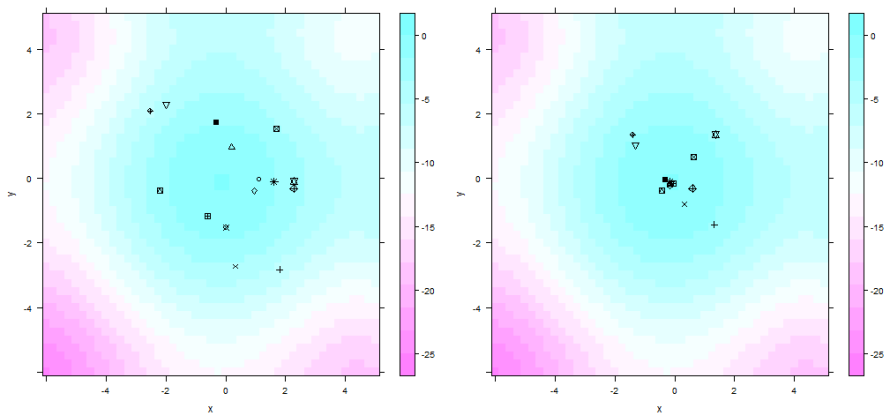
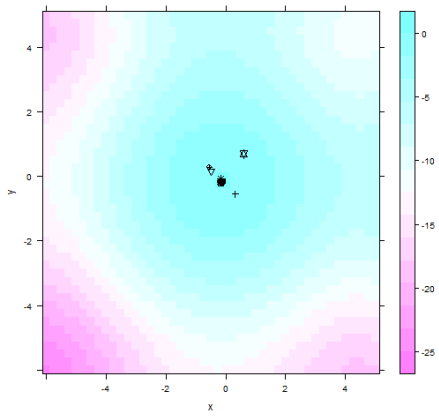


Figura 2: Gráfica con el código original.

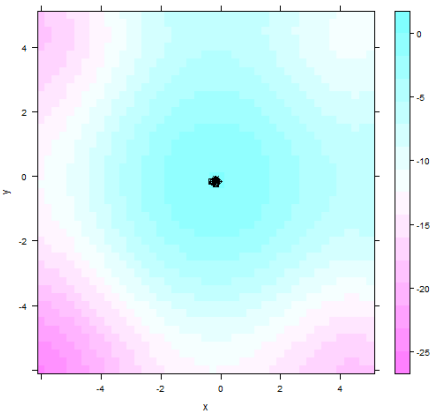


(a) Inicial

(b) repetición 25

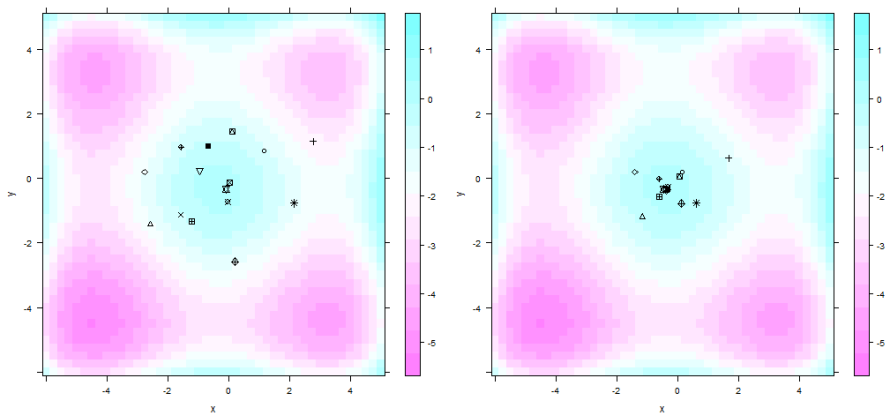


(c) repetición 50



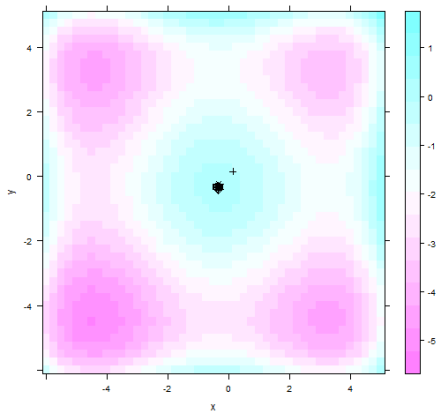
(d) repetición 100

Figura 3: Posición inicial, repetición 25, repetición media 50 y repetición final 100 (función mofificada $g(x, y)$).

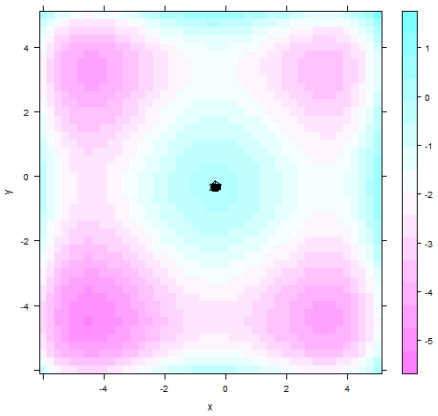


(a) Inicial

(b) repetición 25



(c) repetición 50



(d) repetición 100

Figura 4: Posición inicial, repetición 25, repetición media 50 y repetición final 100 (función original $g(x, y)$).

4. Conclusión

Después de haber realizado la simulación se muestra que el método heurístico nos ayuda a encontrar el camino mas corto en una simulación de una ruta pues como se vio en las gráficas anteriores, el punto de intersección en el origen se encuentra de manera más precisa, además como sabemos al modificar la función también cambia la forma en la aproximación.

5. Reto 1

En este reto se simula un tratamiento térmico muy conocido, el recocido, este tratamiento se le da a los metales para cambiar su estructura interna y así modificar sus propiedades, especialmente la dureza y maleabilidad. El parámetro que se busca modificar es la temperatura con valores entre 5 y 40 grados y así vemos la influencia de esta en el valor de ξ. Como se aprecia en la figura 5, la variación es muy pequeña y se determina que en este rango de temperaturas las propiedades no cambiarán mucho, pero probablemente lo hagan con temperaturas mas altas pues en 40 grados es donde comienza a verse una diferencia notable.

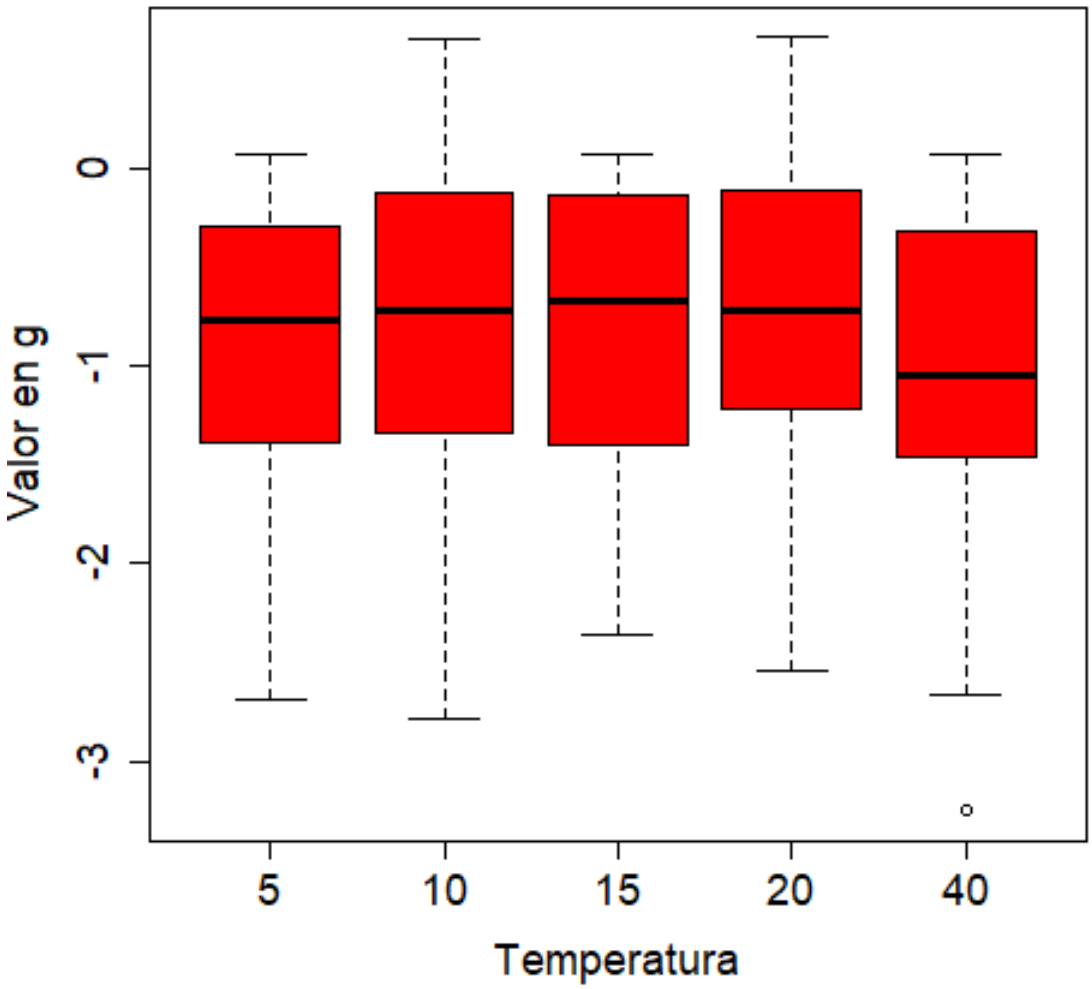


Figura 5: Gráfica de caja-bigote para cada temperatura.

Referencias

[1] E. Schaffer. busqueda local, 2020. URL https://github.com/satuelisa/Simulation/tree/master/LocalSearch.

[2] O. Quiñonez. tareasiete, November 2020. URL https://github.com/OscarNANO/OscarNANO/tree/master/tareasiete.

Modelo de Urnas

Oscar Quiñonez

11 de noviembre de 2020

1. Objetivo

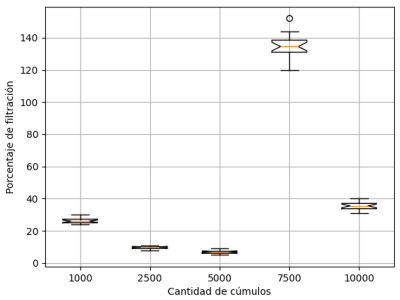
Se busca representar la fragmentación y la unión de partículas, que al unirse forman moléculas más grandes que se denominaran cúmulos, estos a su vez se pueden volver a fragmentar, por lo tanto, se intenta calcular el porcentaje de cúmulos que podrían pasar a través de un filtro debido su tamaño [1].

2. Metodología

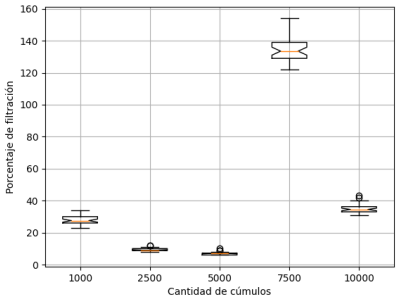
Para lograr la simulación del filtrado de las partículas, es necesario el uso del software Python 3.7, en el que se toma como base el código [2] para representar variaciones en el número de cúmulos (k), al número de partículas (n) y también al número de pasos (t) [3]. A partir de los números colocados en una lista se le da valor a (c) como tamaño critico del filtro, por lo que, sí el valor del cúmulo es mayor, será parte de los filtrados y si es menor, permanecerá en los no filtrados. Los valores de k son: 1000, 2500, 5000, 7500 y 10000; los valores en n son: 100000, 500000 y 1000000; mientras que los valores de t son: 20, 40, 60, 80 y 100.

3. Resultados y Discusión

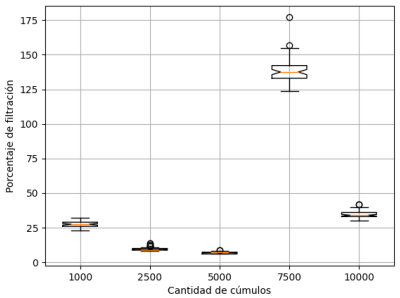
Después de procesar los datos en el código, se obtuvo una gráfica para un valor de t, por lo que fue necesario correr el código 5 veces, una para cada valor. Las gráficas que se muestran a continuación como figura 1 son de tipo caja-bigote y nos sirven para representar los valores promedio del porcentaje de filtración, para t=20 los valores de 1000, 2500, 5000, 7500 y 10000 fueron aproximadamente de 25, 12, 8, 135 y 38 respectivamente; para el resto de los valores en t (40, 60, 80 y 100) se obtuvieron promedios muy similares como se pueden apreciar al comparar las gráficas, esto nos indica que k=7500 se obtuvo el mayor porcentaje de filtración pues se acerca 140 en las 5 gráficas [4].



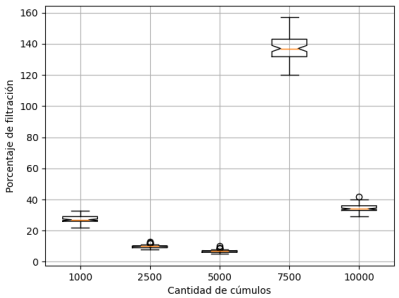
(a) t=20



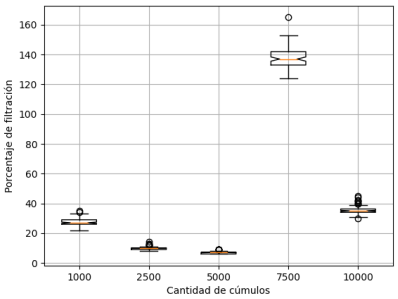
(b) t=40



(c) t=60



(d) t=80



(e) t=100

Figura 1: Gráficas obtenidas para diferentes valores en t.

4. Conclusión

Cuando se incrementa los valores del tamaño de partícula, van generando una gran cantidad de cúmulos lo que provoca una menor cantidad de filtrados, sin embargo, en 7500 no pasa esto, pues es notable la diferencia en la gráfica, lo que se atribuye a que el valor alcanza un punto en donde es menor al critico y luego vuelve a decrecer en 10000.

Referencias

[1] E. Schaffer. Modelo de urnas, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p8.html`.

[2] E. Schaffer. Modelo de urnas, 2020. URL `https://github.com/satuelisa/Simulation/blob/master/UrnModel/aggrFrag.py`.

[3] E. Schaeffer. Conversacion grupal en discord, 2020.

[4] O. Quiñonez. tareaocho, November 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareaocho`.

Interacciones entre partículas

Oscar Quiñonez

18 de noviembre de 2020

1. Objetivo

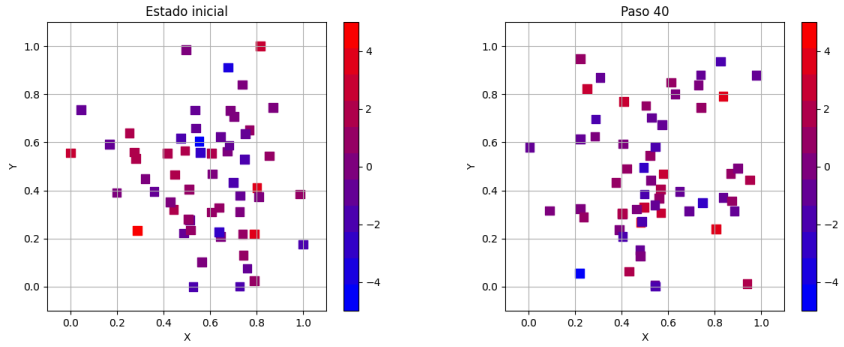
En el presente trabajo se realiza la simulación de un conjunto de partículas que interactúan entre ellos a través de las fuerzas de atracción y repulsión que ejercen sobre ellas las cargas pertenecientes a cada partícula. Así también se simula el efecto de la gravedad y las masas para conocer la manera en que se distribuye la velocidad de estas partículas.

2. Metodología

Para llevar a cabo la simulación, fue necesario el uso del programa Python 3.7, donde se colocaron 60 partículas a las que se les atribuyó una carga eléctrica aleatoria con valores entre -1 y 1, si dos partículas tenían la misma carga se ejercía una repulsión, y si había una carga diferente se presentaba una atracción [1]. Del mismo modo se atribuyó una masa aleatoria a las partículas para representar la forma en la que la gravedad las afectaba, todo esto en un periodo de 200 pasos; se usó como código base [2] el anteriormente proporcionado en clase.

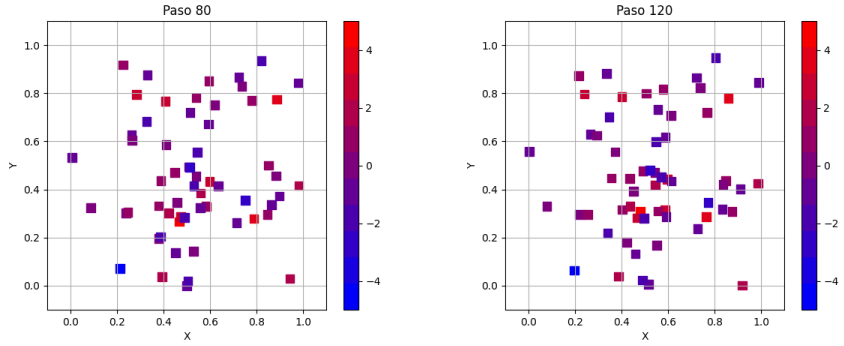
3. Resultados y Discusión

Al término de la ejecución del código se obtuvieron 2 tipos de gráficas: de dispersión, en el que se observan las posiciones de las partículas y el movimiento que realizan debido a las fuerzas de atracción y repulsión que existen entre ellas (ver figura 1) ; y un grupo de histogramas que representan la manera en la que se distribuyen las velocidades con respecto al valor de las cargas que poseen las partículas, se puede apreciar que en el rango de -0.80 a -0.65 la distribución fue nula y entre -0.20 y 0 se obtuvo la mayor parte de esta (ver figura 2). Así también se observan en las gráficas la distribución de las masas (ver figura 3) donde la mayoría de las partículas poseen un valor entre -0.6 y 1.1; además de los promedios de las velocidades (ver figura 4) donde se puede ver que el rango más destacado se encuentra entre 0.91 y 1.



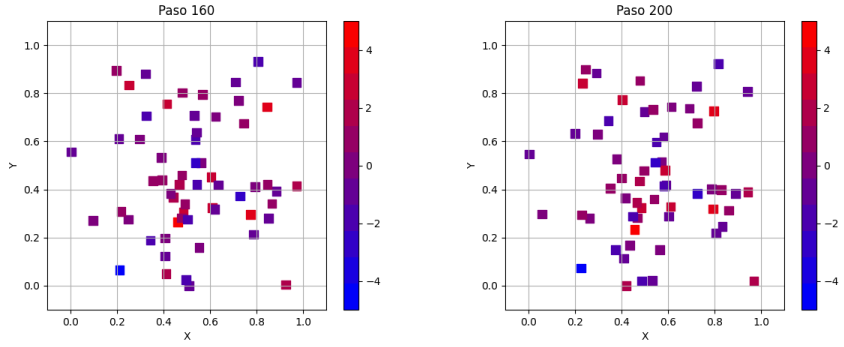
(a) Paso 0

(b) Paso 40



(c) Paso 80

(d) Paso 120



(e) Paso 160

(f) Paso 200

Figura 1: Posiciones de las partículas en incrementos lineales de 40 hasta llegar a 200.

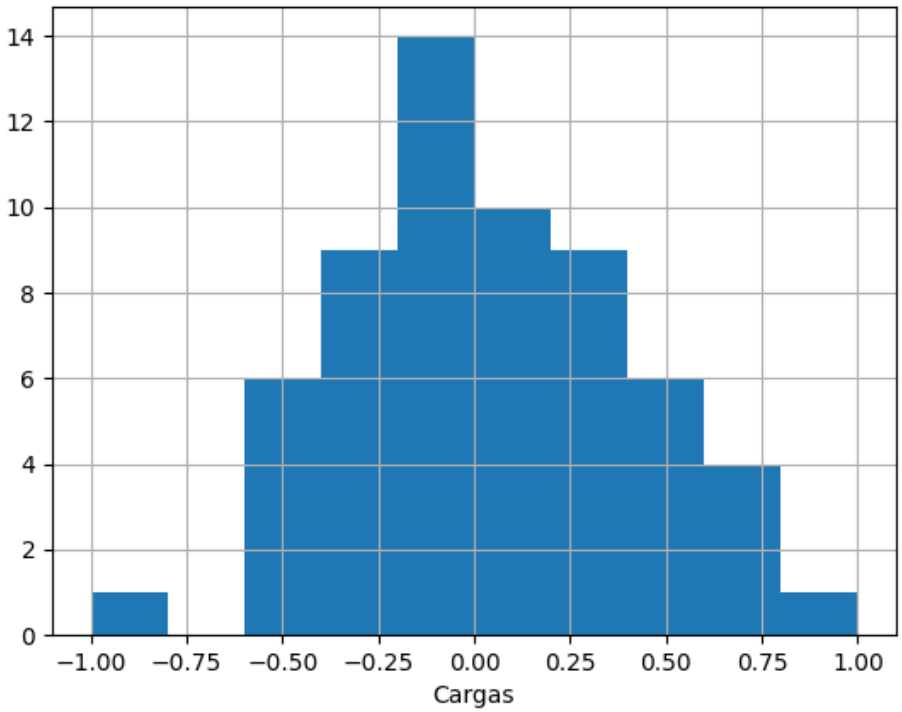


Figura 2: Distribución de las cargas

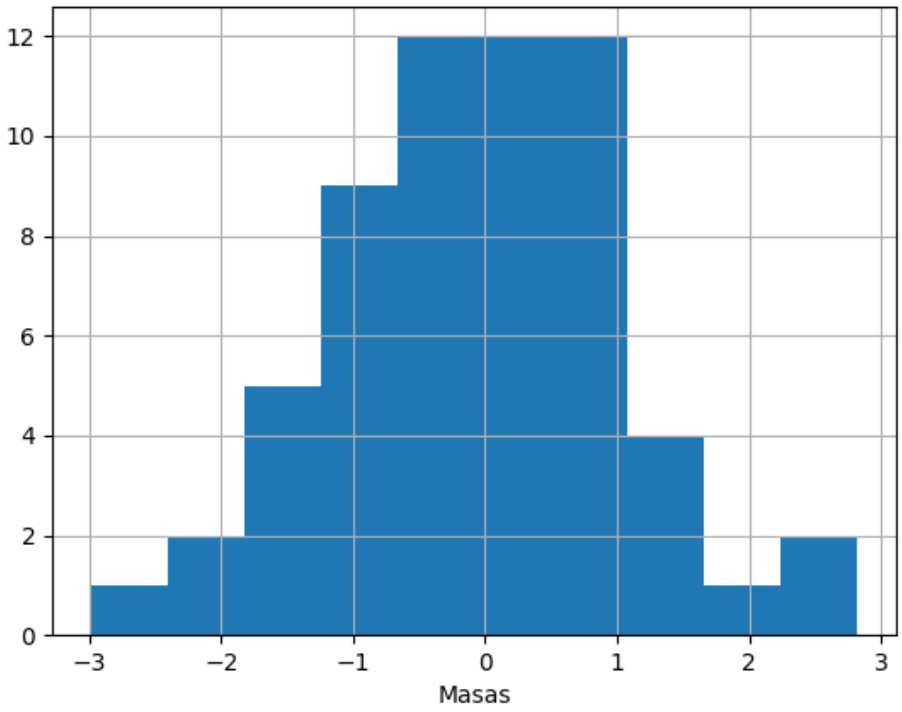


Figura 3: Distribicón de las masas

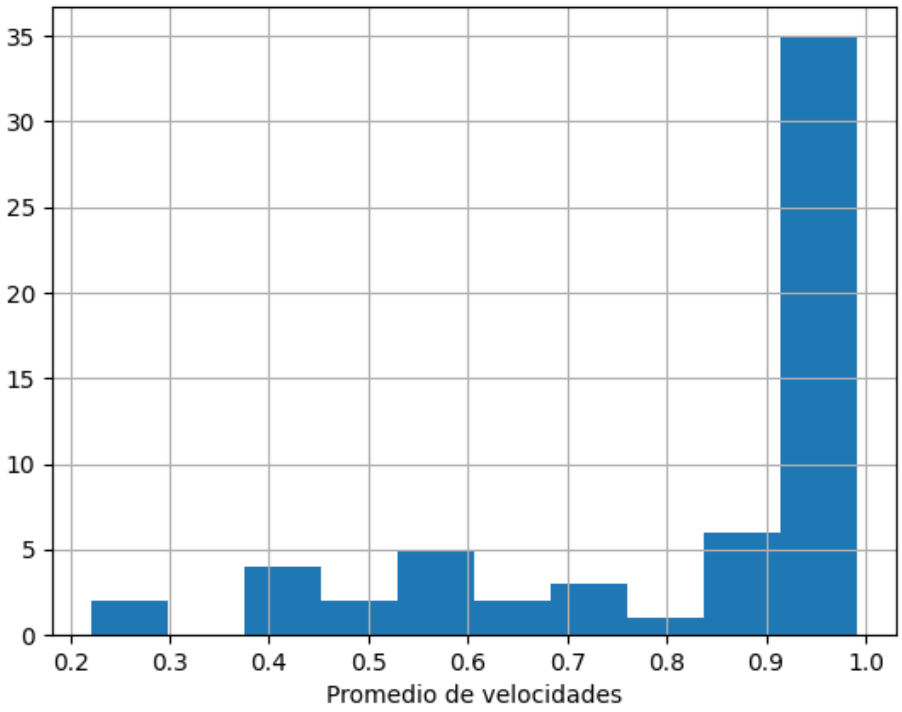


Figura 4: Distribicón de velocidades

4. Conclusión

Se pudo observar como el comportamiento de las partículas se ve afectado de acuerdo a los parámetros que le agregamos en la simulación, las masas de las partículas estaban distribuidas de manera similar entre cargas positivas y negativas, mientras que en el promedio de las velocidades si se veía una distribución muy marcada entre los valores mas altos con respecto a los mas bajos, por su parte las cargas estaban distribuidas de manera equitativa, 30 con valores negativos y 30 con valores positivos. En el apartado correspondiente [3] se encontrará con una imagen en formato GIF de la gráfica de dispersión para los 200 pasos simulados.

Referencias

[1] E. Schaffer. Interacciones entre partículas, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p9.html`.

[2] E. Schaffer. Interacciones entre partículas, 2020. URL `https://github.com/satuelisa/Simulation/blob/master/Particles/interaction.py`.

[3] O. Quiñonez. tareaocho, November 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareanueve`.

Algoritmo genético

Oscar Quiñonez

25 de noviembre de 2020

1. Objetivo

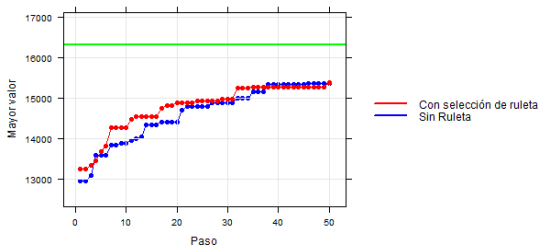
Se plantea un problema conocido como "el problema de la mochila" en el que se debe regular la cantidad de objetos que se colocan dentro de ella, debido a que tiene una capacidad limitada en cuestión de peso, en base a este problema, se busca realizar una simulación de un algoritmo genético el cual se varía la selección de los padres de la manera conocida como "ruleta" [1].

2. Metodología

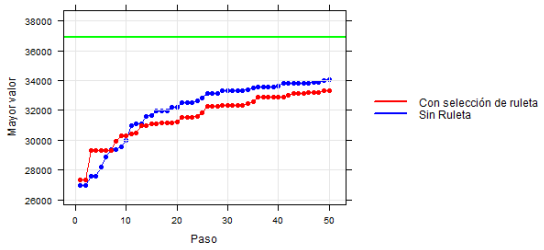
En esta simulación fue necesario el uso del programa Python 3.7 en el que se tomó como código base el proporcionado durante la clase [2] para posteriormente plantear 3 escenarios posibles: 1) el valor y el peso son independientes entre ellos, 2) el valor y el peso están relacionados entre ellos y 3) el peso es independiente pero el valor es inversamente proporcional a este. Al variar la selección de padres a tipo "ruleta", cada uno tenía una probabilidad de ser tomado en cuenta, y al tomar los 3 criterios anteriormente planteados, se puede determinar el valor del algoritmo genético y compararlo con el algoritmo exacto. Los valores de n usados en esta simulación fueron 100, 200, 300 y 400.

3. Resultados y Discusión

Al ejecutar el código con las variaciones mencionadas se obtuvieron primeramente las gráficas que nos indican la diferencia de los valores al usar o no usar la ruleta, estas gráficas se pueden apreciar en la figura 1 que se presenta a continuación. En el cuadro1 se muestra los valores obtenidos al variar la cantidad de objetos y se destacan los valores máximos y óptimos de cada uno, de manera mas representativa se pueden ver las gráficas de estos valores y como varían entre ellos en las figuras 2, 3, 4 y 5.



(a) 100 objetos

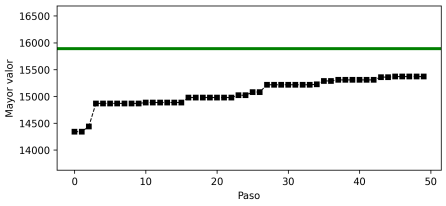


(b) 200 objetos

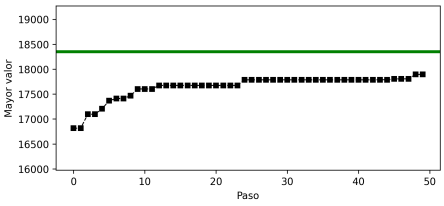
Figura 1: Comparación de la ruleta con 100 y 200 objetos

Cuadro 1: Tabla de resultados para las distintaas cantidades de n.

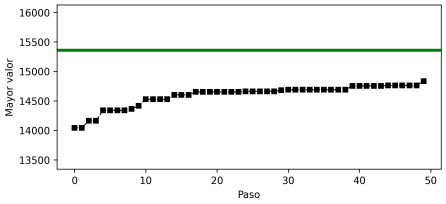
Objetos	Valor máximo	Valor óptimo	Instancia
100	7326.288381842508	75851.55212057414	1
100	68468.98351482986	55795.300566274185	2
100	51964.798682299166	43479.15872730447	3
200	15368.983690630148	13571.3126008619583	1
200	17894.877000934695	8721.51234211876	2
200	14832.970082181477	12135.028209092558	3
300	32763.83926134712	21985.93465812870	1
300	28754.157541097403	18963.883314069720	2
400	52301.06212532473	21369.348512678901	1
400	42034.773495068867	35991.030450982477	2
400	40782.74748446381	27004.364789015584	3



(a) Instancia 1

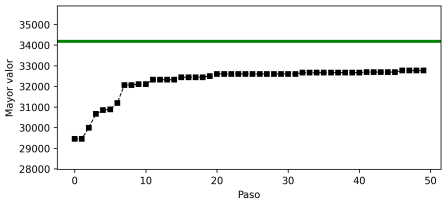


(b) Instancia 2

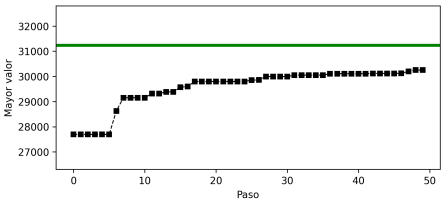


(c) Instancia 3

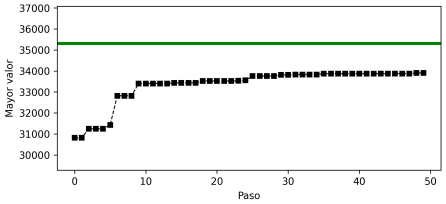
Figura 2: 3 instancias para n=100



(a) Instancia 1

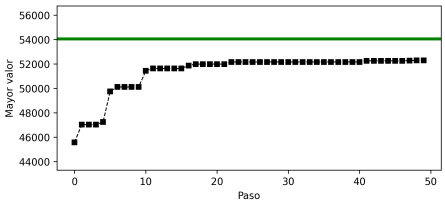


(b) Instancia 2

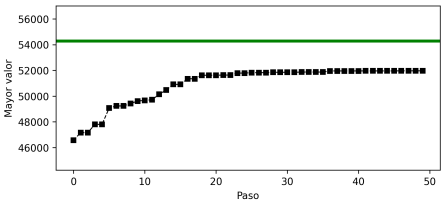


(c) Instancia 3

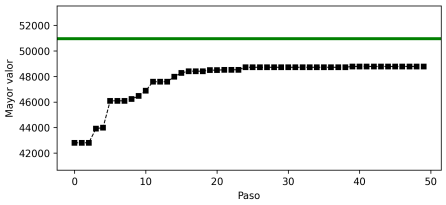
Figura 3: 3 instancias para n=200



(a) Instancia 1

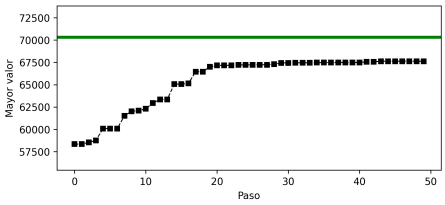


(b) Instancia 2

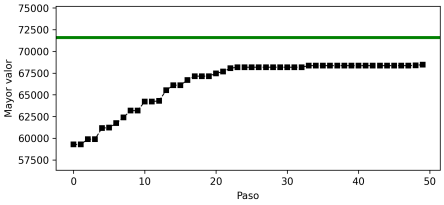


(c) Instancia 3

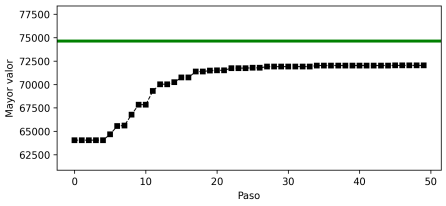
Figura 4: 3 instancias para n=300



(a) Instancia 1



(b) Instancia 2



(c) Instancia 3

Figura 5: 3 instancias para n=400

4. Conclusión

Como se puede ver en las gráficas, existe una línea verde la cual nos refleja el valor máximo de las 3 condiciones iniciales con respecto a las cantidades de n utilizadas en cada una de ellas [3]. Los valores óptimos serían los que tocan la línea verde en los valores de n indicados al pie de las gráficas, sin embargo, en esta simulación no se alcanzó este punto posiblemente por la cantidad de objetos o por la capacidad del CPU para procesar los datos, aunque se puede decir que las 3 instancias muestran resultados muy similares.

Referencias

[1] E. Schaffer. Algoritmo genético, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p10.html`.

[2] E. Schaffer. Algoritmo genético, 2020. URL `https://github.com/satuelisa/Simulation/blob/master/GeneticAlgorithm/basicGA.py`.

[3] O. Quiñonez. tareadiez, November 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareadiez`.

Frentes de Pareto

Oscar Quiñonez

2 de diciembre de 2020

1. Objetivo

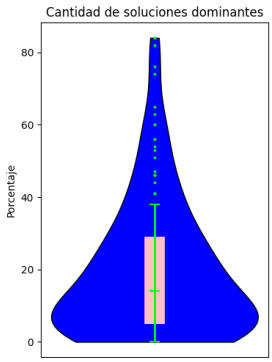
En el presente trabajo se muestran los frentes de Pareto también conocido como eficiencia de Pareto, el cual nos ayuda para la optimización de problemas con múltiples criterios, en el que puede haber contradicción entre ellos, por ejemplo, al mejorar uno de ellos, los demás empeoran y también en sentido contrario, para este trabajo se siguieron las instrucciones [1] dadas en clase.

2. Metodología

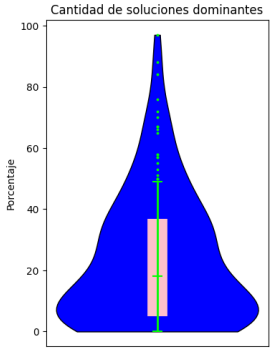
Usando como herramienta Python 3.7 se tomó el código proporcionado en clase [2] en el que se pide graficar las funciones objetivo con valores entre 2 y 12 además de usar una cantidad n de soluciones, que para este caso son 150. Al proceder con la simulación se pudieron ver cambios en las 12 gráficas de tipo violín que fueron generadas.

3. Resultados y Discusión

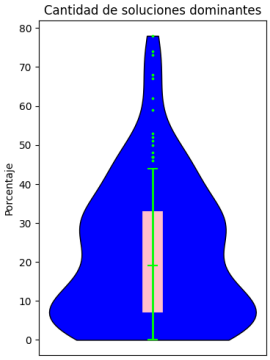
En las gráficas mostradas en la figura 1 se representan cada una de las funciones objetivo, estas nos indican que conforme su aumento, también aumentan las soluciones no dominadas, sin embargo, ninguna llega claramente al 100 %. Todas las gráficas se pueden ver en el repositorio de Github [3].



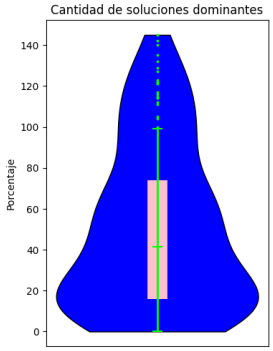
(a) Función objetivo 2



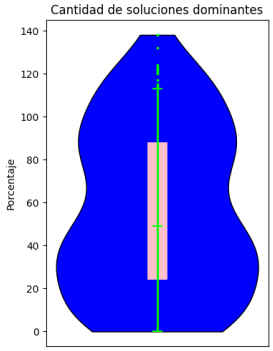
(b) Función objetivo 3



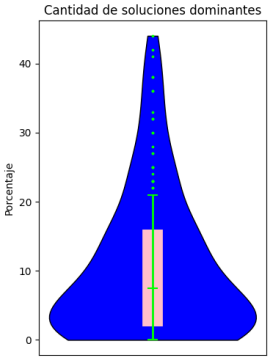
(c) Función objetivo 4



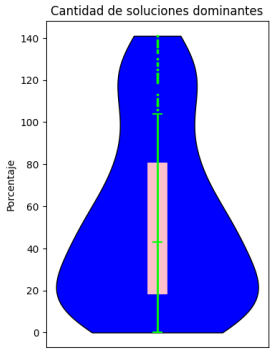
(d) Función objetivo 5



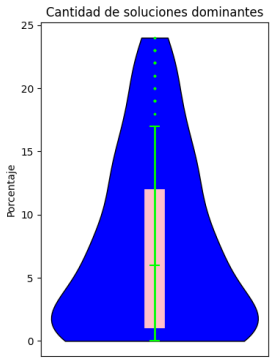
(e) Función objetivo 6



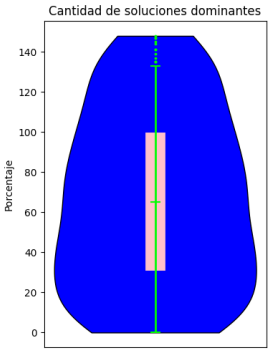
(f) Función objetivo 7



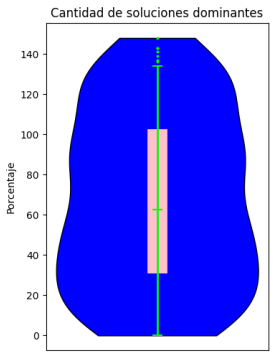
(g) Función objetivo 8



(h) Función objetivo 9

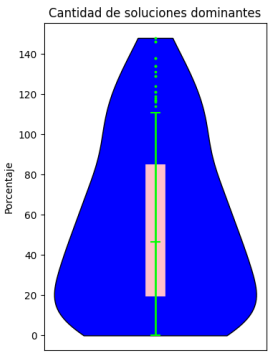


(i) Función objetivo 10



2

(j) Función objetivo 11



(k) Función objetivo 12

Figura 1: Simulación usando 150 soluciones

4. Conclusión

Se puede inferir que el efecto de las 12 muestras objetivos es debido a que se usaron 150 soluciones y tal vez con una cantidad mucho mayor se podría ver una tendencia hacia el 100 %, esto no se realizó debido a la capacidad del equipo utilizado, pero cabe aclarar que este mismo efecto sí nos serviría en una simulación multicriterio como lo son los frentes de Pareto.

Referencias

[1] E. Schaeffer. Frentes de pareto, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p11.html`.

[2] E. Schaeffer. Frentes de pareto, 2020. URL `https://github.com/satuelisa/Simulation/blob/master/ParetoFronts/violin.py`.

[3] O. Quiñonez. tareaonce, 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareaonce`.

Red Neuronal

Oscar Quiñonez

9 de diciembre de 2020

1. Objetivo

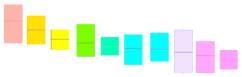
En este trabajo se busca simular la forma en como funciona una red neuronal variando las tres probabilidades de sus colores, estos son utilizados para generar sus dígitos: negro, gris y blanco. Todo esto siguiendo las indicaciones anteriormente dadas [1], ademas del archivo "digits.txt".

2. Metodología

Fue necesario el uso del programa Python 3 para realizar esta simulación, en el que se tomó como código de referencia [2] el que fue proporcionado durante la clase. Se variaron las probabilidades de los colores de forma decimal para después calcular el valor de F o también conocido como "F score" en cada matriz de confusión, todo esto llevado a cabo en 10 repeticiones.

3. Resultados y Discusión

La simulación nos da como resultado una gráfica de caja-bigote en la que se representan los valores obtenidos, estos van disminuyendo con respecto al aumento en la cantidad de ellos de izquierda a derecha (vease figura 1), este comportamiento es posiblemente causado por que se implementan valores más pequeños en la probabilidad [3].



(a) Figura 1: 10 repeticiones comparadas contra el valor de F

4. Conclusión

Las probabilidades, al ser variadas ejercen un comportamiento diferente en la forma en que se desempeña una red neuronal, de esta manera al disminuirlas, también va disminuyendo su rendimiento.

Referencias

[1] E. Schaeffer. Red neuronal, 2020. URL `https://elisa.dyndns-web.com/teaching/comp/par/p12.html`.

[2] E. Schaeffer. Red neuronal, 2020. URL `https://github.com/satuelisa/Simulation/blob/master/NeuralNetwork/ann.py`.

[3] O. Quiñonez. tareadoce, 2020. URL `https://github.com/OscarNANO/OscarNANO/tree/master/tareadoce`.