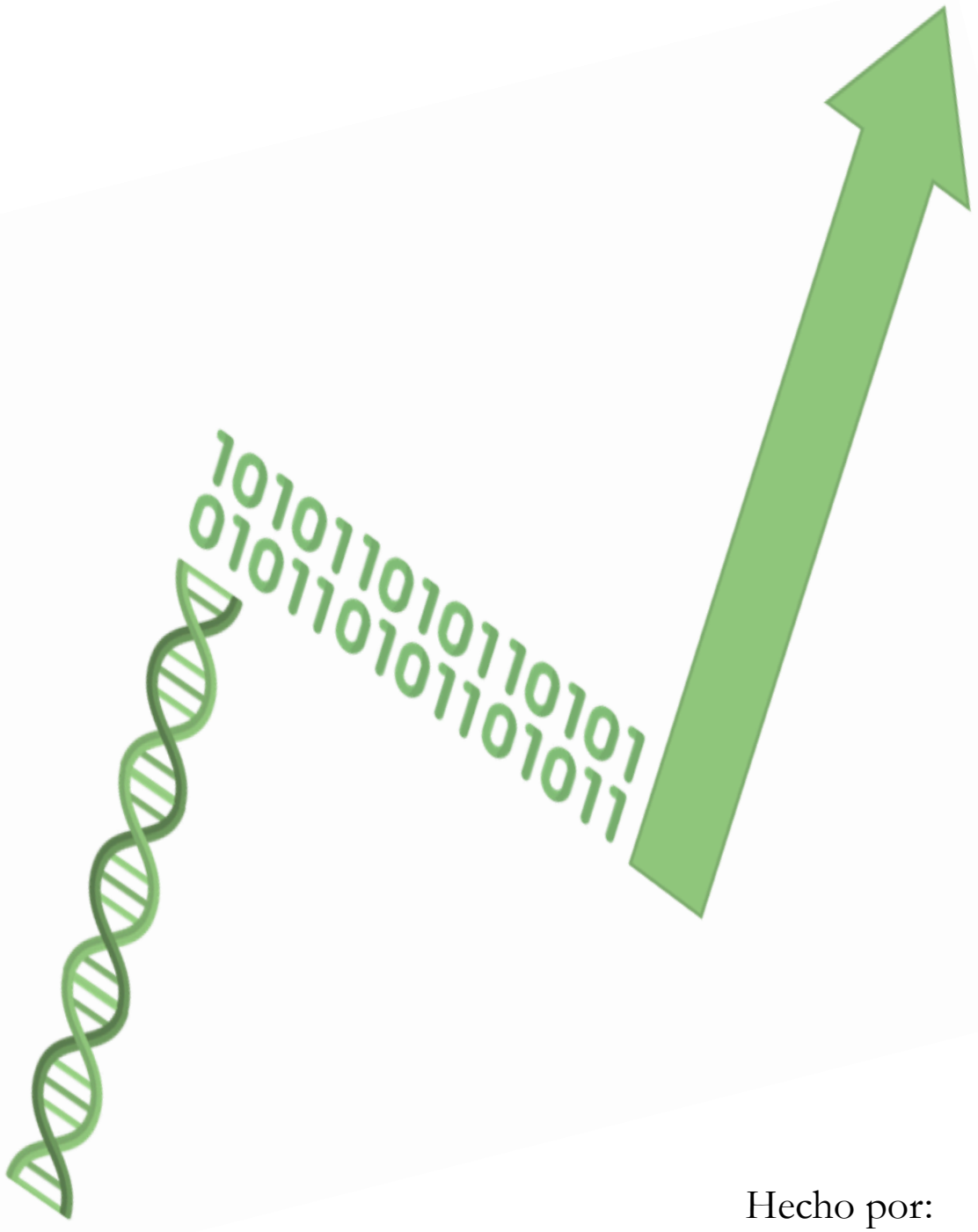
Práctica 1

Algoritmos Genéticos Básicos



Hecho por:

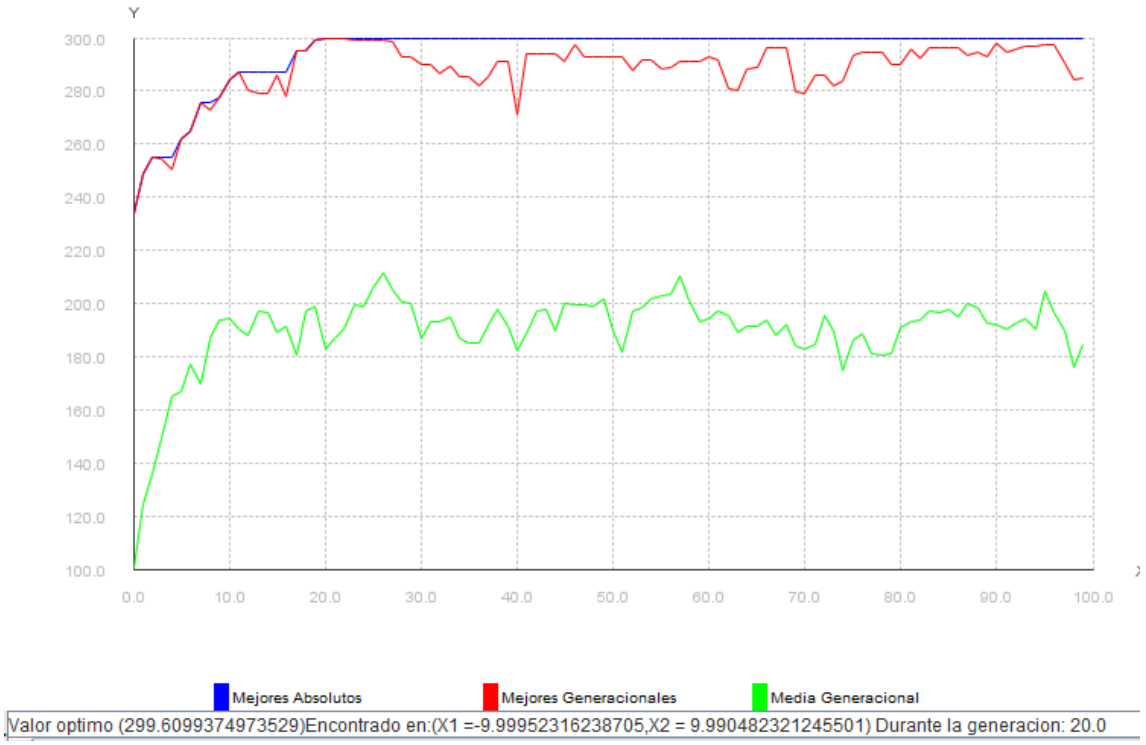
Daniel Couto Seller

Santiago González García

Gráfica representativa de cada función

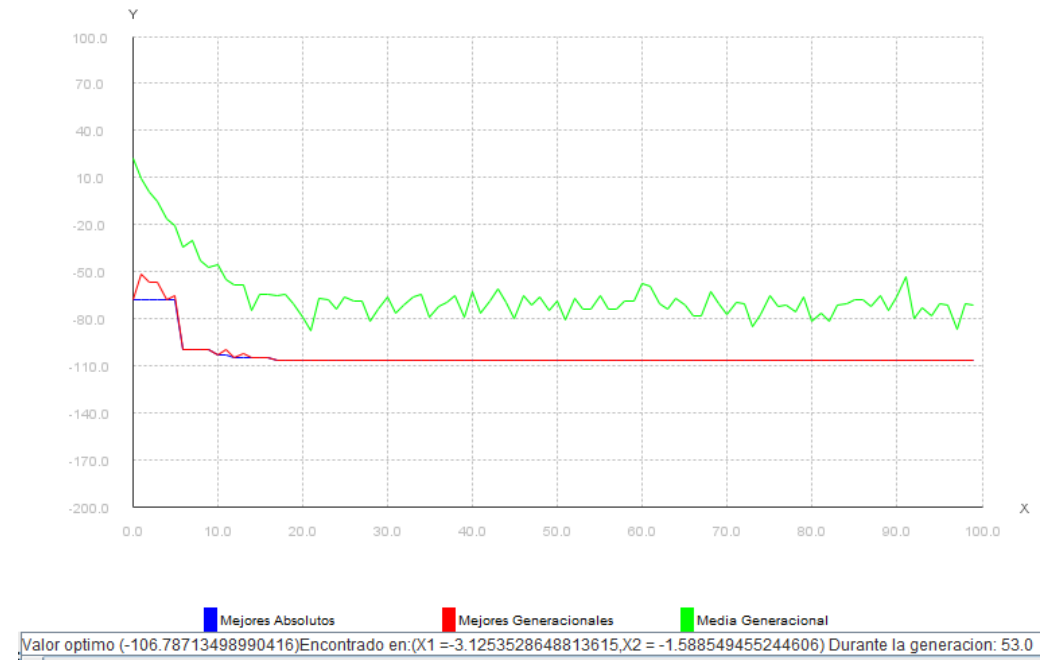
- Función 1:

  Parámetros: Tamaño Generacion (100), Número Generación (100), Porcentaje Cruce (60), Porcentaje Mutación (5), Precisión (0.00001), Selección(Estocástico), Cruce (Mono Punto), Mutación (Básico), Porcentaje Elitismo (0).



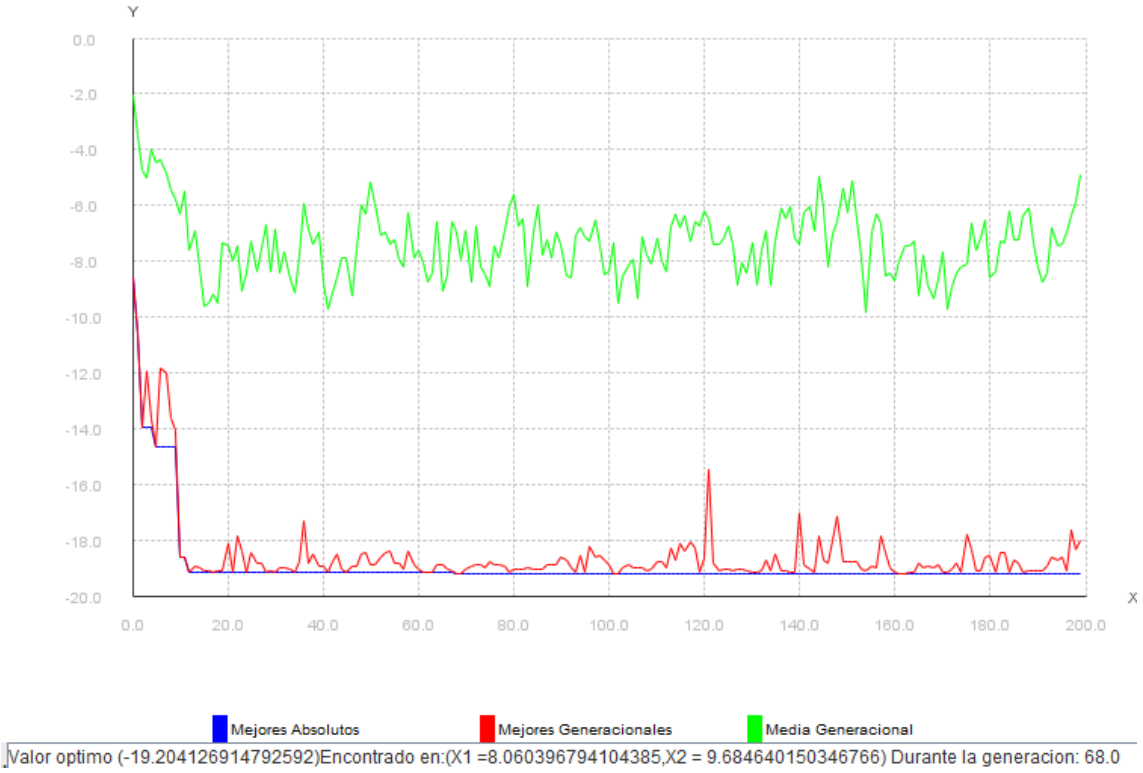
- Función 2:

  Parámetros: Tamaño Generacion (60), Número Generación (100), Porcentaje Cruce (70), Porcentaje Mutación (5), Precisión (0.0001), Selección (Truncamiento), Cruce (Uniforme), Mutación (Básico), Porcentaje Elitismo(0).



- Función 3:

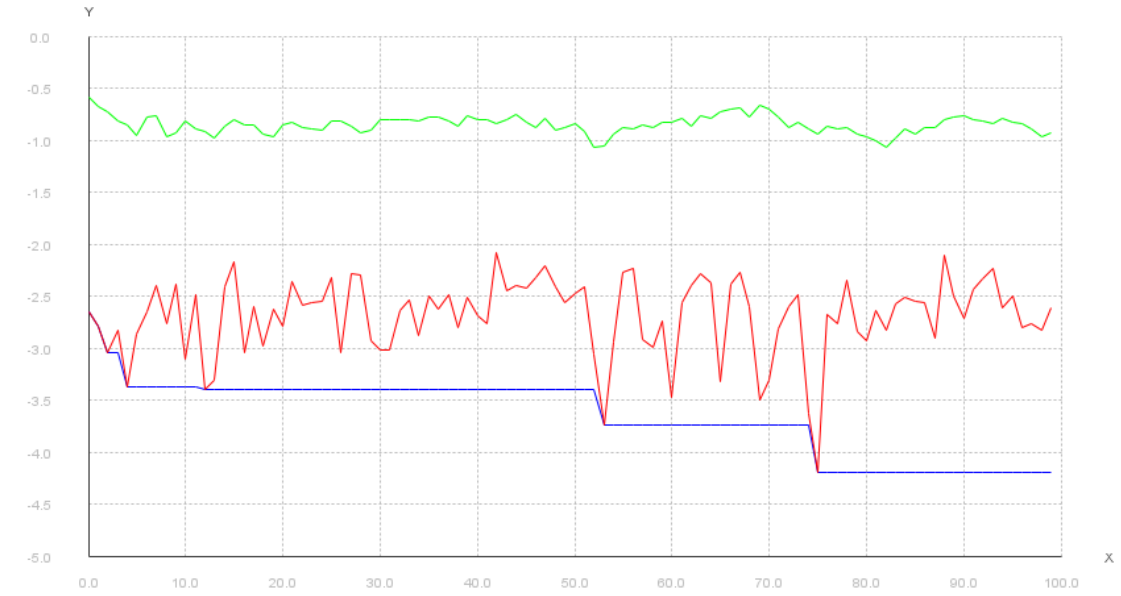
  Parámetros: Tamaño Generacion (50), Número Generación (200), Porcentaje Cruce (70), Porcentaje Mutación (10), Precisión (0.000001), Selección (Restos), Cruce (Uniforme), Mutación (Básico), Porcentaje Elitismo(0).



Valor optimo (-19.204126914792592)Encontrado en:(X1 =8.060396794104385,X2 = 9.684640150346766) Durante la generacion: 68.0

- Función 4:

  Parámetros: Tamaño Generacion (120), Número Generación (100), Porcentaje Cruce (40), Porcentaje Mutación (15), Precisión (0.000001), Selección (Torneo Determinístico), Cruce (MonoPunto), Mutación (Básico), Porcentaje Elitismo(0), Dimensiones (5).



Valor optimo (-4.2006649791403134)

Encontrado en:(

:(X1 = 2.1760606432665037,

X2 = 1.5815349464447257,

X3 = 1.2217683432358366,

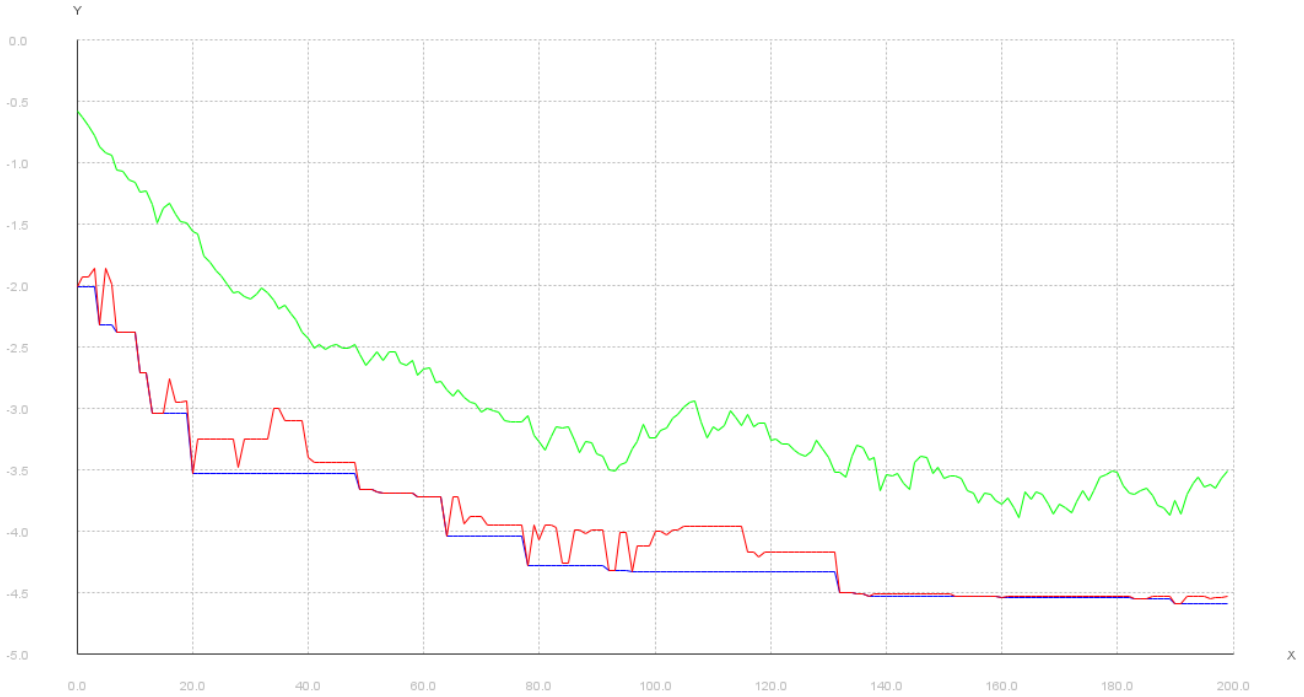
X4 = 1.8852123542949069,

X5 = 1.7180578036376057)

) Durante la generacion: 75.0

- <u>Función 5:</u>

  Parámetros: Tamaño Generacion (100), Número Generación (200), Porcentaje Cruce (50), Porcentaje Mutación (5), Selección (Torneo Determinístico), Cruce (MonoPunto), Mutación (Básico), Porcentaje Elitismo(0), Dimensiones (5).



Valor optimo (-4.5900500569648095)Encontrado en:(X1 = 2.1658487034107172, X2 = 1.580526481730592,

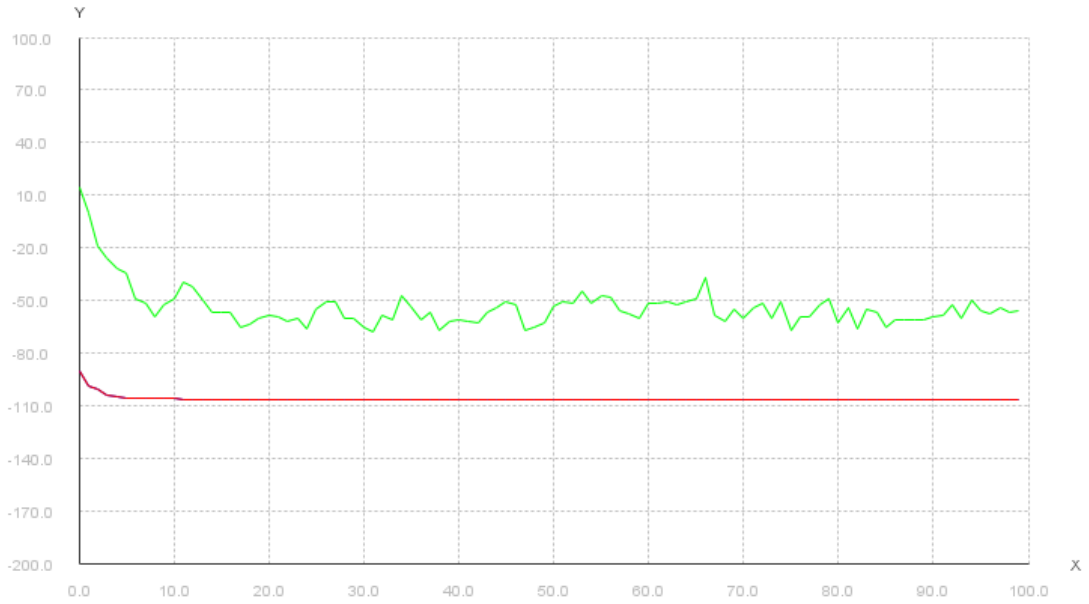
, X3 = 1.2675638352198364, X4 = 1.1003103091225048, X5 = 1.720587204093729) Durante la generacion: 190.0

Conclusiones de los resultados

Tras muchas pruebas con todas las funciones hemos llegado a varias conclusiones.

Elitismo

El elitismo asegura la supervivencia de unos pocos super individuos, seleccionados antes del inicio del proceso evolutivo y reintroducidos después de este proceso. Al aplicar el elitismo, se observa que la línea roja (mejor individuo generacional) coincide en toda la gráfica con la línea azul (mejor individuo de todas las generaciones). Esto se debe a que en cada generación, al preservar una serie de super individuos de la generación anterior y reintroducirlos, reemplazando a los peores individuos de la nueva generación, aseguramos que o bien haya un nuevo mejor individuo en esta nueva generación o que el mejor individuo de la generación anterior se encuentre en la actual. De este modo siempre coincide el mejor generacional con el mejor absoluto. Ejemplo de la función 2 con elitismo de 5%:



Métodos de selección

Entre las funciones de selección nos encontramos con las siguientes: ruleta, estocástico, torneo determinístico y probabilístico, y restos. Después de realizar varias pruebas con todas las funciones, podemos observar que la selección por ruleta, la estocástica y el torneo determinístico tienden a un comportamiento elitista por lo que se pierde diversidad genética. Esto se debe a que depende mucho de la aptitud de los individuos, de este modo si hay un fitness mucho mayor que el resto, este individuo será replicado muchas más veces. Por otra parte, la selección por torneo probabilístico arregla este problema ya que dependiendo de un porcentaje a veces escogerá el mejor o el peor de entre individuos seleccionados al azar. Por otra

parte la selección por truncamiento dependerá de la variable trunc que hayamos puesto. Cuanto menor sea esta variable, mayor elitismo tendrá la función ya que replicará más veces a los mejores aunque es inevitable perder a los peores. Esto se traducirá en poca variabilidad genética, lo que puede llevar a una convergencia prematura en un máximo local.

Mejoras

Al explorar las posibles mejoras sobre nuestro algoritmo genético hemos decidido implementar el desplazamiento de la aptitud sobre las funciones de minimización, el elitismo como nos pedía el enunciado y hemos intentado aplicar la contractividad.

En cuanto al desplazamiento de la aptitud lo hemos aplicado sobre las funciones de minimización al evaluar la aptitud de sus individuos,en cada generación hemos encontrado el máximo y hemos adaptado su aptitud siguiendo la siguiente fórmula

$$fi = fmax - gi$$

Donde la aptitud adaptada de cada individuo i en una generación será dado por el 105% de la máxima aptitud de la generación menos la actitud real del individuo.De esta forma nos aseguramos que todas las aptitudes son positivas, transformando así el problema de minimización en uno de maximización.

En cuanto al elitismo como ya hemos mencionado previamente, en cada generación hemos apartado una serie de super individuos y después del proceso de evolución los hemos reintegrado en la población reemplazando a los individuos menos aptos.

Por último hemos intentado aplicar la contractilidad a nuestro algoritmo, que consiste en solo considerar las generaciones que incrementen su aptitud media respecto a su anterior.Pero al no haber establecido una condición de terminación más que superar el máximo de generaciones establecido el algoritmo se quedaba estancado puesto que la media de las generaciones no puede mejorarse indefinidamente.Por esta razón hemos dejado comentado al línea en el algoritmo.

Análisis de convergencia

Observando los gráficos de convergencia (gráfica donde sale la media generacional, mejor generacional, y mejor absoluto) nos damos cuenta que el mínimo o máximo de la función se alcanza rápidamente durante las primeras generaciones. Esto es un buen indicativo de que el algoritmo aprovecha de forma efectiva las buenas soluciones encontradas en las etapas más tempranas. Sin embargo hay que tener en cuenta que la convergencia sea rápida no garantiza que la calidad de la solución, es decir, cabe la posibilidad de que se estanque en un óptimo local. Tras muchas pruebas hemos podido comprobar que por lo general llega al mínimo o máximo de la función dado en el enunciado, o se queda muy cerca de este.

Detalles de la implementación

Para implementar los algoritmos genéticos nos hemos aprovechado de la POO de java para crear un código modular que sea escalable de cara a futuras prácticas.Los principales elementos de nuestro código son los siguientes.

- Para representar los individuos nos hemos decantado por un esquema donde Individuo<T> es la clase abstract de la cual heredan todos los demás individuos. Esta clase es extendida por otras dos clases abstractas que son IndividuoBinario(donde T=Boolean y que es extendida por los individuos de las funciones 1,2,3 y 4) y IndividuoReal(donde T=Double y que es extendida por los individuo de la función 5).
- Para las poblaciones hemos creado una clase abstracta llamada TPoblación que será extendida por 5 poblaciones que corresponden a las 5 funciones.
- Para los métodos de selección hemos creado una clase abstracta Selector que es extendida por las clases SelectorEstocastico , SelectorRestos ,SelectorRuleta, SelectorTorneoDeterministico, SelectorTorneoProbabilistico y SelectorTruncamiento.
- Para los métodos de cruce hemos creado una clase abstracta Cruzador que es extendida por las clases CruzadorAritmetico, CruzadorBLXa, CruzadorMonoPunto y CruzadorUniforme.
- Para los métodos de mutación hemos creado una clase abstracta Mutador que es extendida por la clase MutadorBasico. Esto parece excesivo para que solo sea extendido por una clase pero está hecho en vista de futuros métodos de mutación.
- Para crear los métodos de selección,cruce,mutación y población
- Para el esquema del algoritmo genético hemos creado una clase que implementa el esquema general del algoritmo genetico.Al solo hacer uso de las clases abstractas el mismo esquema funciona en todas las funciones con todos los métodos de selección,cruce o mutación.
- Para pasar los parámetros desde el controlador al algoritmo genético hemos creado una clase llamada TParametros la cual tiene los atributos necesarios para que el algoritmo pueda ser ejecutado represente lo ocurrido durante las generaciones.
- Para devolver los resultados del algoritmo genético hemos creado una clase llamada TResultStatistics la cual tiene los atributos necesarios para que la vista represente lo ocurrido durante las generaciones.
- Para la comunicación entre presentación y el algoritmo hemos creado la clase Controller la cual recibirá los parámetros seleccionados por el usuario y se encargará de llamar a las factorías para crear los recursos

necesarios para el algoritmo.Una vez creados el controlador le pasara los recursos en forma de TParametros al algoritmo y le mandara los TResultStatistics generados por el algoritmo a la vista.
- Para la interfaz de usuario hemos creado una clase MainWindow que será la encargada de recoger los datos del usuario, pasarlos al controlador y representar en la gráfica los datos que devuelva el controlador

Reparto de tareas

Para la realización del proyecto no ha habido una división exacta de las tareas a realizar. Durante el tiempo trabajado hemos ido avanzando poco a poco en los puntos a realizar que iban surgiendo. Por ejemplo, ambos miembros han trabajado de forma significativa en el algoritmo genético. Si tuviésemos que hacer una aproximada de las tareas podríamos dividirlo de la siguiente forma:

-Santiago González García

- Estructuración de las clases de forma que sea fácilmente reutilizable para futuras prácticas. Esto incluye la creación de factorías, tipos genéricos, interfaces, clases abstractas, y herencia entre clases.
- Desarrollo del algoritmo genético.
- Desarrollo de funciones de cruce y mutación.
- Desarrollo de la interfaz gráfica con WindowBuilder
- Corrección de errores
- Realización de la memoria

- Daniel Couto Seller

- Desarrollo de las funciones de selección.
- Desarrollo del algoritmo genético
- Desarrollo de funciones de cruce.
- Desarrollo de los individuos y poblaciones para las funciones.
- Desarrollo de la interfaz gráfica con WindowBuilder
- Corrección de errores
- Realización de la memoria