Grupo 8

Práctica 1:
Optimización de funciones

Alejandro Tobías Ángeles
Miguel Antonio Amato Hermo

Gráficas

Función 1: Calibración y prueba.

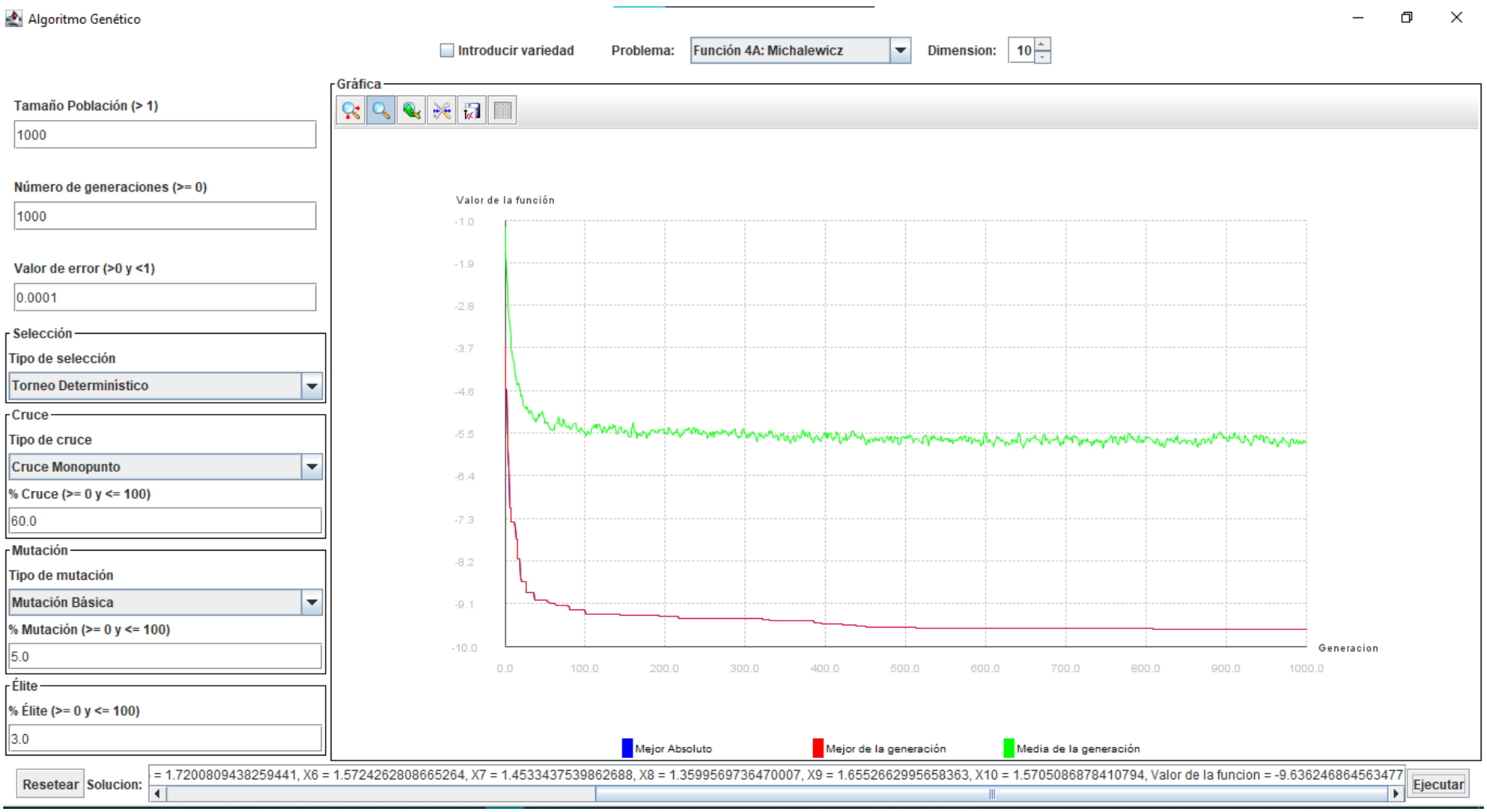
Función 2: Función de GrieWank.

Función 3: Styblinski-tang

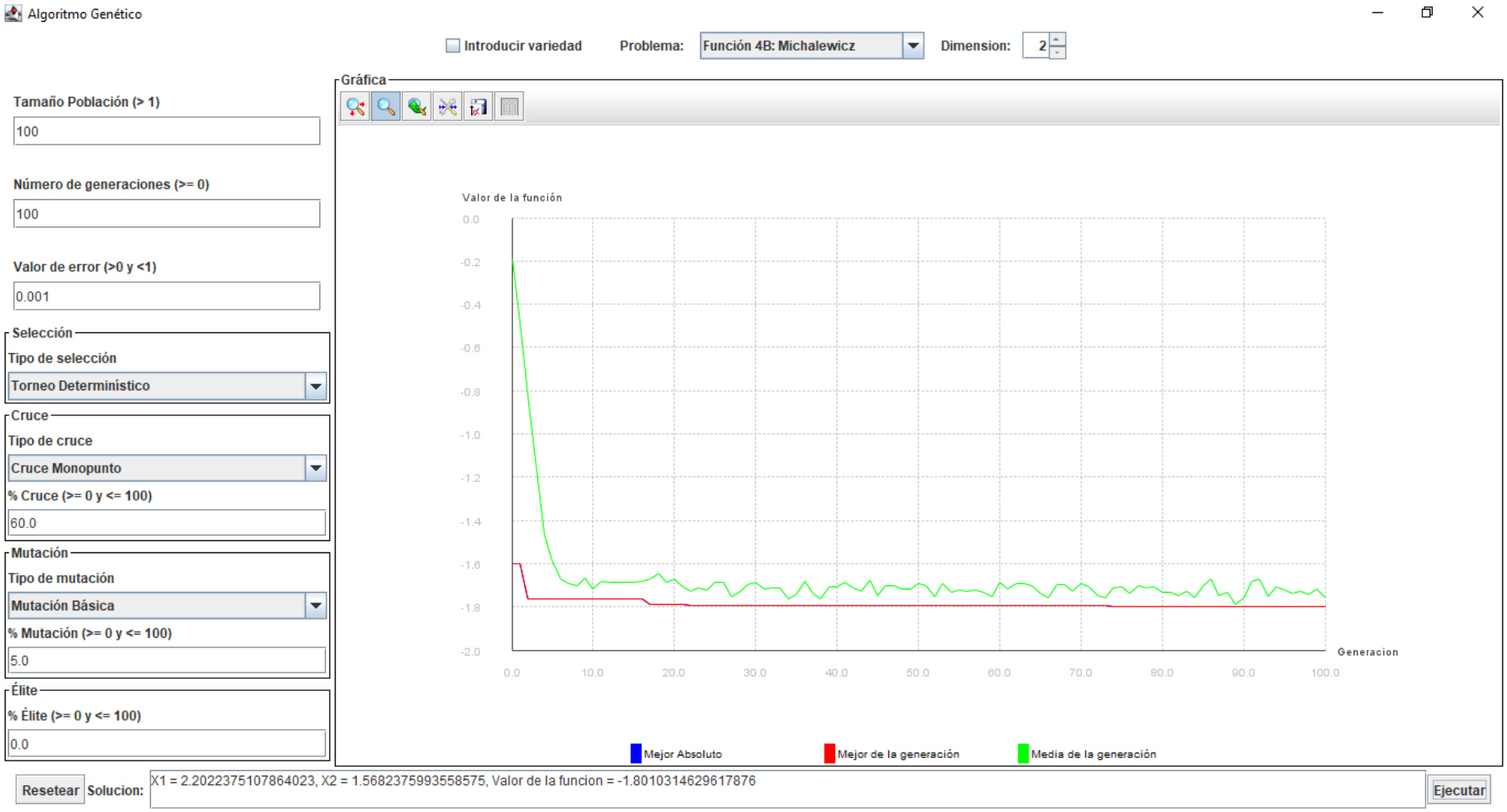
Función 4a: Michalewicz (Dimensión = 2)

Función 4a: Michalewicz (Dimensión = 5)

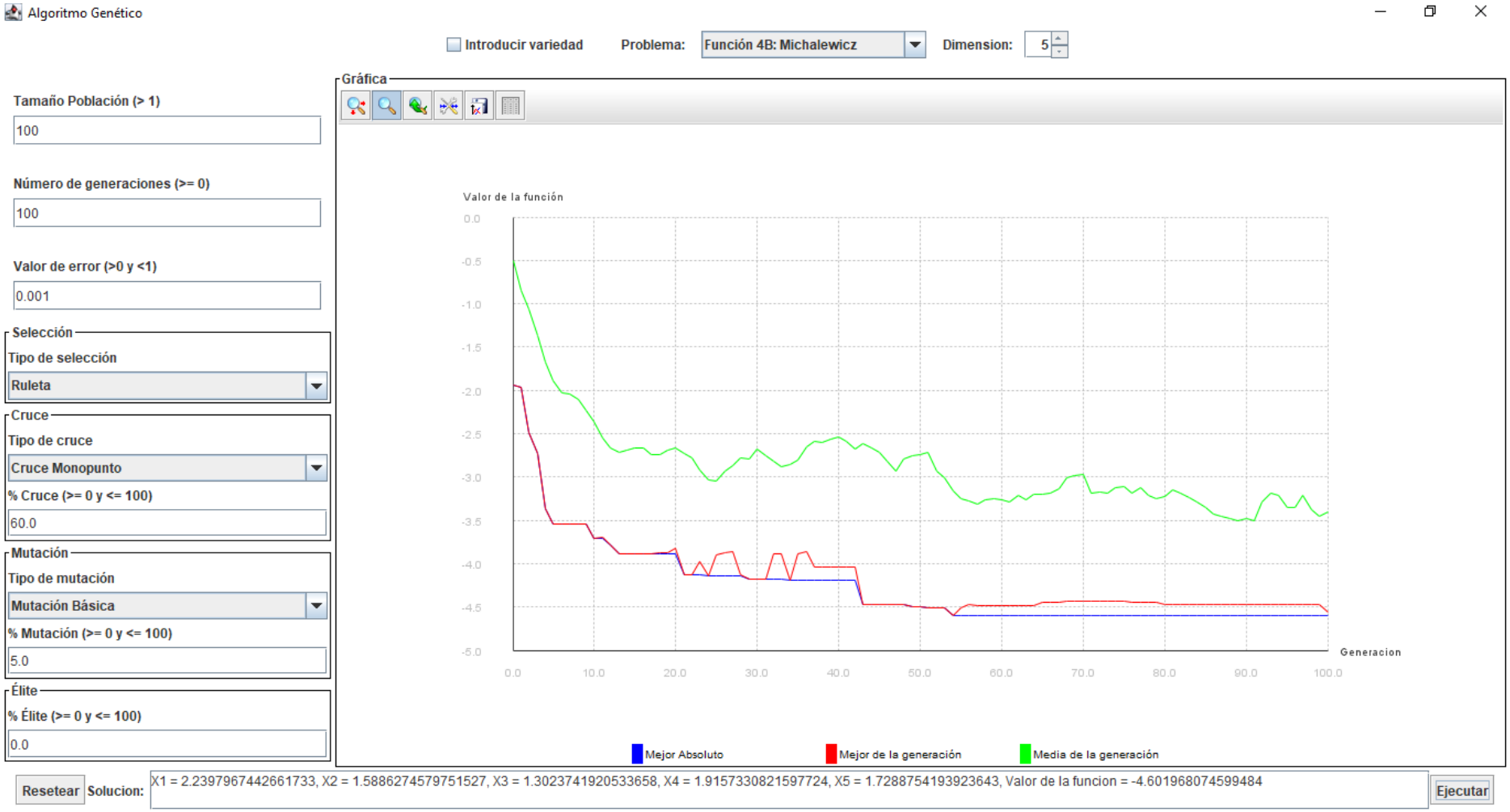
Función 4a: Michalewicz (Dimensión = 10)



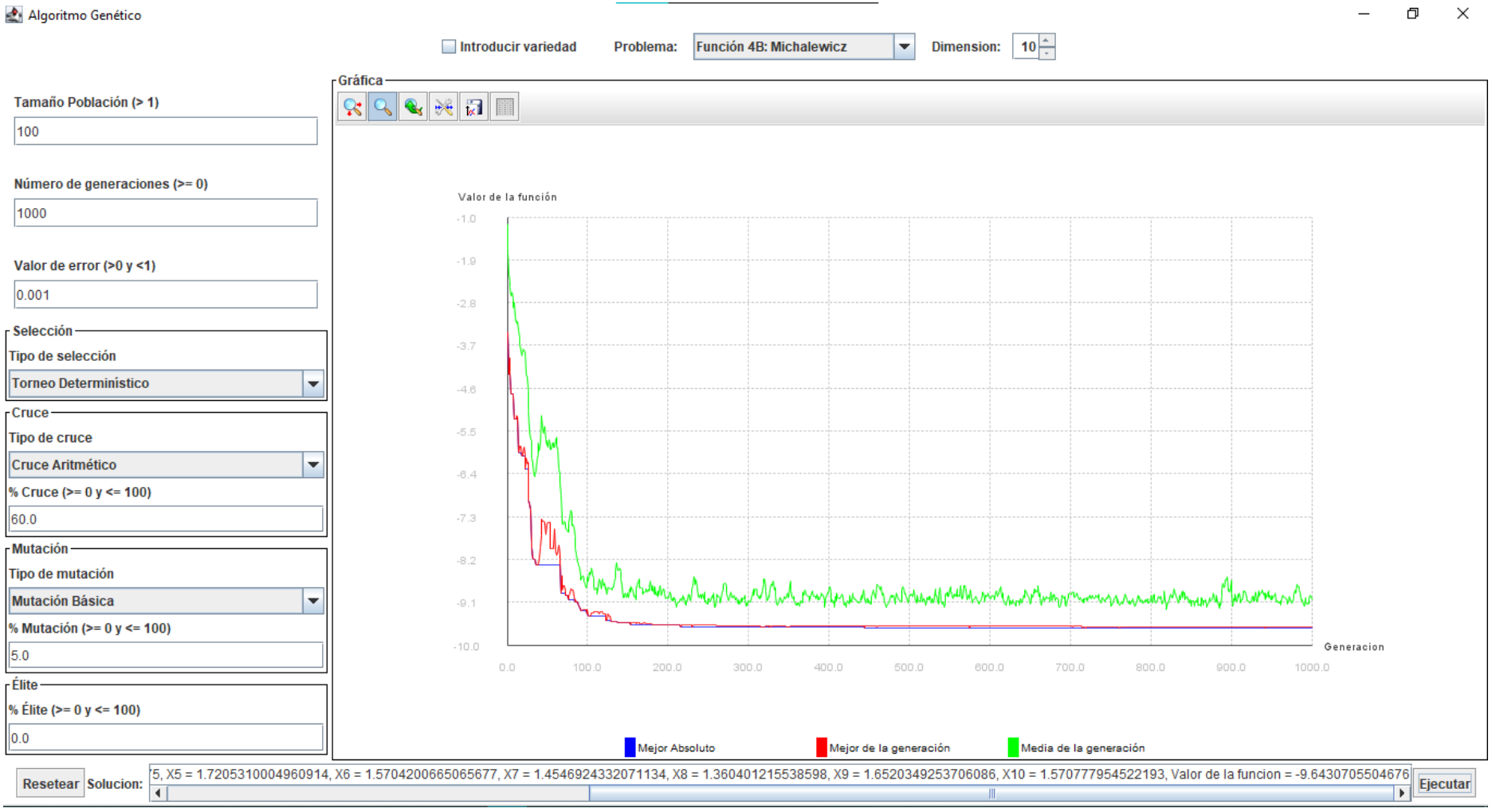
Función 4b: Michalewicz (Dimensión = 2)



Función 4b: Michalewicz (Dimensión = 5)



Función 4b: Michalewicz (Dimensión = 10)



Conclusiones

Función 1:

El método de selección que hemos utilizado ha sido el torneo determinístico. Con esta técnica se observa que la media de la generación asciende rápidamente al principio y luego se estabiliza en torno a un fitness de 33 y que el mejor absoluto y el mejor de la generación se solapan continuamente. No se han dado mejoras notables utilizando elitismo para esta función si la hacemos con torneo determinístico.

Función 2:

Para esta función hemos optado de nuevo por el torneo determinístico, asimismo hemos decidido utilizar algo de elitismo para que si en algún momento surgen individuos cuyas variables sean valores cercanos a 0, que son difíciles de conservar durante las generaciones debido a su genotipo tan difícil de obtener en un rango de valores tan amplio como es el [-600,600] , puedan mantenerse en la población durante la ejecución. También se puede observar como la media de la generación gracias al elitismo y al torneo determinístico desciende muy rápidamente pasando, en este caso, de un valor de aproximadamente 60 a un valor cercano a 8 en cuestión de unas 5 generaciones.

Función 3:

En la función 3 hemos recurrido al método de selección de truncamiento, esto debido al pequeño rango de valores que pueden tomar las variables que componen nuestro individuo ([-5,5]), por lo que aprovechar el elitismo de esta técnica tiene un efecto especialmente bueno a la hora de obtener individuos que alcancen en las primeras generaciones un valor bastante óptimo. Esto último puede observarse con claridad en la gráfica, ya que el mejor de la generación y el mejor absoluto están solapados la mayor parte del tiempo.

Funciones 4a y 4b:

Para la función de Michalewicz hemos optado por estrategias distintas para cada dimensión:
Para la dimensión 0 con individuos booleanos, al solo poder tomar valores en un rango relativamente pequeño, pese a que hemos utilizado el método de ruleta, que cuenta con un fuerte componente de aleatoriedad, se puede ver que la media converge muy rápido en valores cercanos a la solución, por lo que utilizar elitismo no supone un cambio muy relevante. En cuanto a los individuos de tipo double no se observan diferencias más allá del fuerte solapamiento por usar torneo determinístico.

Para la dimensión 5, el método con el que hemos logrado la mejor ejecución con individuos de tipo booleano ha sido el truncamiento, que gracias a su elitismo consigue estabilizar la media rápidamente. Por otro lado, en los individuos de tipo double hemos utilizado la ruleta, que logra una buena convergencia sin elitismo.

Para la dimensión 10, debido a la dificultad que supone conseguir que tantos individuos se sincronicen para tomar unos valores determinados hemos aumentado el número de generaciones, para dar más oportunidad de que surjan buenos individuos. Hemos utilizado torneo determinístico en ambos, ya que hemos observado que, en general, es el método de selección que mejores resultados nos da.

Detalles de la implementación:

Para la arquitectura del programa hemos usado modelo-vista-controlador, donde el modelo es el algoritmo genético. También hemos usado dos patrones de diseño: el patrón factory para la creación de los diferentes individuos, selecciones y cruces; y el patrón observer para la comunicación entre la vista y la lógica.

Todos los individuos (Las funciones) heredan de una clase padre Individuo que contiene aquellos atributos y métodos que comparten todos los individuos en común (tamaño de genes, cromosoma, max, min, fitness). De esta clase heredan otras dos, IndBool para aquellos individuos que tienen la codificación en binario e IndDouble para los que tienen la codificación en números reales. Por último, cada función heredará de alguna de las dos clases anteriores dependiendo de cual sea su codificación.

Cada individuo se crea mediante su propia factoría, esta factoría implementa la interfaz IndividuoFactory, la cual tiene dos métodos: Uno que crea a un individuo desde 0 (los genes generados aleatoriamente) y otro que crea un individuo asignando un cromosoma predeterminado.

Para la selección y el cruce hemos usado también el patrón factory. Cada selección implementa la interfaz Seleccion que tiene el método seleccionar al cual se le pasa por parámetros la población y el número de individuos a seleccionar y devuelve un ArrayList con los índices de los individuos seleccionados de la población. Y cada Cruce implementa la interfaz Cruce la cual tiene el método cruzar donde se le pasa por parámetro la población, el ArrayList de índices de los seleccionados y la probabilidad de cruce. Este método devuelve un ArrayList con los cromosomas de la nueva generación

Por el momento al solo tener que implementar la mutación básica simplemente hemos hecho que cada individuo, en función de si es de tipo booleano o double tenga su propia implementación. En un futuro añadiremos, al igual que hicimos con el cruce y la selección, una interfaz de mutación, que incluirá las técnicas que tengamos que añadir.

Ahora mismo la forma de decidir si cada individuo maximiza o minimiza su función es a través de la implementación que tenga del método compareTo(). Tenemos pensado que para un futuro, al no existir herencia múltiple en java lo más apropiado es crear una interfaz con métodos para máximos y otra con métodos para mínimos y que en función de lo que se busque simplemente los individuos tengan que implementar una u otra.

Reparto de tareas:

Toda la práctica ha sido realizada por ambos miembros del grupo.

Aviso:

Importando el proyecto todo lo relacionado con las gráficas debería funcionar con normalidad. En caso de no hacerlo, únicamente habrá que importar el jar “jmathplot.jar” al classpath del proyecto.