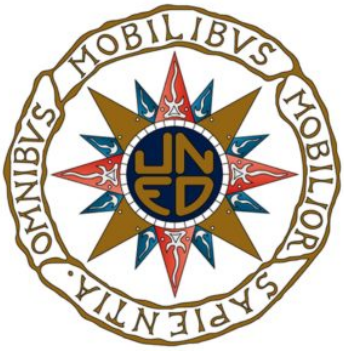
Máster en Investigación en Inteligencia Artificial

Computación evolutiva

Memoria Actividad 3

Autora:
Marta Rodríguez Sampayo



16 de abril de 2020

Índice

Índice de figuras

Índice de tablas

1. Descripción del problema a resolver

El objetivo de este trabajo consiste en el cálculo de la derivada simbólica de una función $f : X \subseteq \mathbb{R} \to \mathbb{R}$. En cuanto a la definición de derivada de una función en un punto, sea $X \subseteq \mathbb{R}$ un intervalo abierto, diremos que $f : X \to \mathbb{R}$ es derivable en $x_0 \in X$, denotado por $f'(x_0)$, si existe y es finito el límite:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Por otro lado, considerando la definición de derivada de una función en un intervalo, sea $X \subseteq \mathbb{R}$ un intervalo abierto, diremos que $f : X \to \mathbb{R}$ es derivable en $[a, b] \subset X$ si f es derivable en cada uno de los puntos de dicho intervalo, es decir:

$$f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}, \forall x \in [a, b]$$

El problema de calcular la derivada de la función $f(x)$ en el intervalo $[a, b]$, partiendo de que esta existe, se puede convertir en un problema de optimización consistente en encontrar una función $\hat{g}(x)$ que consiga minimizar la expresión:

$$\underbrace{\text{mín}}_{\hat{g}(x)} \frac{1}{(b-a)} \int_a^b error[(f'(x), \hat{g}(x))]dx$$

Una aproximación de este último problema, consiste en discretizar el intervalo de definición, de forma que la expresión anterior quedaría:

$$\underbrace{\text{mín}}_{\hat{g}(x)} \frac{1}{N+1} \sum_{i=0}^{N} error_i(f'(a + i \cdot \Delta x), \hat{g}(a + i \cdot \Delta x))$$

donde la anchura del subintervalo de muestreo para conseguir muestrear $N + 1$ puntos en el intervalo $[a, b]$, se define como $\Delta x = \frac{(b-a)}{N}$ y $f'(a + i \cdot \Delta x)$ viene dado por:

$$f'(a + i \cdot \Delta x) = \frac{f((a + i \cdot \Delta x) + h) - f(a + i \cdot \Delta x)}{h}, \forall i \in \{0, 1, \ldots, N\}$$

tomando h como un valor muy pequeño. En cuanto al cálculo del error, se utilizará el error absoluto ponderando cada sumando de este, la expresión exacta se presenta en la siguiente sección de esta memoria junto a una explicación con mayor detalle.

Abstrayéndonos de la finalidad última del problema, se debe tener en cuenta que no se trata de un simple problema de optimización numérica, pues el sistema desarrollado debe generar una expresión matemática válida, $\hat{g}(x)$, que minimice la función de error especificada.

Los experimentos se realizan sobre el siguiente conjunto de funciones:

1. $f(x) = x^3 + 8$ en $[0, 5]$ — Solución: $g(x) = 3x^2$
2. $f(x) = \frac{x-2}{x+2}$ en $[0, 5]$ — Solución: $g(x) = \frac{4}{(x+2)^2}$
3. $f(x) = \frac{1}{5}(x^2 + 1)(x - 1)$ en $[-2, 2]$ — Solución: $g(x) = \frac{1}{5}(3x^2 - 2x + 1)$
4. $f(x) = -e^{-2x^2+2}$ en $[0, 3]$ — Solución: $g(x) = 4xe^{-2x^2+2}$
5. $f(x) = \frac{e^{2x}+e^{-6x}}{2}$ en $[0, 2]$ — Solución: $g(x) = e^{2x} - 3e^{-6x}$
6. $f(x) = xln(1 + 2x)$ en $[0, 5]$ — Solución: $g(x) = ln(1 + 2x) + \frac{2x}{1+2x}$
7. $f(x) = e^{2x} \operatorname{sen} x$ en $[-2, 2]$ — Solución: $g(x) = e^{2x}(2 \operatorname{sen} x + \cos x)$

2. Método de resolución

En el campo de la computación evolutiva, la programación genética (*Genetic Programming* o GP) se caracteriza por la representación de los individuos de la población, que siguen una estructura de árbol en lugar de una cadena, como en un algoritmo genético. Esta característica resulta de especial utilidad cuando el problema a resolver radica en la construcción de una estructura óptima, por ejemplo, el código de un programa. Un subdominio particular de GP consiste en evolucionar expresiones matemáticas. En este caso, el programa evolucionado es una expresión matemática, ejecutarlo significa evaluar dicha expresión y su salida se correspondería con el valor de dicha expresión.

Se han propuesto diferentes variantes englobadas en la definición de programación genética [1], entre ellas destacamos la evolución gramatical (*Grammatical Evolution* o GE) [2]. A grandes rasgos, este método utiliza una forma Backus-Naur (BNF) para expresar el fenotipo de cada individuo, lo que permite expresarlo como una gramática. Esta gramática BNF consiste en símbolos terminales y no terminales que se re-escriben utilizando ciertas reglas de producción de expresiones. Cada individuo se representa como una cadena binaria de longitud variable que contiene la información necesaria para seleccionar una regla de producción de la gramática en sus codones (grupos de 8 bits). Una ventaja de GE es que permite diferenciar el espacio de búsqueda de soluciones de la decodificación de individuos, pudiendo utilizar otros algoritmos de búsqueda para encontrar una solución óptima sin tener en cuenta la complejidad de la estructura que la forma.

La evolución gramatical se ha utilizado para encontrar soluciones a varios problemas complejos de índole matemática, como la regresión simbólica o la resolución de ecuaciones diferenciales. Antes de entrar en detalle sobre aplicaciones concretas, es necesario hacer una breve introducción sobre los múltiples métodos existentes para calcular las derivadas de una función.

En relación con esta memoria, se puede destacar la diferenciación automática (*Automatic differentiation*) [3] que consiste en un conjunto de técnicas para evaluar numéricamente la derivada de una función especificada por un programa de ordenador. Está basado en la aplicación de la regla de la cadena a las operaciones aritméticas básicas que ejecuta un programa (suma, resta, multiplicación, división, etc.) y funciones elementales (exponencial, logarítmica, seno, coseno, etc.) para calcular derivadas de orden arbitrario de cualquier función.

Se han realizado varias aproximaciones utilizando diferentes técnicas para abordar el cálculo de derivadas a través de diferenciación automática [4]. Los métodos tradicionales de cálculo de derivadas de una función, diferenciación simbólica y numérica, pueden llevar a códigos ineficientes al tener que lidiar con la dificultad de convertir un programa en una única expresión e incluso introducir errores en el proceso de discretización, entre otros. Por tanto, la diferenciación automática que resuelve estos problemas, resulta más plausible como método de resolución.

En [5] se detalla un sistema de programación genética y diferenciación automática para la resolución de ecuaciones diferenciales. Los resultados de los experimentos llevados a cabo demuestran que utilizar GP para el cálculo de la derivada de una función resulta factible y más eficaz que otros métodos con los que se compara que no alcanzan soluciones óptimas para ciertos supuestos. Otro artículo más reciente plantea el uso concreto de evolución gramatical para la resolución de estas ecuaciones [6] consiguiendo resultados prometedores.

En conclusión, la cantidad de métodos existentes para resolver el cálculo de la derivada de una función es ingente, puesto que es un problema presente en diferentes campos. Evaluando la efectividad de las investigaciones relacionadas con el campo de la programación genética, se puede considerar esta y, en particular, la evolución gramatical como un método factible para alcanzar una solución válida al problema planteado manteniendo un compromiso razonable entre eficiencia y eficacia.

A continuación se presenta el método de resolución propuesto utilizando, GE junto a un algoritmo genético para llevar a cabo el proceso de búsqueda, incluyendo las características de este último y la

gramática BNF empleada para el proceso de decodificación.

2.1. Configuración del algoritmo de búsqueda

Como se ha presentado en la sección anterior, para llevar a cabo la búsqueda de una solución al problema planteado se utiliza un algoritmo genético. Cada individuo está representado por un cromosoma de longitud variable constituido por números enteros de 8 bits denominados codones. El proceso de búsqueda se efectúa utilizando esta representación y se evalúa a cada individuo tras decodificarlo utilizando la gramática BNF especificada.

En las siguientes subsecciones se presentan las diferentes variantes del algoritmo genético que se han implementado. Se han llevado a cabo diferentes experimentos variando la configuración de este algoritmo, la estructura con la que se alcanzan mejores resultados se presentará en la Sección 4.

2.1.1. Gramática

Aunque no forma parte del algoritmo de búsqueda en sí, la gramática BNF seleccionada tiene un papel clave a la hora de encontrar una solución, ya que se evalúa la expresión matemática que representa cada individuo, es decir, su fenotipo y este varía en función de la decodificación llevada a cabo. Se ha empleado la siguiente gramática:

$$N = \{expr, op, pre-op\}$$

$$T = \{+, -, *, /, sen, cos, exp, ln, x, 1{,}0, (,)\}$$

$$S = \{expr\}$$

Conjunto de reglas de producción (P):

$$< expr >::=< expr >< op >< expr > |(< expr >< op >< expr >)| < pre-op > (< expr >)| < var >$$

$$< op >::= +| - | * |/$$

$$< pre-op >::= sen|cos|exp|ln$$

$$< var >::= X|1{,}0$$

2.1.2. Función de evaluación

El objetivo de la función de evaluación empleada es calcular el error existente entre la derivada exacta de la función $f(x)$, $g(x)$, y la derivada obtenida evolutivamente, $\hat{g}(x)$. Para ello se usa el error absoluto y se pondera cada sumando de error, según la siguiente función:

$$F(x) = \frac{1}{N+1} \sum_{i=0}^{N} \omega_i \cdot |f'(a + i \cdot \Delta x) - \hat{g}(a + i \cdot \Delta x)|$$

$$\omega_i = \begin{cases} K_0 & \text{si } |f'(a + i \cdot \Delta x) - \hat{g}(a + i \cdot \Delta x)| \leq U \\ K_1 & \text{si } |f'(a + i \cdot \Delta x) - \hat{g}(a + i \cdot \Delta x)| > U \end{cases}$$

donde U define el umbral mínimo por debajo del cual el error i-ésimo es admisible (se alcanza un *hit*), con un valor de 10^{-1}; K_0 y K_1 definen, respectivamente, el valor de ponderación (penalización) cuando se alcanza o no un *hit*, siendo $K_0 = 1$ y $K_1 = 10$. Además, el valor de $f'(a + i \cdot \Delta x)$ se calcula utilizando la siguiente expresión:

$$f'(a + i \cdot \Delta x) = \frac{f((a + i \cdot \Delta x) + h) - f(a + i \cdot \Delta x)}{h}, \forall i \in \{0, 1, ..., N\}$$

donde $\Delta x = (b - a)/N$ es la anchura del subintervalo de muestreo para conseguir muestrear $N + 1$ puntos en el intercalo $[a, b]$ y $h = 10^{-5}$. El valor de N utilizado es 50.

A cada individuo se le asigna como valor de adaptación el resultado de aplicar la función de evaluación a su fenotipo, es decir la derivada $\hat{g}(x)$ que representa. Puesto que el objetivo de la búsqueda es minimizar el error entre la derivada obtenida y la derivada exacta de la función a la que se aplica, un individuo se considera mejor cuanto menor es su valor de adaptación. Se considera que se ha encontrado una solución cuando se alcanzan 51 *hits*, es decir, un *hit* en todos los puntos de muestreo.

2.1.3. Inicialización de la población

El tamaño de la población inicial se establece como parámetro y, al comienzo de cada ejecución, se genera el número de individuos especificado. Cada individuo se inicializa con una secuencia de codones de longitud aleatoria, comprendida entre 1 y un número máximo preestablecido como parámetro del algoritmo. Además, el valor inicial de cada codón también es aleatorio, comprendido entre 0 y 255.

2.1.4. Selección de padres

Los individuos a los que se aplicará el método de recombinación, denominados padres, para generar nuevos individuos (descendencia), se escogen empleando el método de selección por torneo. Este método consiste en seleccionar una muestra aleatoria de la población de tamaño k, valor predefinido, y escoger de estos k individuos al que mejor valor de adaptación tenga para cumplir el papel de padre. Como se explicará más adelante, de cada dos padres se obtienen dos hijos, por lo que se repetirá la selección por torneo hasta obtener tantos padres como número de descendientes se desee.

2.1.5. Operadores de variación

2.1.5.1 Mutación

El método de mutación empleado es *random resseting* y consiste en que con cierta probabilidad p_m se cambia un codón de la cadena por un número entero aleatorio en el rango [0, 255]. A la hora de implementarlo se genera un número aleatorio entre 0 y 1 para cada codón de la secuencia del individuo, si este número es menor que p_m, se cambia el codón correspondiente y se repite este proceso hasta recorrer el genoma del individuo.

2.1.5.2 Recombinación

Se han realizado experimentos con dos operadores de cruce, por un lado *one-point crossover* y, por otro, *uniform crossover*, ambos parten de dos padres para generar dos hijos. El primer método consiste en la división de del genoma de cada padre en dos y la creación de cada hijo uniendo una subsecuencia de un padre con la del otro, es decir, si el padre 1 se divide en S1 y S2 y el padre 2 en S3 y S4, el genoma del primer hijo sería el correspondiente a unir S1 y S3 y el del segundo hijo, S2 y S4. Para llevar a cabo la división de las secuencias de codones de ambos padres, se generan dos puntos de corte aleatorios.

Utilizar el método de cruce uniforme, partiendo de dos padres con el mismo número de codones, implica generar una cadena de números aleatorios de esta longitud, cada número se corresponde con un codón del genoma del hijo, si este número es inferior a 0.5 se toma el codón correspondiente al primer padre, en caso contrario, del segundo padre. El segundo hijo se crea de forma análoga, invirtiendo el rol de cada padre. Sin embargo, al trabajar con longitudes variables del genoma de los individuos, se ha realizado una pequeña adaptación de este método de forma que la longitud de la cadena de números aleatorios es la mayor de entre los genomas de ambos padres y, en caso de que se hayan utilizado ya todos los codones del padre de menor longitud y el número aleatorio indique que se debe heredar de este, simplemente se ignora y se pasa al siguiente codón. Un ejemplo de esta variación se muestra en la Figura 1.

Ambos operadores se aplican en su caso con una probabilidad de cruce p_c dada. De esta forma, se genera un número aleatorio entre 0 y 1, si es menor que p_c, se aplica el operador a ambos padres y se generan dos hijos, en caso de ser mayor cada hijo será una copia de cada padre.

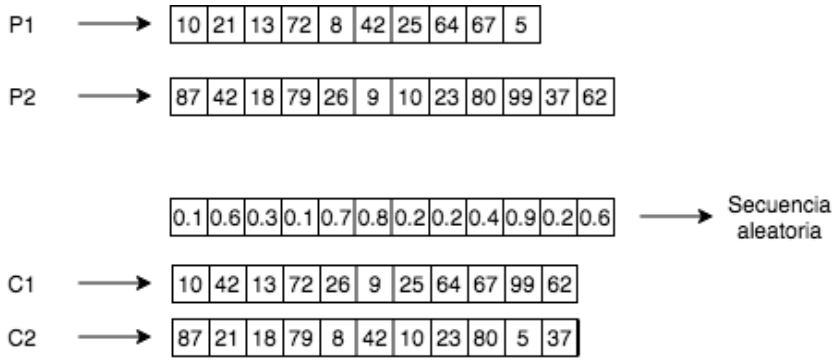


Figura 1: P1 y P2 representan las secuencias de codones de los padres, mientras que C1 y C2 representan a cada hijo generado. Al llegar a la posición 11, aplicando la lógica del operador (0,2 < 0,5) le correspondería a C1 un codón de P1, al haber terminado esta secuencia, se ignora esta posición, se coloca en C2 el codón de P2 correspondiente y se pasa a la siguiente posición, donde ocurre el efecto contrario y se ignora este valor para C2, mientras que a C1 le corresponde el último codón de P2.

2.1.5.3 Duplicación

En [7] se menciona un operador de variación a mayores de los tradicionales explicados anteriormente, denominado operador de duplicación. Puesto que su descripción es vaga, se ha consultado la sección de FAQ de la asignatura para completarla y proceder a su implementación. En base a esto, el efecto de este operador consiste en duplicar una secuencia de codones perteneciente al genoma de un individuo y añadirla justo antes del último codón de la cadena. Para llevar a cabo esta implementación, se generan dos números aleatorios:

- Longitud de la secuencia a duplicar: número aleatorio entre 1 y la longitud del genoma del individuo al que se le aplica el operador.
- Posición de inicio de la cadena a duplicar: número aleatorio entre 0 y la longitud del genoma del individuo menos el número de codones a duplicar.

Puesto que en otros artículos consultados sobre *Grammatical Evolution* no se utiliza este operador, la probabilidad de aplicación, p_d, escogida es baja. Al no especificarse en qué parte del ciclo evolutivo se debería aplicar, se ha decidido seguir las indicaciones de las FAQ nuevamente e introducirlo después de la aplicación del operador de mutación.

2.1.6. Selección de supervivientes

Se han utilizado tres estrategias de selección de supervivientes, en base a otras decisiones de implementación que se han ido tomando a lo largo del desarrollo y se explicarán posteriormente.

- Reemplazo generacional con elitismo: la descendencia generada es del mismo tamaño que la población anterior y la sustituye en cada generación. Puesto que se aplica elitismo, el miembro de la población con mejor valor de adaptación no es sustituido, salvo que algún miembro de las descendencia posea un mejor valor de adaptación.
- Reemplazo basado en valor de adaptación: se crea un número x de descendientes, menor al tamaño de la población, y estos sustituyen a los x individuos con peor valor de adaptación de la población anterior.
- Estrategia de reemplazo (μ, λ): se generan λ descendientes, siendo $\lambda > \mu$ y μ el tamaño de la población. En cada generación los mejores μ descendientes sustituyen a la población actual.

2.1.7. Manejo de restricciones

Puesto que se trata de un problema de optimización con restricciones (*Constraint optimization problem* o COP), se ha optado por mantener las restricciones y tratarlas explícitamente. A continuación, para cada restricción identificada, se define dicha restricción y el mecanismo de manejo empleado.

- Individuos inválidos (mapeo genotipo-fenotipo incompleto): aún utilizando la técnica de *wrapping* definida en [7], es posible que el genoma de un individuo no produzca una expresión válida de acuerdo a la gramática BNF utilizada. Este fenómeno ocurre cuando, habiendo utilizado *wrapping* el número máximo de veces especificado, no se completa el mapeo del genoma del individuo. En [8] se mencionan diferentes estrategias a seguir cuando esto ocurre. Se ha adoptado una estrategia de reparación de individuos (manejo directo de restricciones relacional, aplicando una función *decoder*) de forma determinista consistente en el establecimiento de una serie de reglas que convierten cada símbolo no terminal en un símbolo terminal válido. Las reglas utilizadas, según el símbolo no terminal a sustituir, son las siguientes:

  - $< expr >$: se trata como símbolo $' < var >'$ y se utiliza el primer codón de la secuencia para seleccionar el símbolo terminal correspondiente a esta regla de producción.
  - $< op >$: se utiliza el segundo codón de la cadena para determinar el símbolo terminal correspondiente a esta regla de producción.
  - $< pre - op >$: se utiliza el tercer codón de la cadena para determinar el el símbolo terminal correspondiente a esta regla de producción.
  - $< var >$: se utiliza el cuarto codón de la cadena para determinar el el símbolo terminal correspondiente a esta regla de producción.

  En los tres últimos casos si la longitud del genoma no es suficiente para utilizar el codón especificado, se utiliza el anterior. De esta forma se consigue un mapeo completo de todos los individuos a una expresión válida y se mantiene la restricción implícita de que al mapear un mismo genotipo siempre se obtenga el mismo fenotipo.

- Longitud del genoma de los individuos: al aplicar los distintos operadores de variación al genoma de un individuo es posible que la secuencia de codones exceda el máximo establecido. Para abordar este problema se ha aplicado una estrategia de manejo directo de restricciones preservando la factibilidad de las soluciones. Esta estrategia afecta a las siguientes etapas del algoritmo:

  - Inicialización de la población: tanto si se utiliza una longitud variable del genoma como fija, cada individuo se inicializa especificando su longitud inicial, nunca mayor que el máximo establecido.
  - Operadores de variación: en el diseño de estos operadores se tiene en cuenta que los nuevos individuos no deben superar la longitud máxima establecida, repitiendo el proceso de duplicación o cruce hasta que los individuos tengan una longitud adecuada.

- Construcción de expresiones matemáticas inválidas: al evaluar la expresión matemática resultante del fenotipo de un individuo, es posible que esta sea inválida (división por cero, logaritmo de un número negativo, etc.). En este caso, se utiliza manejo indirecto de restricciones, asignando a cada individuo que cumpla esta condición un valor de adaptación infinito.

2.1.8. Control de parámetros

Puesto que los valores de los distintos parámetros son los que definen el comportamiento del algoritmo genético, se han implementado diferentes mecanismos de adaptación y autoadaptación de algunos de ellos para mejorar la eficacia de la búsqueda de una solución. Las diferentes estrategias empleadas para cada parámetro son:

- Adaptación de las probabilidades de mutación y recombinación: se han implementado dos métodos de adaptación de parámetros básicos presentados en [9]. Ambos son deterministas y basados en el tiempo de búsqueda transcurrido. El primero consiste en un incremento lineal de la probabilidad de

mutación, partiendo de un valor bajo, al mismo tiempo que se decrementa la probabilidad de cruce, inicializada a un valor alto (ILM-DHC). Esta variación viene dada por las siguientes fórmulas:

$$p_m = \frac{LG}{Gn} \ , \ p_c = 1 - \frac{LG}{Gn}$$

donde LG es el número de generación actual y Gn el número máximo de generaciones que ejecuta el algoritmo. La segunda aproximación trata de disminuir la probabilidad de mutación y aumentar la probabilidad de cruce, intercambiando las fórmulas anteriormente presentadas (DHM-ILC).

- Adaptación de la probabilidad de mutación: otra aproximación determinista que se encuentra bastante en la literatura relacionada se presenta en [10] (ecuación 2). Esta estrategia presentada por Bäck y Schütz consiste en la aplicación de la siguiente fórmula:

$$p_m = \left(2 + \frac{n-2}{T-1} \cdot t\right)^{-1}$$

donde n es la longitud del genoma del individuo a mutar, t el número de la generación actual y T el número máximo de generaciones.

- Probabilidad de mutación autoadaptativa: en el mismo artículo en el que se define el mecanismo anterior, se presenta una estrategia de autoadaptación de la probabilidad de mutación, un poco más explicada en [11] donde se aplica a un caso particular. De esta forma, se asigna a cada individuo una probabilidad de mutación inicial y antes de aplicar el operador de mutación al genoma, se muta esta probabilidad según la siguiente fórmula:

$$p'_m = \left(1 + \frac{1-p_m}{p_m} \cdot e^{\gamma \cdot N(0,1)}\right)^{-1}$$

donde p_m es la probabilidad de mutación actual del individuo, γ una constante, normalmente 0.22, y N(0,1) un valor aleatorio de una distribución normal.

En el artículo consultado se recomienda, para una mayor eficacia del mecanismo autoadaptativo, utilizar un esquema de selección de supervivientes (μ, λ) y por ello se realizan experimentos aplicando estas estrategias en conjunto.

- Tamaño de población variable: ya que en el libro base de la asignatura se menciona que ciertos algoritmos pueden verse beneficiados al aplicar un tamaño de población variable se ha realizado una implementación sencilla de este mecanismo. En primer lugar, se consideró utilizar GAVaPS [12] ya que aparece mencionado en el libro, sin embargo al realizar algún experimento se comprobó que la población crecía rápidamente hasta alcanzar varios miles de individuos, perjudicando gravemente al rendimiento. Por ello, se optó por utilizar la aproximación planteada en [13], donde se comenta el inconveniente encontrado de GAVaPS y se parte de mejorar este mecanismo. Así, a cada individuo se el asigna un "tiempo de vida restantez en cada generación se decrementa este valor de cada individuo de la población, con la excepción de aquel que tenga el mejor valor de adaptación, cuando el tiempo de vida llega a 0, el individuo es eliminado de la población.

El cálculo del tiempo de vida inicial o RLT (*Remaining LifeTime*) se realiza según la siguiente expresión:

$$RLT(i) = \begin{cases} MinLT + \eta \cdot \frac{WorstFit - fitness(i)}{WorstFit - AvgFit} & \text{si } fitness(i) \geq AvgFit \\ \frac{1}{2}(MinLT + MaxLT) + \eta \cdot \frac{AvgFit - fitness(i)}{AvgFit - BestFit} & \text{si } fitness(i) < AvgFit \end{cases}$$

donde $\eta = \frac{1}{2}(MaxLT - MinLT)$ y los límites inferior y superior del tiempo de vida son respectivamente $MinLT = 1$ y $MaxLT = 11$. Además, se tienen en cuenta el peor valor de fitness de la población actual ($Worstfit$), el mejor ($BestFit$), el fitness medio ($AvgFit$) y el fitness del propio

individuo ($fitness(i)$), para favorecer a los individuos con mejor valor de fitness.

Puesto que se recomienda utilizar un modelo de selección de supervivientes *steady-state*, se implementa esta estrategia junto al reemplazo basado en el valor de adaptación explicado anteriormente.

El valor del resto de parámetros se establece de forma estática en base a una serie de experimentos realizados que se explicarán posteriormente junto a los efectos de las estrategias que se presentan en este apartado.

2.1.9. Mecanismo de búsqueda local

Para aumentar la eficiencia de la búsqueda, se implementa un algoritmo de búsqueda local consistente en el cálculo de la vecindad del individuo con mejor valor de adaptación de la descendencia y la búsqueda de un individuo mejor adaptado entre esta.

Se ha considerado como vecino de un individuo a aquel que difiera en un codón de este. Por tanto, para calcular el conjunto de vecinos se ha optado por aplicar *random resseting* a cada codón del genoma del individuo, en caso de que el nuevo individuo tenga un fitness mejor, se termina la búsqueda y este sustituye al primero en la población.

Específicamente, las características principales de este mecanismo de búsqueda local son:

- Regla pivote: ascenso voraz o *greedy ascent*.
- Profundidad de la búsqueda: una sola iteración (en el peor caso, cálculo y evaluación de toda la vecindad).
- Función de vecindad: mutación de un codón por cada vecino utilizando *random resetting*.
- Algoritmo memético *lamarckiano*.
- Aplicado antes del proceso de selección de supervivientes (tras la aplicación de los operadores de variación).

3. Resultados

La evaluación de los experimentos se realiza utilizando las siguientes métricas:

- *Success Rate (SR)*: porcentaje de ejecuciones terminadas con éxito.

$$SR = \frac{\text{Número de ejecuciones exitosas}}{\text{Número total de ejecuciones}} \cdot 100$$

- *Mean Best Fitness (MBF)*: media de los mejores valores de la función objetivo obtenidos al acabar cada ejecución.

$$MBF = \frac{1}{N} \sum_{i=1}^{N} BF_i$$

  donde N es el número de ejecuciones y BF_i el mejor valor obtenido en cada ejecución.

- *Average number of Evaluations to a Solution (AES)*: el número medio de evaluaciones hasta una solución. Calculado sobre el número de ejecuciones que han resultado en éxito, no sobre el total.

$$AES = \frac{\mu}{N'} \sum_{i=1}^{N'} G_i$$

  donde μ representa el tamaño de la población (sobre la que se realizan las evaluaciones), N' el número de ejecuciones exitosas y G_i el número de generaciones necesarias para alcanzar la solución

en cada ejecución.

En caso de implementar el mecanismo de búsqueda local, la fórmula se modifica de la siguiente forma:

$$AES = \frac{1}{N'}(LE + \mu \cdot \sum_{i=1}^{N'} G_i)$$

donde LE representa el número total de evaluaciones locales realizadas en caso de alcanzar una solución.

Cuando el método de selección de supervivientes empleado evalúa los descendientes y no la población, como en el caso de selección (μ, λ), se utiliza el tamaño del conjunto de la descendencia para calcular esta medida.

- Tiempo de ejecución: la media (sobre todas las ejecuciones) del tiempo que tarda en ejecutarse una única vez el algoritmo completo (hasta finalizar el número de generaciones configurado), en segundos.
- Gráficas de progreso: representan la variación del valor de adaptación (valor de la función objetivo) a medida que aumenta el número de generaciones. Con el objetivo de suavizar estas curvas y permitir una mejor representación, se muestra el mejor fitness medio para cada generación en cierto número de ejecuciones. Para una mejor visualización de estas gráficas, en el eje Y (correspondiente a los valores de adaptación) se utiliza una escala logarítmica.
- Representación gráfica del resultado: al disponer de la derivada exacta de cada función, se representa la solución obtenida por el algoritmo junto a la representación gráfica de la derivada, para evaluar visualmente la precisión del sistema.

Tanto para el cálculo del SR como del AES, se debe definir qué se considera "éxito". En este caso, se considera que se ha realizado una ejecución exitosa si la función obtenida consigue 51 *hits*, es decir, un *hit* en cada punto de muestreo. En el caso de que no se alcance una solución, se representará gráficamente el resultado con mayor número de *hits* junto a la derivada exacta.

Con el objetivo de detallar el estudio del comportamiento del algoritmo según las diferentes configuraciones posibles implementadas, esta sección sigue la siguiente estructura:

- En el apartado 3.1 se presentan en primer lugar los resultados de los experimentos realizados para obtener valores óptimos de los parámetros fijos, seguido de los resultados obtenidos añadiendo o modificando el modelo de algoritmo genético (autoadaptación, búsqueda local, modificación de operadores de variación, etc.). En todos los experimentos realizados se aplican los mecanismos de manejo de restricciones definidos anteriormente.
- En el apartado 3.2 se muestran las representaciones gráficas del mejor resultado obtenido para cada función con la configuración óptima implementada del algoritmo junto a sus gráficas de progreso.

Para todos los experimentos realizados se emplea el método de mutación *random resetting* y el método de cruce *one-point crossover* salvo que se indique lo contrario, el resto de métodos utilizados y valores de los parámetros se indicarán en las tablas correspondientes.

3.1. Adaptación de las diferentes configuraciones

3.1.1. Parámetros algoritmo genético

Función	Tamaño población	SR	MBF	AES	Tiempo (s)
F1	100	90 %	$7.51 \cdot 10^{-5}$	6930.67	32.44
	200	90 %	$7.51 \cdot 10^{-5}$	5009	60.74
	300	100 %	$7.499 \cdot 10^{-5}$	13508.7	90.75
F2	100	90 %	0.028	6003.89	46.82
	200	100 %	0.0179	7054.7	72.56
	300	100 %	0.01404	9075.6	105.64
F3	100	0 %	2.2817	0	46.82
	200	0 %	2.225	0	92.49
	300	0 %	2.1827	0	141.09
F4	100	0 %	9.3224	0	51.47
	200	0 %	7.7997	0	92.64
	300	0 %	7.5758	0	127.99
F5	100	0 %	2.7574	0	56.90
	200	0 %	2.6703	0	102.22
	300	0 %	3.3831	0	146.11
F6	100	0 %	1.5396	0	48.20
	200	0 %	1.2914	0	92.28
	300	0 %	1.4056	0	127.09
F7	100	0 %	16.6269	0	62.94
	200	0 %	14.4024	0	106.0794
	300	0 %	14.2966	0	133.17

Tabla 1: **Tamaño de población**. Ejecuciones: 10; Generaciones: 200; Tamaño de torneo: 3; Tamaño de descendencia: igual al tamaño de población utilizado; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo

Función	Generaciones	SR	MBF	AES	Tiempo (s)
F1	200	90 %	$7.51{\cdot}10^{-5}$	5009	60.74
	500	100 %	$7.499{\cdot}10^{-5}$	15249	165.22
	1000	100 %	$7.499{\cdot}10^{-5}$	14648.1	308
F2	200	100 %	0.0179	7054.7	72.56
	500	100 %	0.0133	4595.4	198.28
	1000	100 %	0.0098	6633.6	364.92
F3	200	0 %	2.225	0	92.49
	500	0 %	1.8213	0	236.55
	1000	0 %	2.3226	0	429.67
F4	200	0 %	7.7997	0	92.64
	500	0 %	5.6244	0	226.08
	1000	0 %	5.3312	0	465.29
F5	200	0 %	2.6703	0	102.22
	500	0 %	2.5249	0	265.92
	1000	0 %	2.3180	0	540.569
F6	200	0 %	1.2914	0	92.28
	500	0 %	1.3269	0	318.66
	1000	10 %	0.8990	78020	468.67
F7	200	0 %	14.4024	0	106.08
	500	0 %	12.5522	0	266.905
	1000	0 %	11.0871	0	564.22

Tabla 2: **Número de generaciones**. Ejecuciones: 10; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo

Función	Probabilidad de mutación	SR	MBF	AES	Tiempo (s)
F1	0.01	40 %	23.4494	5463.25	75.17
	0.05	80 %	15.0201	5083.125	61.32
	0.1	90 %	$7{,}51 \cdot 10^{-5}$	5009	60.74
F2	0.01	70 %	0.0855	2331.14	130.72
	0.05	90 %	0.03182	3145.88	136.07
	0.1	100 %	0.0179	7054.7	72.56
F3	0.01	0 %	2.5609	0	116.53
	0.05	0 %	2.2274	0	112.57
	0.1	0 %	2.225	0	92.49
F4	0.01	0 %	8.0523	0	160.33
	0.05	0 %	8.8348	0	116.49
	0.1	0 %	7.7997	0	92.64
F5	0.01	0 %	9.2350	0	116.30
	0.05	0 %	3.6549	0	108.96
	0.1	0 %	2.6703	0	102.22
F6	0.01	0 %	1.9891	0	179.09
	0.05	0 %	1.1828	0	133.39
	0.1	0 %	1.2914	0	92.28
F7	0.01	0 %	17.7251	0	161.19
	0.05	0 %	11.5824	0	143.77
	0.1	0 %	14.4024	0	106.08

Tabla 3: **Probabilidad de mutación**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Selección de supervivientes: generacional con elitismo

Función	Probabilidad de cruce	SR	MBF	AES	Tiempo (s)
F1	0.5	90 %	$7{,}51 \cdot 10^{-5}$	5009	60.74
	0.7	70 %	7.3117	18068.71	61.05
	0.9	40 %	19.5881	7556.5	61.04
F2	0.5	100 %	0.0179	7054.7	72.56
	0.7	100 %	0.02378	9875.6	77.39
	0.9	100 %	0.01948	9956.2	68.84
F3	0.5	0 %	2.225	0	92.49
	0.7	0 %	2.0468	0	103.45
	0.9	0 %	1.8955	0	105.63
F4	0.5	0 %	7.7997	0	92.64
	0.7	0 %	8.3824	0	92.38
	0.9	0 %	7.4083	0	89.95
F5	0.5	0 %	2.6703	0	102.22
	0.7	0 %	3.0207	0	105.84
	0.9	0 %	9.0195	0	101.47
F6	0.5	0 %	1.2914	0	92.28
	0.7	10 %	1.3746	34605	85.34
	0.9	0 %	1.5332	0	83.98
F7	0.5	0 %	14.4024	0	106.08
	0.7	0 %	13.8071	0	98.50
	0.9	0 %	17.3603	0	92.89

Tabla 4: **Probabilidad de cruce**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo

Función	Tamaño de torneo	SR	MBF	AES	Tiempo (s)
F1	3	90 %	$7.51{\cdot}10^{-5}$	5009	60.74
	20	50 %	16.2316	15489.4	90.98
	100	20 %	13.1596	11813.5	110.52
F2	3	100 %	0.0179	7054.7	72.56
	20	80 %	0.11015	8113.875	112.47
	100	100 %	0.02286	6254.7	136.87
F3	3	0 %	2.225	0	92.49
	20	0 %	1.7777	0	126.29
	100	0 %	2.1493	0	103.73
F4	3	0 %	7.7997	0	92.64
	20	0 %	9.9655	0	142.99
	100	0 %	7.4776	0	153.9158
F5	3	0 %	2.6703	0	102.22
	20	0 %	4.2312	0	115.32
	100	0 %	6.7132	0	133.65
F6	3	0 %	1.2914	0	102.22
	20	0 %	1.5199	0	177.94
	100	10 %	1.1649	9420	152.22
F7	3	0 %	14.4024	0	106.8
	20	0 %	15.8872	0	142.35
	100	0 %	16.3750	0	150.82

Tabla 5: **Tamaño de torneo**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo

3.1.2. Parámetros decodificación

Función	Máx. wrapping	SR	MBF	AES	Tiempo (s)
F1	2	50 %	30.2292	11729.2	64.47
	3	100 %	$7{,}499 \cdot 10^{-5}$	10326.8	63.48
F2	2	100 %	0.01929	2952.8	73.88
	3	100 %	0.01897	4734.4	76.76
F3	2	0 %	1.6912	0	141.05
	3	0 %	2.2325	0	87.71
F4	2	0 %	7.0199	0	90.96
	3	0 %	7.2887	0	96.89
F5	2	0 %	2.5972	0	98.92
	3	0 %	2.886	0	105.97
F6	2	0 %	1.3348	0	90.06
	3	0 %	1.3747	0	88.80
F7	2	0 %	13.7648	0	108.87
	3	0 %	14.8986	0	107.31

Tabla 6: **Valor de *Wrapping***. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo; Búsqueda local

Función	Máxima longitud genoma	SR	MBF	AES	Tiempo (s)
F1	10	80 %	10.0169	12157	61.08
	20	100 %	$7.499{\cdot}10^{-5}$	12927	64.96
	Fija (20)	20 %	10.6346	22920	91.93
F2	10	60 %	0.1888	3641.5	3641.5
	20	100 %	0.01897	4734.4	76.76
	Fija (20)	100 %	0.01635	5177.4	88.38
F3	10	0 %	2.3978	0	99.78
	20	0 %	2.2325	0	87.71
	Fija (20)	0 %	1.7409	0	119.83
F4	10	0 %	12.2707	0	104.79
	20	0 %	7.2887	0	96.89
	Fija (20)	0 %	7.3797	0	99.13
F5	10	0 %	8.802	0	106.24
	20	0 %	2.886	0	105.67
	Fija (20)	0 %	4.1143	0	137.10
F6	10	0 %	1.9718	0	99.91
	20	0 %	1.3747	0	88.80
	Fija (20)	0 %	1.2983	0	88.64
F7	10	0 %	25.8016	0	105.11
	20	0 %	14.8986	0	107.31
	Fija (20)	0 %	14.8443	0	107.72

Tabla 7: **Longitud máxima de genoma**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 200; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo; Búsqueda local

3.1.3. Añadidos

Función	SR	MBF	AES	Tiempo (s)
F1	50 %	31.2388	21371.6	74.24
F2	90 %	0.14773	5966.22	81.80
F3	0 %	2.1096	0	100.05
F4	0 %	10.1190	0	94.56
F5	0 %	13.0484	0	112.49
F6	0 %	1.6485	0	113.04
F7	0 %	22.117	0	106.47

(a) **Uniform crossover**.

Función	SR	MBF	AES	Tiempo (s)
F1	100 %	$7.499{\cdot}10^{-5}$	12927	64.96
F2	100 %	0.0174	6254.9	78.49
F3	0 %	2.2766	0	94.75
F4	0 %	7.6221	0	97.74
F5	0 %	2.9650	0	115.069
F6	0 %	1.5072	0	96.37
F7	0 %	15.5181	0	109.19

(b) **Duplicación**. Probabilidad: 0.01

Tabla 8: **Operadores de variación**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo; Búsqueda local

Función	SR	MBF	AES	Tiempo (s)
F1	100 %	$7.499{\cdot}10^{-5}$	10326.8	63.48
F2	100 %	0.01897	4734.4	76.76
F3	0 %	2.2325	0	87.71
F4	0 %	7.2887	0	96.89
F5	0 %	2.886	0	105.67
F6	0 %	1.3747	0	88.80
F7	0 %	14.8986	0	107.31

Tabla 9: **Mecanismo de búsqueda local**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 200; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo

Función	SR	MBF	AES	Tiempo (s)
F1	70 %	5.3867	11152.43	38.31
F2	90 %	0.0405	6424.44	56.48
F3	0 %	2.1575	0	50.49
F4	0 %	9.2801	0	66.26
F5	0 %	3.6771	0	56.01
F6	0 %	1.3506	0	71.49
F7	0 %	14.6135	0	66.18

(a) **Tamaño de población variable**. Tamaño inicial de población: 200; Tamaño de descendencia: 100; Selección de supervivientes: basada en *fitness*; Probabilidad de mutación: 0.1

Función	SR	MBF	AES	Tiempo (s)
F1	40 %	4.0393	5707	182.13
F2	90 %	0.0279	8217	170.73
F3	0 %	2.1143	0	176.65
F4	0 %	8.5510	0	222.51
F5	0 %	3.1461	0	183.02
F6	0 %	1.4957	0	237.94
F7	0 %	16.0994	0	190.68

(b) **Probabilidad de mutación autoadaptativa**. Probabilidad de mutación inicial:1/longitud del genoma del individuo; Selección de supervivientes: (μ, λ); μ: 100; λ: 300.

Tabla 10: **Mecanismos de autoadaptación**. Ejecuciones: 10; Generaciones: 200; Tamaño de torneo: 3; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Búsqueda local

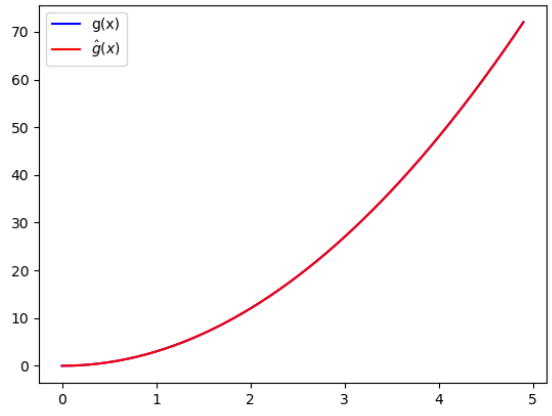
Función	Adaptación	SR	MBF	AES	Tiempo (s)
F1	DHM-ILC	90 %	1.4882	18541.78	29.19
	ILM-DHC	70 %	1.2537	6121.28	33.31
	Bäck+Schütz	80 %	1.6526	13385.5	38.65
F2	DHM-ILC	100 %	0.0188	13031.4	36.77
	ILM-DHC	100 %	0.0228	4812.9	42.45
	Bäck+Schütz	100 %	0.01901	8955.9	51.69
F3	DHM-ILC	0 %	2.3847	0	42.24
	ILM-DHC	0 %	2.2298	0	44.01
	Bäck+Schütz	0 %	2.1658	0	57.14
F4	DHM-ILC	0 %	7.9534	0	46.04
	ILM-DHC	0 %	9.7503	0	54.55
	Bäck+Schütz	0 %	7.1345	0	63.19
F5	DHM-ILC	0 %	7.5939	0	47.81
	ILM-DHC	0 %	2.5338	0	51.43
	Bäck+Schütz	0 %	2.9877	0	56.61
F6	DHM-ILC	0 %	1.5308	0	52.04
	ILM-DHC	0 %	1.7387	0	48.42
	Bäck+Schütz	0 %	1.5669	0	71.87
F7	DHM-ILC	0 %	17.4236	0	53.98
	ILM-DHC	0 %	14.9654	0	55.78
	Bäck+Schütz	0 %	13.9967	0	65.75

Tabla 11: **Mecanismos de adaptación de probabilidades de mutación y/o cruce**. Ejecuciones: 10; Generaciones: 200; Tamaño de población: 200; Tamaño de torneo: 3; Tamaño de descendencia: 100; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Selección de supervivientes: basada en *fitness*; Búsqueda local

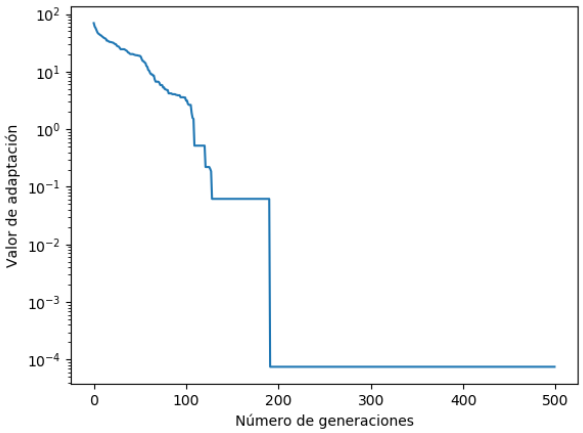
Función	SR	MBF	AES	Tiempo (s)	Hits
F1	100 %	$7{,}499 \cdot 10^{-5}$	16753.92	224.54	51
F2	100 %	0.0108	4773.066	245.20	51
F3	0 %	2.2999	0	319.43	28
F4	0 %	6.2638	0	326.93	34
F5	0 %	2.5333	0	367.49	43
F6	3 %	1.1312	102916	330.86	51
F7	0 %	12.168	0	397.91	17

Tabla 12: **Configuración final**. Ejecuciones: 30; Generaciones: 500; Tamaño de población: 300; Tamaño de torneo: 3; Tamaño de descendencia: 300; Longitud máxima de genoma: 20; Valor de *wrapping*: 3; Probabilidad de cruce: 0.5; Probabilidad de mutación: 0.1; Selección de supervivientes: generacional con elitismo

3.2. Representaciones gráficas de los resultados

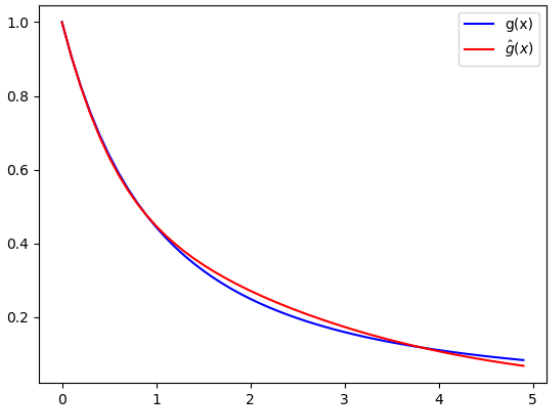


(a) $g(x) = 3x^2$
$\hat{g}(x) = (x * x) + (x * x) + (x * x)$

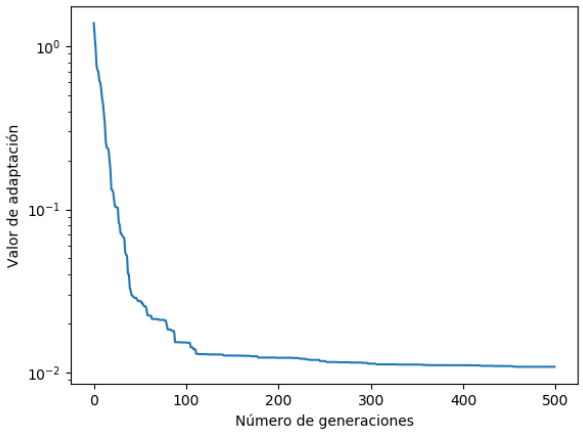


(b) Gráfica de progreso

Figura 2: Función 1

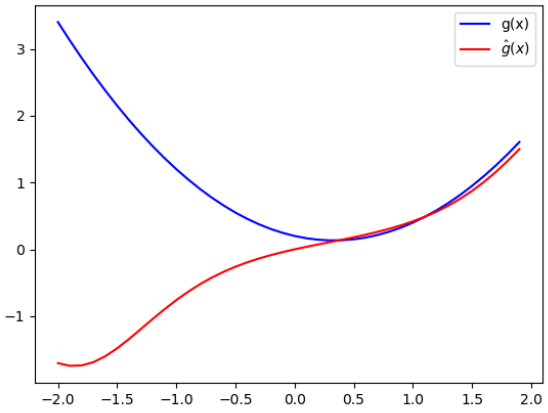


(a) $g(x) = \frac{4}{(x+2)^2}$
$\hat{g}(x) = 1{,}0/e^{xcos(1{,}0-sen(1{,}0/e^{xcos(1{,}0-sen(1{,}0))}))}$

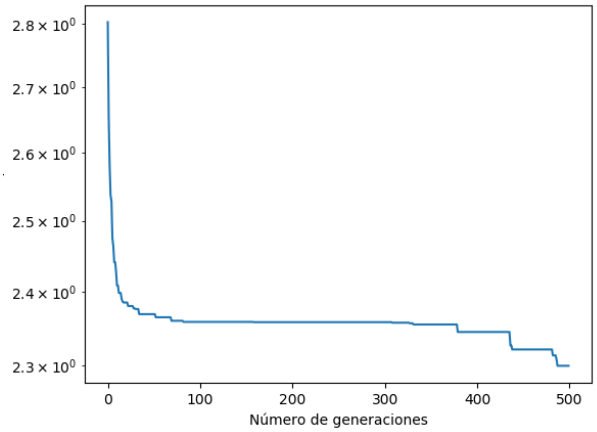


(b) Gráfica de progreso

Figura 3: Función 2

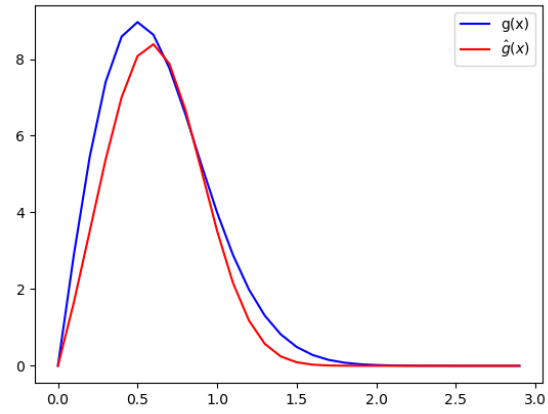


(a) $g(x) = \frac{1}{5}(3x^2 - 2x + 1)$
$\hat{g}(x) = sen(e^{cos(e^{1,0}+x)}) \cdot x$

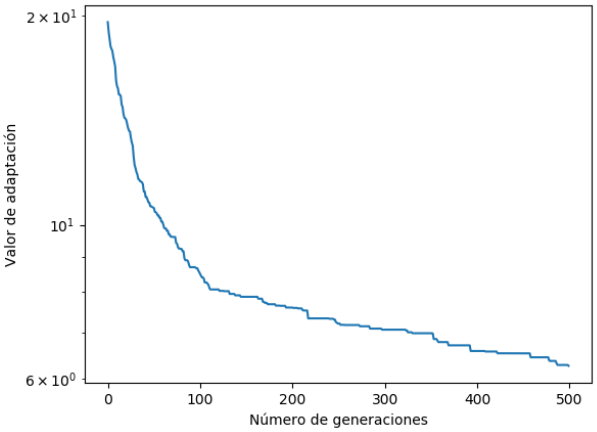


(b) Gráfica de progreso

Figura 4: Función 3

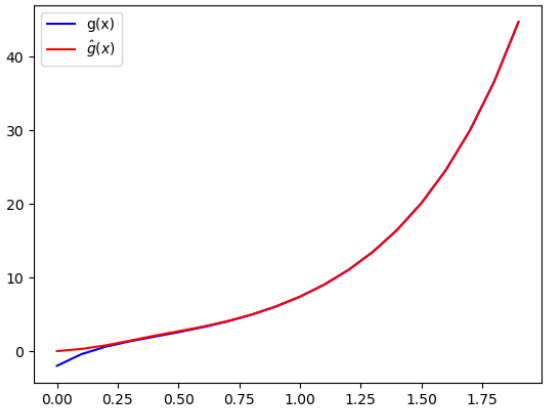


(a) $g(x) = 4xe^{-2x^2+2}$
$\hat{g}(x) = ee^{1,0} - (x - cos(x) + x \cdot x) \cdot x \cdot x$

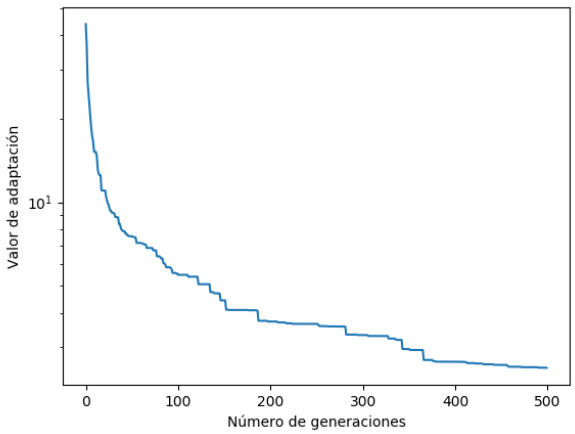


(b) Gráfica de progreso

Figura 5: Función 4

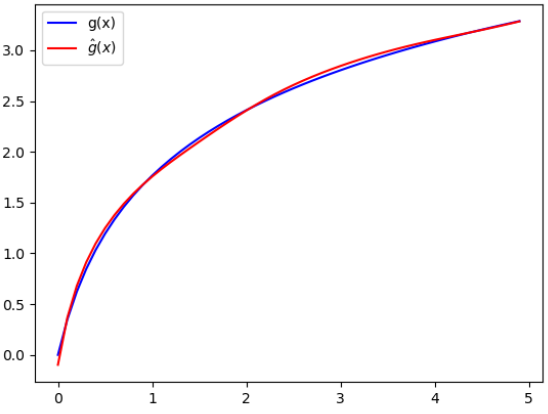


(a) $g(x) = e^{2x} - 3e^{-6x}$
$\hat{g}(x) = e^x \cdot e^x - e^{e^x \cdot e^x - e^{x+x+x+x}+x+x}$

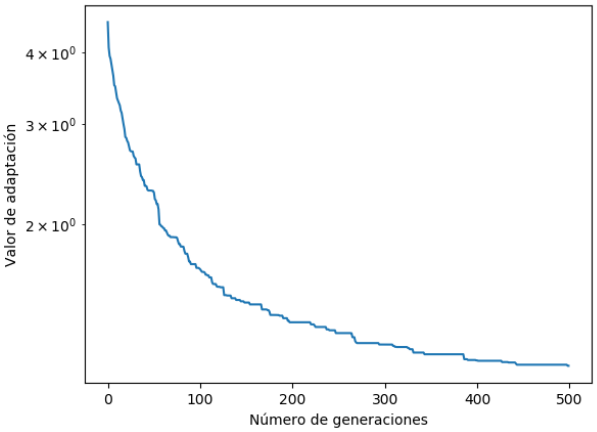


(b) Gráfica de progreso

Figura 6: Función 5

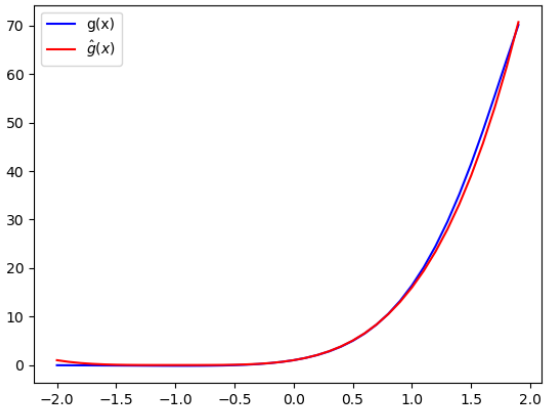


(a) $g(x) = ln(1 + 2x) + \frac{2x}{1+2x}$
$\hat{g}(x) = ln(x + 1 - cos(sen(cos(x \cdot 1))) + x) + 1$

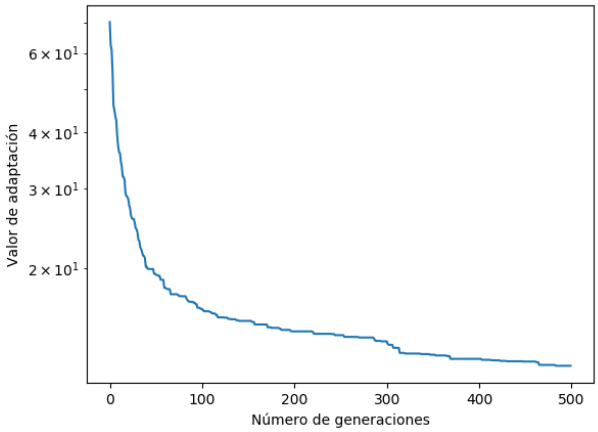


(b) Gráfica de progreso

Figura 7: Función 6



(a) $g(x) = e^{2x}(2\,\mathrm{sen}\,x + \cos x)$
$\hat{g}(x) = x^4 + 4x^3 + 6x^2 + 4x + 1$



(b) Gráfica de progreso

Figura 8: Función 7. Se ha simplificado la expresión de $\hat{g}(x)$ debido a su excesiva longitud.

4. Análisis y comparación de resultados

4.1. Evaluación de resultados numéricos

En primer lugar, se debe tener en cuenta que cada una de las funciones de las que se desea obtener la derivada tiene un comportamiento diferente y en ningún caso se comparan los resultados obtenidos en cada una entre ellos. Por otro lado, a la hora de escoger la mejor configuración se han tenido en cuenta los valores de los parámetros y las modificaciones de la estructura general del algoritmo que mejoraban los resultados en la mayor parte de las funciones, es decir, en al menos cuatro de las siete. Puesto que en la mayoría de funciones no se han alcanzado éxitos (51 *hits*), las métricas principales que se han utilizado de guía son el MBF y el tiempo medio de ejecución. El tercer criterio utilizado ha sido la eficiencia de la implementación, intentando encontrar un compromiso entre tiempo de ejecución y calidad de la solución encontrada.

Los primeros experimentos se realizan con el objetivo de encontrar los valores que optimicen los parámetros del algoritmo genético. Para ello se parte de una configuración escogida en base a las soluciones propuestas en los artículos comentados en la Sección 2 y se realizan experimentos variando los valores de un determinado parámetro.

De esta forma, en la Tabla 1 se puede apreciar que, para todas las funciones, disminuye el valor de MBF al aumentar el tamaño de la población, pero, simultáneamente, aumenta el tiempo medio de ejecución, llegando casi a duplicarse. Esto quiere decir que un mayor número de individuos permite conseguir mejores soluciones a costa de un mayor tiempo de procesado. En el caso de este parámetro, los resultados obtenidos en las diferentes funciones mantienen el mismo comportamiento. Se escoge como tamaño de población adecuado 200 individuos y se parte de estos resultados para comparar con otras evaluaciones.

El siguiente parámetro considerado es el número de generaciones durante las que se ejecuta el algoritmo, cuya evaluación se presenta en la Tabla 2. El efecto de aumentar su valor es similar al producido con el aumento del tamaño de población, sin embargo, el tiempo de ejecución se incrementa en una proporción mayor mientras que los cambios en el mejor *fitness* medio no son destacables, llegando incluso a empeorar en algunas funciones. Por ello y para mantener un tiempo de ejecución no demasiado elevado, se opta por ejecutar el algoritmo durante 200 generaciones.

En cuanto a las probabilidad de aplicación de los operadores de variación tradicionales (mutación

y cruce), se realizan modificaciones sobre los valores más utilizados en la literatura consultada para establecer un valor de referencia, aunque posteriormente se aplicarán mecanismos de adaptación y auto-adaptación. Así, en la Tabla 3 se muestra la evaluación de tres valores de probabilidad de mutación. Respecto al paradigma de GE se debe tener en consideración que aplicar mutación a un individuo puede resultar en una nueva cadena de codones que represente el mismo fenotipo que la inicial y, por tanto, su valor de adaptación no se vea afectado. Por ello las probabilidades utilizadas son relativamente bajas. En las dos primeras funciones aumenta la tasa de éxitos al utilizar el valor de 0.1 y en las cinco primeras el MBF disminuye; también es menor el tiempo medio de cada ejecución. Por ello se escoge este valor para dicha probabilidad. El análisis realizado a los distintos valores de la probabilidad de cruce (Tabla 4) es similar, salvo que en este caso es el menor valor probado, 0.5, el que mejora la eficacia de la búsqueda. Cabe destacar que para la sexta función se obtiene una tasa de éxito del 10 % con una probabilidad de 0.7, sin embargo, el MBF menor se sigue obteniendo con 0.5 y se considera efecto de la aleatoriedad inherente al algoritmo genético.

Otro parámetro relevante es el tamaño del torneo llevado a cabo para seleccionar a los padres en cada generación (Tabla 5). En este caso hay mayor diversidad entre los resultados obtenidos para cada función, pero la mayoría(F1, F2, F5 y F7) se ven beneficiadas, es decir su MBF es menor, utilizando 3 individuos en cada torneo.

Respecto a la decodificación, se consideran dos parámetros importantes, por un lado el número de veces que se realiza el *wrapping* y, por otro, la longitud máxima de la secuencia de codones[1]. Probablemente debido a que se emplea un mecanismo de reparación de individuos inválidos, el valor máximo de *wrapping* no parece afectar demasiado a los resultados obtenidos como se observa en la Tabla 6, ya que los valores de SR y MBF no sufren grandes variaciones (a excepción de la primera función). Se ha escogido el valor 3 para mantener la alta tasa de éxito obtenida en las dos primeras funciones. EN cuanto a la longitud de la secuencia de codones, se han llevado a cabo experimentos con longitud variable limitada a 10 codones y a 20 codones y, por otra parte, con una longitud fija de 20 codones. A la vista de la Tabla 7, se puede concluir que, permitir secuencias más largas beneficia el resultado final. Además, la longitud variable disminuye tanto el valor de AES de las dos primeras funciones como los tiempos medios de ejecución.

La última fase de ejecución de pruebas de evaluación consiste en la modificación de la estructura del algoritmo genético. Como referencia para este bloque, se toman los valores de la Tabla 9. En la Tabla 8a se muestra el efecto que produce cambiar el tipo de recombinación utilizada de *one-point crossover* a *uniform crossover*, claramente perjudicial tanto a nivel de calidad de la solución obtenida como en relación al tiempo de ejecución, por lo que se descarta su uso. Por otro lado, también relativo a los operadores de variación, se añade el operador de duplicación al proceso de búsqueda, con una probabilidad de 0.01. Sin embargo, aplicar duplicación no parece tener un efecto notable en los resultados presentados en la Tabla 8b, manteniéndose las tasas de éxito y subiendo mínimamente los valores tanto de MBF como de AES y tiempo de ejecución.

El mecanismo de búsqueda local se evalúa explícitamente en la Tabla 9, pero se implementa tanto en las tablas 6 y 7 como en 8a y 8b debido a que mejora la convergencia del algoritmo, aunque tampoco se consigue aumentar las tasas de éxito de las funciones 3 a 7 con esta estrategia.

Hasta este momento el método de selección de supervivientes empleado ha sido generacional con elitismo, sin embargo, en el caso de las Tablas 10a y 10b se utilizan respectivamente el método de reemplazo basado en *fitness* y la estrategia (μ, λ). En el primer caso, se emplea el mecanismo de auto-adaptación del tamaño de la población explicado al inicio de esta memoria. Al utilizar el método de selección de supervivientes basado en su valor de adaptación los tiempos de ejecución disminuyen considerablemente, pero solo se consigue un valor de MBF menor en 3 de las funciones tratadas y se disminuye la tasa de éxito de las dos primeras funciones. El segundo caso consiste en la auto-adaptación de la probabilidad de mutación, en el cual se obtienen resultados peores en cuanto a tiempo de ejecución y empeoramiento leve del MBF y SR de las funciones.

[1] Aunque la longitud de la representación de un individuo también forma parte de los parámetros del algoritmo genético, al afectar directamente a la velocidad de decodificación se ha incluido su valoración en este apartado.

Para finalizar la presentación de los resultado experimentales, en la Tabla 11 se muestra la evaluación de los tres modelos deterministas definidos para adaptar las probabilidades de mutación y cruce. Se mantiene el método de selección de supervivientes basado en *fitness* puesto que según la teoría beneficia este tipo de esquemas. Entre los tres métodos adaptativos no es posible escoger uno sobre el resto, puesto que su comportamiento varía de una función a otra, sin mostrar una mejoría constante y clara. En algunas de las funciones se mejoran los resultados anteriores, pero de nuevo esto puede ser debido a la aleatoriedad.

4.2. Análisis de las representaciones gráficas

En el apartado 3.2 se muestran los resultados gráficos correspondientes a la ejecución de la Tabla 12, donde se ha tomado el mejor resultado obtenido para cada función como $\hat{g}(x)$. Es necesario puntualizar que se entiende como mejor resultado aquel con valor de adaptación más bajo y no con mayor número de *hits*, esta distinción se explica en mayor profundidad en la Sección 5.

Para la primera función se obtiene la derivada exacta, como se puede observar en la Figura 2a las representaciones son idénticas. Su gráfica de progreso (Figura 2b) es bastante escalonada, indicando que la búsqueda se estanca durante cierto número de generaciones alrededor de un valor de *fitness* y este mejora bruscamente. Además, a partir de las 200 generaciones aproximadamente se alcanza la solución. Aunque el resto de gráficas de progreso presentan también este comportamiento escalonado, en ninguna se muestra tan acentuado. En el caso de la función 2, aunque la solución conseguida cumple la condición de éxito, no es idéntica a la derivada exacta, aunque la diferencia es mínima como se aprecia en la Figura 3a. En cuanto a la curva de progreso de la Figura 3b, el descenso del mejor valor de adaptación es pronunciado hasta llegar a las 100 generaciones, donde continúa disminuyendo de forma menos brusca. En estas dos primeras funciones se consigue la tasa de éxito más alta, llegando a valores de adaptación en ambos casos muy próximos a 0.

Las derivadas estimadas para las funciones 3 y 4 son las que más difieren de sus derivadas exactas en el intervalo considerado. En el caso de la tercera función, el valor de adaptación representado en la Figura 4b disminuye rápidamente en las primeras generaciones y se mantiene sobre 2.4, bajando progresivamente hasta 2.3, pero sin conseguir mejoras destacables. Este error se observa en la Figura 4a, donde la solución obtenida es prácticamente opuesta a la derivada real de la función en el intervalo [-2, 0.25], aunque a partir de este último punto comienza a asemejarse (se obtiene aproximadamente la mitad de los *hits* totales). En cuanto a la función 4, la forma de la derivada estimada por el algoritmo (Figura 5a) es muy similar a $g(x)$, sin embargo no coincide en la mayoría de puntos del intervalo, explicando el valor elevado del error calculado, presentado en la Figura 5b, y que solo se hayan obtenido 34 *hits*.

Los resultados de la función 5 son un ejemplo de que la tasa de éxito no sirve de indicador único del buen funcionamiento del sistema. Con 43 hits conseguidos de 51, la función $\hat{g}(x)$, representada en la Figura 7a, es idéntica a la derivada exacta, salvo en el intervalo aproximado [0,0.25]. Por otro lado, la Figura 6b muestra el mismo comportamiento escalonado y un descenso progresivo del valor de adaptación que podría indicar la necesidad de unas generaciones más para converger hacia un valor óptimo.

Puesto que de todas las ejecuciones realizadas sobre la sexta función, solo resulta exitosa una, en la Figura 7b, donde se muestran los valores medios del mejor valor de adaptación en cada generación, se presenta una curva más suave con un descenso menor del *fitness* durante las primeras generaciones. La derivada $\hat{g}(x)$ obtenida consigue los 51 *hits* con un error aproximado de 0.03 que explica las diferencias mínimas presentes en la Figura 7a entre ambas funciones.

La Figura 8a es quizás la que más se contradice en apariencia con los resultados numéricos obtenidos para la séptima función, ya que la representación gráfica de la mejor solución parece prácticamente igual a la derivada exacta y, sin embargo, solo son 17 los *hits* obtenidos. Por otro lado, los valores de *fitness* mostrados en la Figura 8b son los más elevados de entre todas las funciones, indicando un error mayor entre la solución real y la inferida por el algoritmo.

5. Conclusiones

En esta memoria se ha presentado un sistema basado en el paradigma de la programación genética, más concretamente en evolución gramatical, para resolver el problema de cálculo de la derivada exacta de una función en un intervalo dado. En la Sección 2, además de especificarse la estructura del método de resolución propuesto, se concluye que las características de un algoritmo de GE se adecúan al problema planteado, resultando una elección prometedora para llevar a cabo esta tarea.

Las diferentes modificaciones del algoritmo genético empleado como mecanismo de búsqueda que se han llevado a cabo se basan en los resultados obtenidos en la documentación consultada o en teorías planteadas en estos, de forma que deberían mejorar en algún aspecto el rendimiento y/o eficacia del sistema implementado. Sin embargo, en los resultados experimentales obtenidos al aplicar el algoritmo al conjunto de las siete funciones que forman el conjunto de estudio del problema (Sección 3.1), no se refleja la validez del método para encontrar una solución a cada problema. En contraposición, los resultados gráficos mostrados en la Sección 3.2, sí parecen apoyar la idea inicial de que es posible estimar la derivada de una función utilizando esta implementación, ya que las representaciones gráficas muestran mayor similitud entre solución óptima y estimación. Este último hecho lleva a considerar que quizás se haya sido demasiado estricto con la definición de éxito y que relajando un poco la condición para alcanzar un *hit*, se podría dar por buena una aproximación al óptimo global. Aún así, al presentarse los resultados gráficos junto a los valores de las métricas aplicadas se comprende mejor el comportamiento del algoritmo.

La configuración final elegida no incluye muchas de las opciones descritas, ya que no aportaban una mejora al diseño básico. Lejos de conseguir resultados óptimos, el enfoque aplicado presenta un desarrollo coherente, acercándose a soluciones reales a través del método evolutivo planteado. Cabe mencionar que tanto el mecanismo de búsqueda local como el método de reparación de individuos inválidos han mejorado los resultados de forma considerable. Otras líneas de actuación que quizás hubiesen mejorado los resultados serían la variación de las reglas que forman la gramática, añadiendo operaciones o variables; el estudio de un rango más amplio de valores para los parámetros del algoritmo o de otras estrategias de adaptación o autoadaptación; o la aplicación de otros métodos de mutación diferentes, por ejemplo. La gestión que se ha realizado del tiempo y las decisiones tomadas sobre la implementación no han permitido llevar a cabo estas medidas, aunque se han considerado.

Por último, comentar que el hecho de tratar con varias funciones simultáneamente dificulta el ajuste del algoritmo, puesto que se podrían conseguir resultados mucho mejores si se tratase cada función como un problema a parte, aunque este último enfoque se alejaría del objetivo inicialmente planteado de implementar un algoritmo que obtenga los mejores resultados en todas las funciones. Esta característica dificulta establecer si el sistema general se ve beneficiado o no aumentando la exploración, por ejemplo con un tamaño de torneo menor y una probabilidad de mutación mayor o aplicando más explotación, con una probabilidad de cruce menor. En general, se ha buscado mantener un equilibrio entre estas características, permitiendo un espacio de búsqueda suficientemente amplio y, simultáneamente, explorando las posibles soluciones en la vecindad del mejor individuo de la población.

6. Descripción del código

El lenguaje de programación empleado es *Python* y se estructura el código de la siguiente forma:

- Paquete *ge* (*Grammatical Evolution*):
  - *grammatical_evolution.py*: contiene la función principal donde se llama secuencialmente a las funciones de lectura de gramática, evaluación y búsqueda de solución, esta última durante el número de ejecuciones que se haya especificado.
  - *evaluation.py*: el cálculo de las métricas de evaluación y representaciones gráficas se realiza a través de esta clase.
  - *grammar.py*: en un objeto instancia de esta clase se realiza la lectura del archivo con la gramática BNF a utilizar, almacenando las reglas de producción y los tipos de símbolos. También se encuentra la función de decodificación de cada individuo, incluyendo la reparación de individuos inválidos en caso de utilizarse.
- Paquete *ga* (*Genetic Algorithm*):
  - *genetic_algorithm.py*: en este script se realizan las operaciones de control del proceso de búsqueda (inicialización de la población, definición del modelo de algoritmo genético según los parámetros y ejecución secuencial de cada etapa de la búsqueda en cada generación). También se encuentran las funciones de cálculo de tiempo de vida y disminución del mismo utilizadas cuando el tamaño de la población es variable.
  - *params.py*: en este archivo se definen como constantes los parámetros del algoritmo genético.
    - La función de la que se desea obtener su derivada, límites inferior y superior del intervalo y derivada exacta.
    - Número de generaciones y ejecuciones.
    - Tamaño de población, torneo, descendencia, genoma del individuo y máximo valor entero de cada codón.
    - Varios *booleanos* de control para determinar el uso de: tamaño de población variable, reparación de individuos inválidos, auto-adaptación, adaptación determinista y qué tipo, tipo de selección de supervivientes, búsqueda local, duplicación y tipo de recombinación.
    - En caso de utilizar tamaño variable de población, se especifican los límites inferior y superior del tiempo de vida y la constante η utilizada en su cálculo.
    - En cuanto a la decodificación del individuo se especifica también el número de veces que se realiza *wrapping*.
    - Probabilidades de aplicación de los operadores de variación.
  - *fitness.py*: en este archivo se encuentran las funciones de evaluación del valor de adaptación de cada individuo y cálculo del fitness medio de cada generación.
  - *individual.py*: cada individuo de la población es una instancia de la clase definida en este archivo. Se incluyen atributos para el genoma (cadena de enteros), valor de adaptación, fenotipo, número de hits, probabilidad de mutación y tiempo de vida (número de generaciones que permanece en la población).
  - *parent_selection.py*: contiene la función de selección por torneo que devuelve una lista con los individuos ganadores de cada torneo.
  - *recombination.py*: contiene las funciones necesarias para inicializar la probabilidad de cruce según el método escogido (adaptativo o fijo), una función para utilizar *one-point crossover* y otra para *uniform crossover*, ambas devuelven los dos nuevos individuos generados al aplicar la recombinación escogida a dos padres.
  - *mutation.py*: contiene las funciones necesarias para inicializar la probabilidad de mutación según el método escogido (auto-adaptativo, adaptativo o fijo), una función para mutar esta probabilidad en caso de utilizar auto-adaptación y la función que implementa el método de mutación *random resetting*.

- *duplication.py*: contiene la función que aplica el operador de duplicación a un individuo con la probabilidad especficada en los parámetros del algoritmo.
- *local_search.py*: contiene la función de búsqueda local, donde también se calculan el número de evaluaciones realizadas durante este proceso para el posterior cálculo del AES.
- *survivor_selection.py*: además de una función para cada esquema de selección de supervivientes (generacional con elitismo, basado en fitness y (μ, λ)), en este archivo se encuentra una función que comprueba el tiempo de vida de cada individuo de la población y lo elimina de esta si ha llegado a 0.

Para una correcta ejecución se debe modificar el archivo *params.py* para definir la configuración deseada y, posteriormente, ejecutar el script *grammatical_evolution.py*. Tras finalizar una ejecución completa se muestra por pantalla el resultado de dicha ejecución formado por: la secuencia de codones; la expresión matemática que representan; el valor de fitness; número de hits; probabilidad de mutación; tiempo de vida (en caso de no utilizar tamaño de población variable siempre es 0). Una vez finalizadas todas las ejecuciones establecidas, se muestra la gráfica de progreso y la representación de la primera solución obtenida y la derivada exacta de la función a la que se aplica el algoritmo.

Referencias

[1] E. Guogis and A. Misevičius, "Comparison of genetic programming, grammatical evolution and gene expression programming techniques," in *Information and Software Technologies*, G. Dregvaite and R. Damasevicius, Eds. Cham: Springer International Publishing, 2014, pp. 182–193.

[2] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[3] C. C. Margossian, "A review of automatic differentiation and its efficient implementation," *WIREs Data Mining and Knowledge Discovery*, vol. 9, no. 4, Mar 2019. [Online]. Available: http://dx.doi.org/10.1002/WIDM.1305

[4] M. Saroufim, "Automatic differentiation step by step," 2019. [Online]. Available: https://medium.com/@marksaroufim/automatic-differentiation-step-by-step-24240f97a6e6

[5] W. J. A. Lobão, D. M. Dias, and M. A. C. Pacheco, "Genetic programming and automatic differentiation algorithms applied to the solution of ordinary and partial differential equations," in *2016 IEEE Congress on Evolutionary Computation (CEC)*, 2016, pp. 5286–5292.

[6] I. Tsoulos and I. Lagaris, "Solving differential equations with grammatical evolution," 04 2020.

[7] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[8] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. Mcdermott, and A. Brabazon, "Geva: grammatical evolution in java," *ACM SIGEVOlution*, vol. 3, pp. 17–22, 07 2008.

[9] A. B. A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, "Choosing mutation and crossover ratios for genetic algorithms - a review with a new dynamic approach," *Information*, vol. 10, p. 390, 2019.

[10] T. Bäck and M. Schütz, "Intelligent mutation rate control in canonical genetic algorithms," in *Foundations of Intelligent Systems*, Z. W. Raś and M. Michalewicz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 158–167.

[11] R. Breukelaar and T. Bäck, "Self-adaptive mutation rates in genetic algorithm for inverse design of cellular automata," 01 2008, pp. 1101–1102.

[12] J. Arabas, Z. Michalewicz, and J. Mulawka, "Gavaps-a genetic algorithm with varying population size," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 1994, pp. 73–78 vol.1.

[13] T. Bäck, A. E. Eiben, and N. A. L. van der Vaart, "An emperical study on gas "without parameters"," in *Parallel Problem Solving from Nature PPSN VI*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 315–324.