

# Introducción a la Computación Evolutiva

## Tarea No. 1

Instructor: Dr. Carlos Artemio Coello Coello

Alumno: Ángel Alonso Galarza Chávez

16 de mayo de 2025

### Estrategias evolutivas

Los algoritmos de optimización evolutiva se caracterizaban por ser estrategias simples basadas en dos individuos, formalmente denominadas (1+1)-EE. En esta estrategia evolutiva, la representación de los individuos se realiza mediante vectores en un espacio continuo [1]. En cada iteración generacional, se genera un nuevo individuo (hijo) a partir de la adición de un número aleatorio, muestreada independientemente para cada componente del vector antecedente (padre). El individuo resultante es evaluado mediante una función de aptitud; si su valor de aptitud supera al del antecedente, este último es reemplazado, prosiguiendo así el proceso evolutivo. El esquema algorítmico del (1+1)-EE se explicita en el Algoritmo 1.

---

**Algorithm 1** Algoritmo Evolutivo Simple

---

$t$  = contador de generaciones,  $n$  = número de variables

$Gmax$  = número máximo de generaciones

$t \leftarrow 0$

Inicializar variables  $\bar{x}$

Evaluar  $f(\bar{x})$

**while**  $t \leq Gmax$  **do**

    inicializar semilla de aleatorios

    mutar el vector  $x_i$  usando:

$x'_i = x_i + \sigma[t] \times N_i(0, 1) \quad \forall i \in n$

    Evaluar  $f(\bar{x}')$

    Comparar  $\bar{x}$  con  $\bar{x}'$  y seleccionar el mejor

    Imprimir en un archivo los resultados

$t \leftarrow t + 1$

**if**  $(t \bmod n == 0)$  **then**

$$\sigma[t] = \begin{cases} \sigma[t - n]/c & \text{if } p_s > 1/5 \\ \sigma[t - n] \cdot c & \text{if } p_s < 1/5 \\ \sigma[t - n] & \text{if } p_s = 1/5 \end{cases}$$

**else**

$\sigma[t] = \sigma[t - 1]$

**end if**

**end while**

---

La primera fase de este trabajo consiste en la implementación del algoritmo (1+1)-EE utilizando el lenguaje de programación C/C++, con el objetivo de minimizar la función objetivo definida a continuación:

$$f(x_1, x_2) = \left(4 - 2,1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2 \quad (1)$$

La Figura 1 presenta la gráfica tridimensional de la función objetivo, generada con Octave. La visualización muestra la superficie en el dominio  $x_1 \in [-3, 3]$ ,  $x_2 \in [-2, 2]$ , donde se pueden observar claramente sus características topológicas, incluyendo los múltiples óptimos locales.

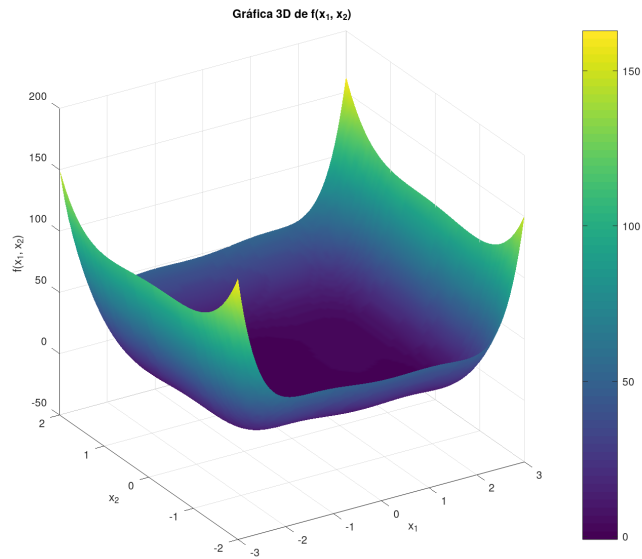


Figura 1: Grafica de la función 1

A continuación se presenta un análisis detallado del código fuente, organizado por componentes funcionales. La explicación se estructura en bloques lógicos que cubren cada aspecto fundamental de la implementación.

Al final de esta sección se incluye:

- El código fuente completo del algoritmo (1+1)-EE
- Instrucciones para compilar el programa
- Ejemplo de ejecución con parámetros típicos
- Resultados estadísticos de 20 ejecuciones independientes

La primera parte del código define las bibliotecas necesarias para su funcionamiento. Entre ellas destacan:

- `<ctime>` para la función `time(0)`, utilizada para generar una nueva semilla aleatoria en cada ejecución.
- `<fstream>` para almacenar los resultados en un archivo de texto.

Además, se incluyen las funciones auxiliares `initrandom` y `N`, proporcionadas en el material del curso Código proporcionado, que implementan:

- `N(0, 1)`: Genera números pseudoaleatorios con distribución normal (media cero y desviación estándar uno), cumpliendo con los requerimientos del problema.

```

1 #include <cstdlib>
2 #include <ctime>
3 #include <fstream>
4 #include <iostream>
5
6 // FUNCIONES UTILIZADAS DEL CODIGO PROPORCIONADO
7 void initrandom(int seed);
8 double N(double m, double sigma);

```

La función  $f$  representa la función objetivo a minimizar, previamente definida en la ecuación 1. Esta función recibe dos parámetros de entrada,  $x_1$  y  $x_2$ , y devuelve el valor resultante de evaluar la expresión matemática con estos argumentos.

```

1 // FUNCION A EVALUAR
2 float f(float x1, float x2) {
3     float x1_2 = x1 * x1;
4     float x2_2 = x2 * x2;
5     float x1_4 = x1_2 * x1_2;
6     return (4 - 2.1 * (x1 * x1) + (x1_4 / 3)) * x1_2 + x1 * x2 +
7           (-4 + 4 * (x2_2)) * x2_2;
8 }

```

Al inicio de la función principal, se solicita al usuario mediante línea de comandos el parámetro  $G_{\max}$ , que corresponde a un valor entero que define el número máximo de generaciones del algoritmo. Este parámetro debe cumplir con la siguiente especificación:

- $G_{\max}$  debe ser un valor entero mayor o igual a 10 ( $G_{\max} \geq 10$ )

Además de solicitar al usuario el número máximo de generaciones ( $G_{\max}$ ), el programa implementa la creación de un archivo de texto para almacenar los resultados de cada evaluación realizada durante la ejecución.

```

1 // VERIFICACION DEL ARGUMENTO
2 if (argc < 2) {
3     std::cout << "falta proporcionar un valor para Gmax" << std::endl;
4     return 1;
5 }
6 int gmax = std::atoi(argv[1]);
7
8 // GUARDAR LOS RESULTADOS DE LAS EVALUACIONES EN UN TXT
9 std::ofstream archivo("resultados.txt");
10 if (!archivo.is_open()) {
11     std::cerr << "Error al abrir el archivo." << std::endl;
12     return 1;
13 }

```

En esta sección se inicializan los parámetros fundamentales del algoritmo evolutivo (1+1)-EE. A continuación se detalla el propósito de cada variable:

**n\_vars** Número de variables de optimización en la función objetivo (ecuación 1)

**c** Factor de ajuste para la regla 1/5, fijado en 0.817  
**t** Contador de generaciones/iteraciones transcurridas  
**exito** Acumulador de mutaciones que mejoran la solución  
**vars\_x** Vector solución actual (individuo padre)  
**vars\_x\_** Vector solución mutada (individuo hijo)  
**ps** Tasa de éxito de mutaciones en el último bloque  
**fold** Valor actual de la función objetivo  
**fnew** Valor evaluado de la solución mutada  
**sigma** Magnitud del paso de mutación (distribución normal)

Los valores iniciales se establecen según las especificaciones del problema:

- Tamaño inicial de mutación:  $\sigma[0] = 3,0$
- Constante de ajuste: Se ha seleccionado  $c = 0,817$  de entre el rango recomendado  $[0,817, 1,0]$ .

```

1  // PARAMETROS
2  const int n_vars = 2;
3  const float c = 0.817;
4  int t = 0;
5  int exito = 0;
6
7  float vars_x[2] = {0.0, 0.0};
8  float vars_x_[2] = {0.0, 0.0};
9
10 float ps = 0.0;
11 float fold = 0;
12 float fnew = 0;
13
14 float *sigma = new float[gmax]();
15 sigma[0] = 3.0;
  
```

Después de la inicialización de las variables, se inicializan los valores de  $vars_x[0]$  y  $vars_x[1]$  que representan  $x_1$  y  $x_2$  respectivamente, utilizando la función  $N$  para generar valores con media 0 y desviación estándar 1.

Con los valores de  $vars_x$  se realiza la primera evaluación de la función objetivo y se almacena en  $fold$  y  $fnew$ , misma evaluación se almacenan en el archivo de texto.

```

1 // INICIALIZACION DE VARIABLES ALEATORIAS
2   initrandom(time(0));
3   vars_x[0] = N(0, 1);
4   vars_x[1] = N(0, 1);
5
6   fold = f(vars_x[0], vars_x[1]);
7   fnew = fold;
8
9   archivo << "primera evaluacion f(" << vars_x[0] << ", " << vars_x[1]
10      << ") = " << fold << std::endl;
11   std::cout << "primera evaluacion f(" << vars_x[0] << ", " << vars_x[1]
12      << ") = " << fold << std::endl;

```

El algoritmo implementa un ciclo **while** que itera desde  $t = 0$  hasta alcanzar el valor de  $G_{\max}$  especificado por el usuario. En cada iteración:

- Se reinicializa la semilla aleatoria mediante la función `initrandom(time(0))`
- `time(0)` es una función estándar de C que devuelve el tiempo actual del sistema en segundos, garantizando así diferentes secuencias aleatorias en cada ejecución

Este diseño cumple con el requerimiento especificado en el enunciado del problema, asegurando la generación de números pseudoaleatorios distintos en cada ejecución del programa.

```

1 while (t < gmax) {
2     // SEMILLA ALEATORIO
3     initrandom(time(0));

```

En la sección de mutación del algoritmo, los valores de `vars_x` se copian a `vars_x_` y posteriormente se modifican mediante un proceso iterativo. Para cada elemento del vector, se aplica una mutación sumando al valor actual el producto del paso de mutación  $\sigma$  por un número pseudoaleatorio generado mediante la función  $N(0, 1)$ . Este proceso se repite para todas las variables del problema, garantizando así una perturbación controlada de la solución actual.

Posterior a la mutación, el algoritmo verifica que los valores resultantes para  $x_1$  y  $x_2$  se mantengan dentro de los rangos establecidos ( $x_1 \in [-3, 3]$  y  $x_2 \in [-2, 2]$ ). Cuando algún valor excede estos límites, se implementa un mecanismo de corrección que consiste en reemplazar el valor fuera de rango por cero y luego sumarle (o restarle, según corresponda) un nuevo valor generado por  $N(0, 1)$ .

```

1 // MUTACION DE LAS VARIABLES
2   vars_x_[0] = vars_x[0];
3   vars_x_[1] = vars_x[1];
4   for (int i = 0; i < n_vars; i++) {
5       double random_value = N(0, 1);
6       vars_x_[i] += sigma[t] * random_value;
7   }
8

```

```

9      // VERIFICACIOON DEL RANGO DE LAS VARIABLES
10     if (vars_x_[0] < -3.0 || vars_x_[0] > 3.0) {
11         if (vars_x_[0] > 0) {
12             vars_x_[0] = 0 + N(0, 1);
13         } else {
14             vars_x_[0] = 0 - N(0, 1);
15         }
16     }
17     if (vars_x_[1] < -2.0 || vars_x_[1] > 2.0) {
18         if (vars_x_[1] > 0) {
19             vars_x_[1] = 0 + N(0, 1);
20         } else {
21             vars_x_[1] = 0 - N(0, 1);
22         }
23     }

```

En este fragmento del código se evalúa la función con los valores mutados y se almacena el resultado en *fnew*, después si el resultado de *fnew* es menor que *fold* se entra a la condición if, donde se suma en 1 la variable *exito*, se actualiza el valor de *fold* = *fnew* y los valores de *vars<sub>x</sub>*. Al hacer esta operación es cuando el individuo mutado es mejor que su antecesor y lo remplazamos, así para continuar con las demás generaciones.

```

1  // EVALUAR EL NUEVO PUNTO
2      fnew = f(vars_x_[0], vars_x_[1]);
3
4      // SI EL NUEVO PUNTO ES MEJOR QUE EL ANTERIOR
5      if (fnew < fold) {
6          exito++;
7          fold = fnew;
8          for (int i = 0; i < n_vars; i++) {
9              vars_x[i] = vars_x_[i];
10         }
11     }

```

El valor de *ps* que indica la frecuencia de éxito de mutaciones, se actualiza cada 20 generaciones ( $10 * \text{numero de variables}$ ) donde se reinicia el contador de *exito*, como indica el siguiente punto.

```

1  // ACTUALIZAR VALOR DE PS CADA 10 * N GENERACIONES
2      if (t % (10 * n_vars) == 0) {
3          ps = (float)exito / (10 * (float)n_vars);
4          exito = 0;
5      }

```

Después, se imprime el siguiente texto de la evolución con los valores de *vars<sub>x</sub>* y se almacenan en el archivo de texto.

```

1 archivo << "evaluacion " << t << ": f(" << vars_x[0] << ", " << vars_x[1]
2       << ") = " << fold << std::endl;
3
4     t++;

```

El algoritmo actualiza el valor de  $\sigma$  cada dos generaciones, ajustándolo según el valor actual de  $p_s$ . El mecanismo de actualización sigue la regla 1/5 de Rechenberg:

- Si  $p_s > \frac{1}{5}$ :  $\sigma_{t+1} = \frac{\sigma_t}{c}$  (reduce el paso de mutación)
- Si  $p_s < \frac{1}{5}$ :  $\sigma_{t+1} = \sigma_t \cdot c$  (aumenta el paso de mutación)
- Si  $p_s = \frac{1}{5}$ :  $\sigma_{t+1} = \sigma_t$  (mantiene el paso actual)

```

1 // ACTUALIZACION DEL VALOR DE SIGMA
2 if (t > 0 && t % n_vars == 0) {
3     if (ps > 0.2) {
4         sigma[t] = sigma[t - n_vars] / c;
5     } else if (ps < 0.2) {
6         sigma[t] = sigma[t - n_vars] * c;
7     } else {
8         sigma[t] = sigma[t - n_vars];
9     }
10 } else if (t > 0) {
11     sigma[t] = sigma[t - 1];
12 }
13 }

```

Por ultimo, se imprime en pantalla la ultima evaluación con el mejor valor obtenido por los valores de  $vars_x$ , además de eliminar la variable *sigma* para no generar fugas de memoria así también como cerrar el archivo y terminar el programa.

```

1 std::cout << "evaluacion " << t << ": f(" << vars_x[0] << ", " << vars_x
  [1]
2       << ") = " << fold << std::endl;
3
4 delete[] sigma;
5 archivo.close();
6 return 0;
7 }

```

A continuación se presenta la implementación completa del algoritmo evolutivo (1+1)-EE, que incluye todos los bloques de código anteriormente explicados.

```

1 #include <cstdlib>
2 #include <ctime>
3 #include <fstream>
4 #include <iostream>

```



```

5
6 // FUNCIONES UTILIZADAS DEL CODIGO PROPORCIONADO
7 void initrandom(int seed);
8 double N(double m, double sigma);
9
10 // FUNCION A EVALUAR
11 float f(float x1, float x2) {
12     float x1_2 = x1 * x1;
13     float x2_2 = x2 * x2;
14     float x1_4 = x1_2 * x1_2;
15     return (4 - 2.1 * (x1 * x1) + (x1_4 / 3)) * x1_2 + x1 * x2 +
16         (-4 + 4 * (x2_2)) * x2_2;
17 }
18
19 int main(int argc, char *argv[]) {
20
21     // VERIFICACION DEL ARGUMENTO
22     if (argc < 2) {
23         std::cout << "falta proporcionar un valor para Gmax" << std::endl;
24         return 1;
25     }
26     int gmax = std::atoi(argv[1]);
27
28     // GUARDAR LOS RESULTADOS DE LAS EVALUACIONES EN UN TXT
29     std::ofstream archivo("resultados.txt");
30     if (!archivo.is_open()) {
31         std::cerr << "Error al abrir el archivo." << std::endl;
32         return 1;
33     }
34
35     // PARAMETROS
36     const int n_vars = 2;
37     const float c = 0.817;
38     int t = 0;
39     int exito = 0;
40
41     float vars_x[2] = {0.0, 0.0};
42     float vars_x_[2] = {0.0, 0.0};
43
44     float ps = 0.0;
45     float fold = 0;
46     float fnew = 0;
47
48     float *sigma = new float[gmax]();
49     sigma[0] = 3.0;
50
51     // INICIALIZACION DE VARIABLES ALEATORIAS
52     initrandom(time(0));
53     vars_x[0] = N(0, 1);
54     vars_x[1] = N(0, 1);
55
56     fold = f(vars_x[0], vars_x[1]);

```

```

57 fnew = fold;
58
59 archivo << "primera evaluacion f(" << vars_x[0] << ", " << vars_x[1]
60      << ") = " << fold << std::endl;
61 std::cout << "primera evaluacion f(" << vars_x[0] << ", " << vars_x[1]
62      << ") = " << fold << std::endl;
63
64 while (t < gmax) {
65     // SEMILLA ALEATORIO
66     initrandom(time(0));
67
68     // MUTACION DE LAS VARIABLES
69     vars_x_[0] = vars_x[0];
70     vars_x_[1] = vars_x[1];
71     for (int i = 0; i < n_vars; i++) {
72         double random_value = N(0, 1);
73         vars_x_[i] += sigma[t] * random_value;
74     }
75
76     // VERIFICACIOON DEL RANGO DE LAS VARIABLES
77     if (vars_x_[0] < -3.0 || vars_x_[0] > 3.0) {
78         if (vars_x_[0] > 0) {
79             vars_x_[0] = 0 + N(0, 1);
80         } else {
81             vars_x_[0] = 0 - N(0, 1);
82         }
83     }
84     if (vars_x_[1] < -2.0 || vars_x_[1] > 2.0) {
85         if (vars_x_[1] > 0) {
86             vars_x_[1] = 0 + N(0, 1);
87         } else {
88             vars_x_[1] = 0 - N(0, 1);
89         }
90     }
91
92     // EVALUAR EL NUEVO PUNTO
93     fnew = f(vars_x_[0], vars_x_[1]);
94
95     // SI EL NUEVO PUNTO ES MEJOR QUE EL ANTERIOR
96     if (fnew < fold) {
97         exito++;
98         fold = fnew;
99         for (int i = 0; i < n_vars; i++) {
100             vars_x[i] = vars_x_[i];
101         }
102     }
103
104     // ACTUALIZAR VALOR DE PS CADA 10 * N GENERACIONES
105     if (t % (10 * n_vars) == 0) {
106         ps = (float)exito / (10 * (float)n_vars);
107         exito = 0;
108     }

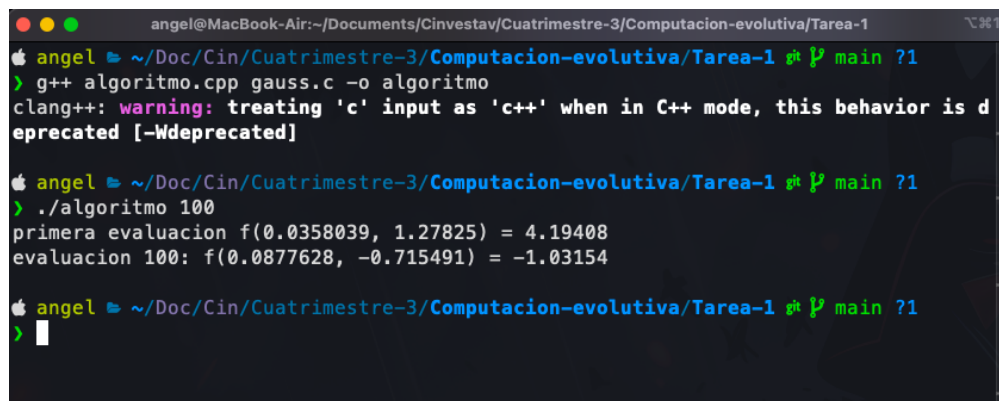
```

```

109
110     archivo << "evaluacion " << t << ": f(" << vars_x[0] << ", " << vars_x
[1]
111         << ") = " << fold << std::endl;
112
113     t++;
114
115     // ACTUALIZACION DEL VALOR DE SIGMA
116     if (t > 0 && t % n_vars == 0) {
117         if (ps > 0.2) {
118             sigma[t] = sigma[t - n_vars] / c;
119         } else if (ps < 0.2) {
120             sigma[t] = sigma[t - n_vars] * c;
121         } else {
122             sigma[t] = sigma[t - n_vars];
123         }
124     } else if (t > 0) {
125         sigma[t] = sigma[t - 1];
126     }
127 }
128 std::cout << "evaluacion " << t << ": f(" << vars_x[0] << ", " << vars_x
[1]
129         << ") = " << fold << std::endl;
130
131 delete[] sigma;
132 archivo.close();
133 return 0;
134 }

```

La forma de compilación y ejecución del código fuente se presenta en la siguiente figura 2 donde además se muestra la salida esperada del programa.



```

angel@MacBook-Air:~/Documents/Cinvestav/Cuatrimestre-3/Computacion-evolutiva/Tarea-1
angel ~ /Doc/Cin/Cuatrimestre-3/Computacion-evolutiva/Tarea-1 # main ?1
> g++ algoritmo.cpp gauss.c -o algoritmo
clang++: warning: treating 'c' input as 'c++' when in C++ mode, this behavior is deprecated [-Wdeprecated]

angel ~ /Doc/Cin/Cuatrimestre-3/Computacion-evolutiva/Tarea-1 # main ?1
> ./algoritmo 100
primera evaluacion f(0.0358039, 1.27825) = 4.19408
evaluacion 100: f(0.0877628, -0.715491) = -1.03154

angel ~ /Doc/Cin/Cuatrimestre-3/Computacion-evolutiva/Tarea-1 # main ?1
>

```

Figura 2: Compilación y ejecución del código

Al terminar la ejecución del programa, se genera un archivo de texto llamado resultados.txt” donde se almacenan todas las evaluaciones realizadas por el programa, a continua-

ción, se muestra una muestra del contenido del archivo resultados.txt”.

Salida de ejecución del algoritmo (1+1)-EE
primera evaluacion f(0.0358039, 1.27825) = 4.19408
evaluacion 0: f(-0.0567041, -1.10356) = 1.13661
evaluacion 1: f(0.286405, 0.76621) = -0.43606
evaluacion 2: f(0.286405, 0.76621) = -0.43606
evaluacion 3: f(0.286405, 0.76621) = -0.43606
:
evaluacion 96: f(0.0948604, -0.717938) = -1.03133
evaluacion 97: f(0.0877628, -0.715491) = -1.03154
evaluacion 98: f(0.0877628, -0.715491) = -1.03154
evaluacion 99: f(0.0877628, -0.715491) = -1.03154

A continuación se presenta la Tabla 1, que resume las 20 ejecuciones realizadas con el mismo valor de  $G_{\max}$  y diferentes valores iniciales de  $x_1$  y  $x_2$ , junto con el resultado final de  $f(x_1, x_2)$ .

En las estadísticas de las 20 ejecuciones, se observa que el **mejor valor encontrado** de  $f(x_1, x_2)$  es  $-1,03163$  (no  $-1,03053$ , que corresponde a la *media*), el cual aparece en múltiples corridas (3, 7, 8, 9, 11 y 17). Esto sugiere que el algoritmo encontró un **mínimo global** o un **mínimo local muy atractivo**, dada su recurrencia.

Además, se incluyen las siguientes métricas de dispersión:

- **Media** ( $\mu$ ):  $-1,03053$
- **Varianza** ( $\sigma^2$ ):  $9,1446 \times 10^{-6}$
- **Desviación estándar** ( $\sigma$ ):  $0,00302$

Cuadro 1: Resultados de las 20 corridas del algoritmo de optimización

Corrida	Valor inicial $x_1$	Valor inicial $x_2$	Mejor $f(x_1, x_2)$
1	1.84379	0.684678	$f(-0,0919961, 0,713641) = -1,0316$
2	-0.243764	1.3023	$f(0,097629, -0,742447) = -1,02405$
3	0.388637	0.133796	$f(-0,0900318, 0,712907) = -1,03163$
4	0.704838	0.331579	$f(0,112762, -0,718138) = -1,02947$
5	1.33724	-0.836928	$f(0,0923476, -0,717548) = -1,03142$
6	1.65344	-0.219302	$f(0,090587, -0,713294) = -1,03162$
7	-0.117918	0.596106	$f(-0,0905788, 0,712822) = -1,03163$
8	0.198283	-1.19003	$f(-0,0896456, 0,712486) = -1,03163$
9	0.830684	-0.374617	$f(0,0906394, -0,712765) = -1,03163$
10	1.14688	0.243008	$f(-0,093904, 0,712493) = -1,03156$
11	1.77929	-0.925499	$f(0,0899107, -0,712916) = -1,03163$
12	-0.308272	1.67604	$f(0,0572176, -0,728506) = -1,02484$
13	0.324129	0.507535	$f(-0,0912269, 0,714111) = -1,03161$
14	0.64033	-1.2786	$f(-0,0937421, 0,705779) = -1,03116$
15	1.27273	-0.463189	$f(0,0937676, -0,709963) = -1,0315$
16	1.90513	-1.6317	$f(0,090517, -0,714133) = -1,03161$
17	-0.182426	0.969845	$f(0,0896629, -0,712578) = -1,03163$
18	0.449975	-0.198662	$f(-0,0902724, 0,713977) = -1,03161$
19	0.766176	0.418964	$f(0,0720736, -0,718426) = -1,03002$
20	1.39858	1.23437	$f(0,0903892, -0,716529) = -1,03151$

**Estadísticas de los valores finales de  $f(x)$ :**

Media ( $\mu$ ):  $-1,03053$

Varianza ( $\sigma^2$ ):  $9,1446 \times 10^{-6}$

Desviación estándar ( $\sigma$ ):  $0,00302$

Mejor valor:  $-1,03163$  (Corridas 3, 7, 8, 9, 11, 17)

Peor valor:  $-1,02405$  (Corrida 2)

## Antecedentes biológicos

1.- (20 puntos) Investigue en qué consiste el *equilibrio puntuado* (*punctuated equilibrium*, en inglés) e indique si considera que se opone a los preceptos del Neo-Darwinismo o no. Fundamente bien sus argumentos.

**Respuesta:** El equilibrio puntuado no se opone al Neo-Darwinismo, pero sí amplía su marco teórico dentro de la Síntesis Moderna de la evolución.

**Argumento:** El equilibrio puntuado es una teoría propuesta por Stephen J. Gould y Niles Eldredge [2] que trata del ritmo y modo de la evolución. Esta teoría plantea que los cambios evolutivos importantes se producen por especiación divergente, es decir, cuando una

población queda aislada geográficamente y experimenta una rápida evolución, seguida de un período de estasis (estado estático) donde no se observan cambios físicos significativos. No se opone a la teoría Neo-Darwinista porque no rechaza los principios fundamentales como la selección natural, mutaciones o deriva génica, sino que complementa la teoría al proponer un patrón diferente del ritmo evolutivo.

2.- **(20 puntos)** Investigue en qué consisten las *mutaciones neutrales* (*neutral mutations*, en inglés), e indique si considera que se opone o no a los preceptos del Neo-Darwinismo. Fundamente bien sus argumentos.

**Respuesta:** Las mutaciones neutrales no se oponen a la teoría del Neo-Darwinismo, ya que añaden una nueva categoría de mutaciones que explican la variabilidad molecular no adaptativa.

**Argumento:** La teoría neutralista, propuesta por Motoo Kimura [3], amplía el conocimiento sobre las mutaciones al demostrar que la mayoría de los cambios moleculares son neutrales, es decir, no favorece o perjudica. Un concepto fundamental de esta teoría es la deriva génica, que explica cómo estas mutaciones neutrales pueden fijarse o eliminarse aleatoriamente en las poblaciones [4].

El Neo-Darwinismo ya contemplaba mecanismos no adaptativos como la deriva génica, por lo que la teoría neutralista no lo contradice, sino que muestra una visión de los procesos evolutivos a nivel molecular.

## Referencias

- [1] A E Eiben y J E Smith. *Introduction to evolutionary computing*. en. 2.<sup>a</sup> ed. Natural Computing Series. Berlin, Germany: Springer, jul. de 2015.
- [2] Sthepen Jay Gould. *uv.mx*. <https://www.uv.mx/personal/tcarmona/files/2010/08/Gould-19821.pdf>. [Accessed 15-05-2025].
- [3] Enrique P Lessa. “Vigencia del Darwinismo”. es. En: *Gayana (Concepci  )* 73 ( de 2009), págs. 73-84. ISSN: 0717-6538. URL: [http://www.scielo.cl/scielo.php?script=sci\\_arttext&pid=S0717-65382009000300007&nrm=iso](http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0717-65382009000300007&nrm=iso).
- [4] *Neutralismo* — *e1.portalacademico.cch.unam.mx*. <https://e1.portalacademico.cch.unam.mx/alumno/biologia2/neutralismoyequilibrio/neutralismo>. [Accessed 16-05-2025].