Josue Santana Robledo Corona 325073061

Maestría en Ciencias de la Robótica e Inteligencia Artificial

Algoritmos Bio-Inspirados

Centro Universitario de Ciencias Exactas e Ingeniería.

Mtro. Carlos Alberto López Franco

1. Introducción

En este proyecto se implementa la Evolución Diferencial en Python, con el fin de observar su comportamiento y capacidad de convergencia al aplicar distintas funciones objetivo. A través de la visualización iterativa, se analiza cómo la población evoluciona en cada generación y cómo el algoritmo se aproxima progresivamente hacia soluciones óptimas.

2. Descripción del código

El código implementa el algoritmo de Evolución Diferencial (DE) dentro de una clase en Python.

Su estructura está organizada en métodos que representan las principales operaciones del algoritmo:

inicializar_poblacion(): genera de forma aleatoria la población inicial dentro de los límites establecidos.

mutacion(): crea nuevos vectores mutantes combinando individuos seleccionados al azar de la población actual.

cruzamiento(): forma soluciones candidatas al mezclar los genes del individuo actual con los del vector mutante.

seleccion(): evalúa cada candidato frente a su individuo original y conserva el mejor.

ejecutar(): coordina todo el proceso evolutivo, almacenando el mejor valor por iteración y generando una gráfica del progreso.

Esta organización modular permite una implementación clara, reutilizable y fácilmente adaptable a distintas funciones objetivo.

Componentes del Código

Variables y parámetros iniciales

El programa inicia definiendo una serie de parámetros que controlan el comportamiento del a lgoritmo:

función_objetivo: Es la función matemática que se desea optimizar. Puede representar cualq uier problema, ya sea de minimización o maximización.

dimension: Número de variables del problema.

limites: Rango de valores permitido para cada variable, por ejemplo, entre -5 y 5.

max_iter: Cantidad máxima de iteraciones o generaciones del algoritmo.

tam_poblacion: Número de individuos que componen la población en cada iteración.

prob_cruza (CR): Probabilidad de cruzamiento entre las soluciones mutadas y las originales.

factor_escala (F): Coeficiente que controla la magnitud de la perturbación durante la mutació n diferencial.

minimizar: Indica si el objetivo es encontrar el valor mínimo o máximo de la función.

3. Algoritmo Principal

El proceso del algoritmo de Evolución Diferencial se desarrolla en varias etapas consecutivas:

Inicialización: Se genera aleatoriamente una población de posibles soluciones dentro de los límites especificados. Cada individuo es un vector de valores reales.

Mutación: Para cada individuo, se seleccionan tres soluciones distintas de la población.

Cruzamiento: Cada individuo original se combina con su respectivo vector mutante. Con probabilidad CR, se reemplaza un componente por el correspondiente del vector mutante, generando así una solución candidata.

Selección: Se evalúan las soluciones candidatas mediante la función objetivo. Si la nueva solución es mejor que la original, la reemplaza en la siguiente generación.

Evaluación y actualización: En cada iteración se registra el mejor valor obtenido, observando cómo la población converge progresivamente hacia la solución óptima.

Finalmente, el algoritmo devuelve el mejor valor encontrado y la correspondiente solución.

4. Resultados Obtenidos

Vamos primero con las 3 funciones en dos dimensiones

Primero vamos con el descendente de Esfera

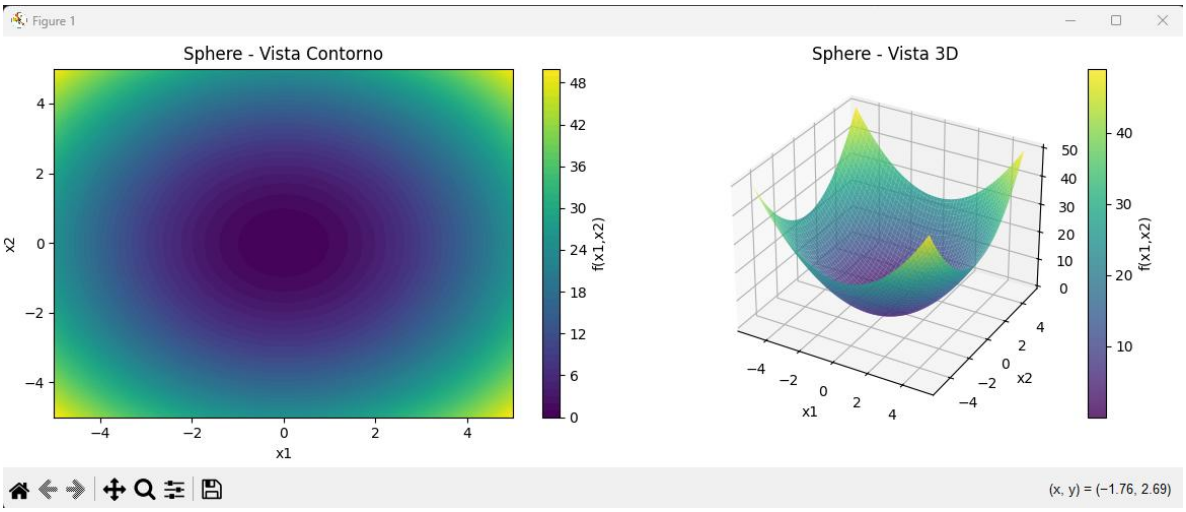

Imagen 1. Grafica de Esfera

Iteración 48/50 - Mejor valor (fitness): 0.000000
Iteración 49/50 - Mejor valor (fitness): 0.000000
Iteración 50/50 - Mejor valor (fitness): 0.000000

Mejor solución encontrada: [-5.28312906e-05 -3.48425407e-05]
Mejor valor de la función: 4.005147913768441e-09

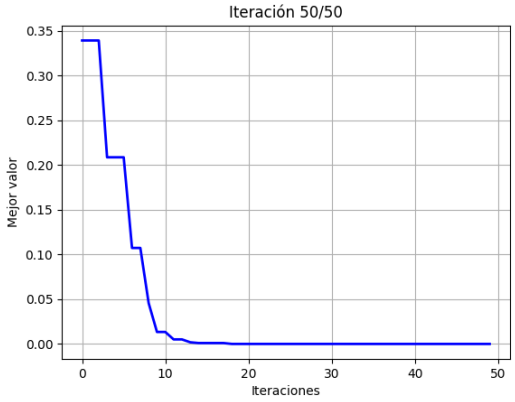

Imagen 2. Descendente Esfera

Ahora vamos con el ascendente

Iteración 48/50 - Mejor valor (fitness): 50.000000
Iteración 49/50 - Mejor valor (fitness): 50.000000
Iteración 50/50 - Mejor valor (fitness): 50.000000

Mejor solución encontrada: [5. -5.]
Mejor valor de la función: 50.0

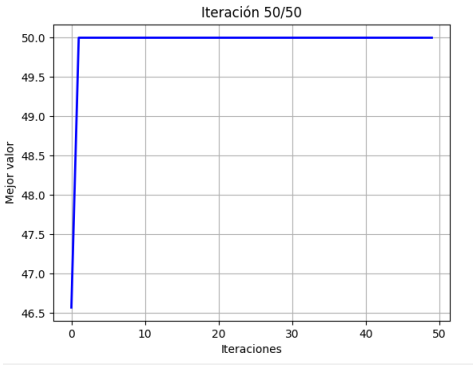

Imagen 3. Ascendente Esfera

Primero vamos con el descendente para la función Quadrico

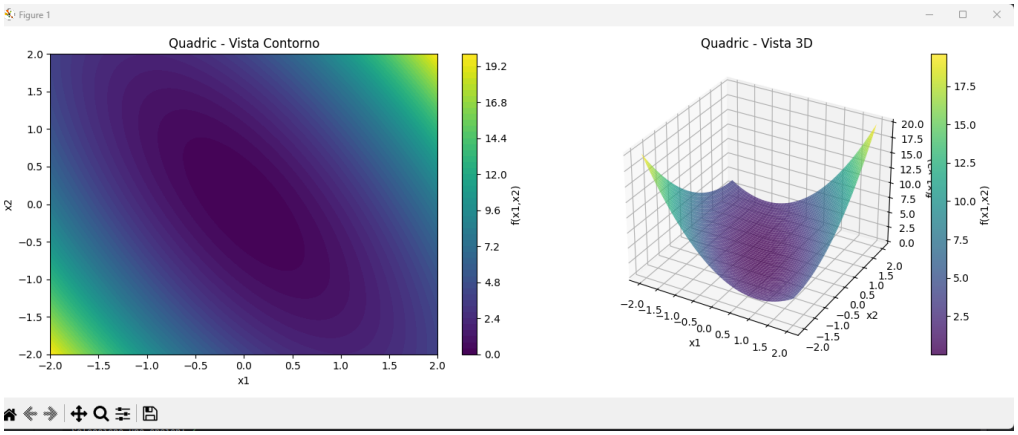

Imagen 4. Grafica de Quadrico

Iteración 48/50 - Mejor valor (fitness): 0.000000
Iteración 49/50 - Mejor valor (fitness): 0.000000
Iteración 50/50 - Mejor valor (fitness): 0.000000

Mejor solución encontrada: [-3.2904532e-04 3.7111227e-05]
Mejor valor de la función: 1.9349633726550862e-07

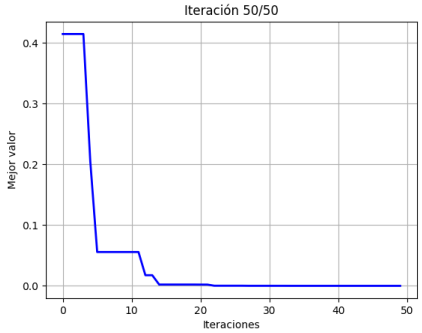

Imagen 5. Descendente Quadrico

Iteración 48/50 - Mejor valor (fitness): 125.000000
Iteración 49/50 - Mejor valor (fitness): 125.000000
Iteración 50/50 - Mejor valor (fitness): 125.000000

Mejor solución encontrada: [5. 5.]
Mejor valor de la función: 125.0

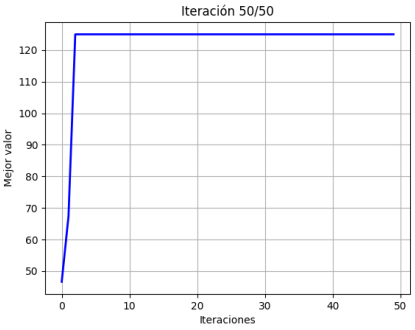

Imagen 6. Ascendente Quadrico

Primero vamos con el descendente para la función Schwefel

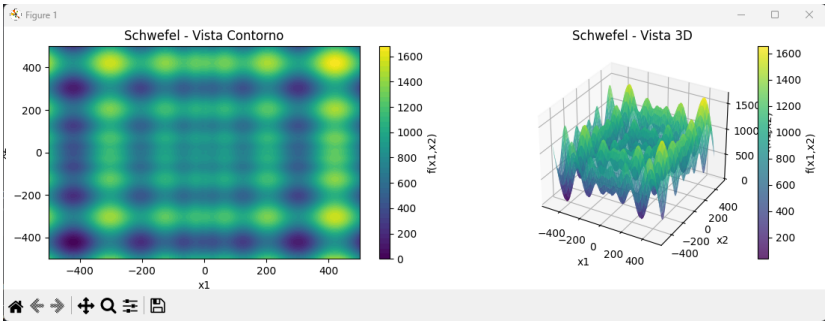

Imagen 7. Grafica de Schwefel

Iteración 48/50 - Mejor valor (fitness): 830.098309
Iteración 49/50 - Mejor valor (fitness): 830.098309
Iteración 50/50 - Mejor valor (fitness): 830.098309

Mejor solución encontrada: [-5. -5.]
Mejor valor de la función: 830.0983086845278

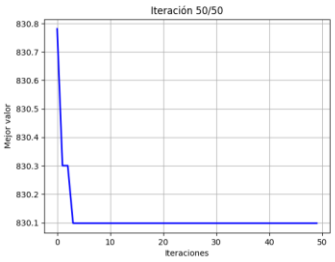

Imagen 8. Descendente Schwefel

Iteración 49/50 - Mejor valor (fitness): 845.833291
Iteración 50/50 - Mejor valor (fitness): 845.833291

Mejor solución encontrada: [5. 5.]
Mejor valor de la función: 845.8332913154721

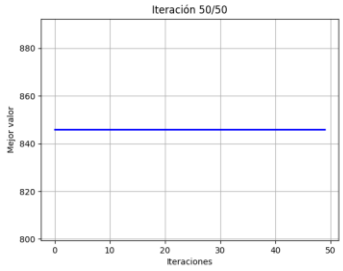

Imagen 9. Ascendente Schwefel

Vamos ahora con 3 funciones en n dimensiones

dimension = 5

Cambiamos esta línea en el código

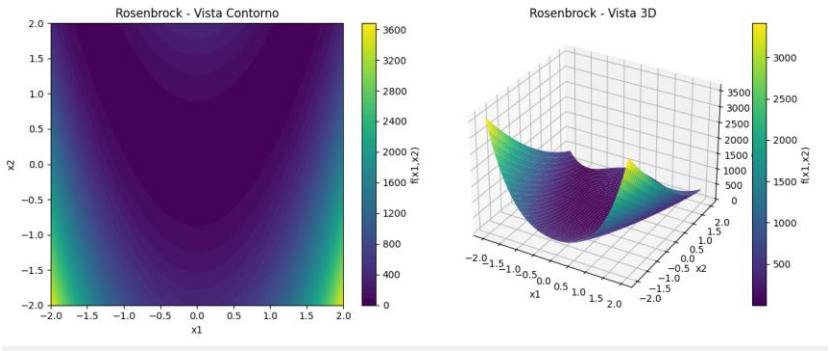

Imagen 10. Grafica de rosenbrock

Primero vamos con el descendente para la función rosenbrock

Iteración 48/50 - Mejor valor (fitness): 13.254100
Iteración 49/50 - Mejor valor (fitness): 13.254100
Iteración 50/50 - Mejor valor (fitness): 13.254100

Mejor solución encontrada: [0.41358948 0.1406726 0.15806514 -0.13104894 0.25685392]
Mejor valor de la función: 13.254099500145514

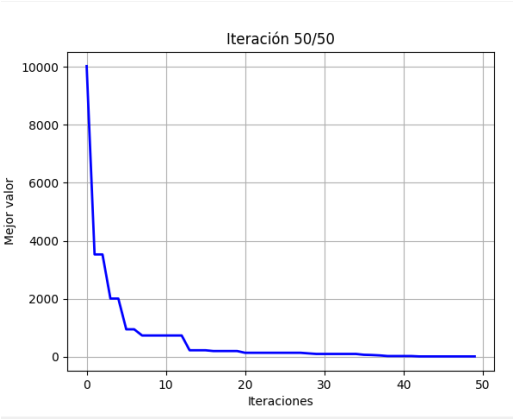

Imagen 11. Descendente rosenbrock

Iteración 49/50 - Mejor valor (fitness): 360144.000000
Iteración 50/50 - Mejor valor (fitness): 360144.000000

Mejor solución encontrada: [-5. -5. -5. -5. -5.]
Mejor valor de la función: 360144.0

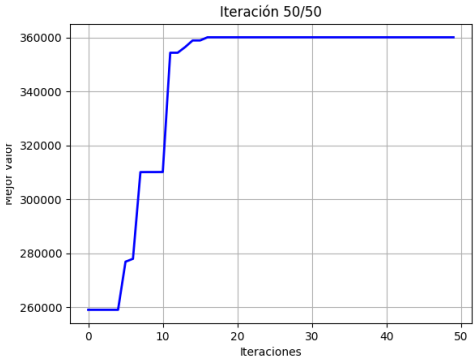

Imagen 12. Ascendente rosenbrock

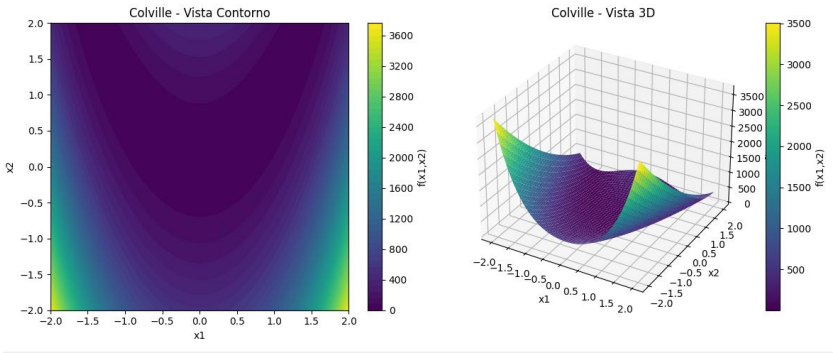

Imagen 13. Grafica de colville

Primero vamos con el descendente para la función colville

Iteración 48/50 - Mejor valor (fitness): 10.005298

Iteración 49/50 - Mejor valor (fitness): 3.634872
Iteración 50/50 - Mejor valor (fitness): 3.634872

Mejor solución encontrada: [1.3284564 1.8771394 -0.32549626 0.15326862 3.60717049]
Mejor valor de la función: 3.6348715710505672

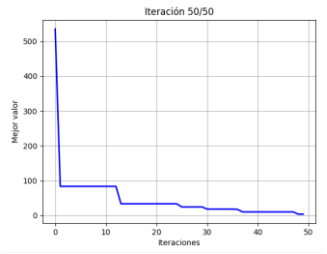

Imagen 14. Descendente colville

Iteración 49/50 - Mejor valor (fitness): 172492.000000

Iteración 50/50 - Mejor valor (fitness): 172492.000000

Mejor solución encontrada: [5. -5. -5. -5. -3.33331197]

Mejor valor de la función: 172492.0

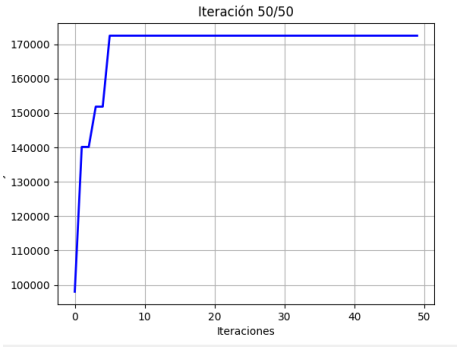

Imagen 15. Ascendente colville

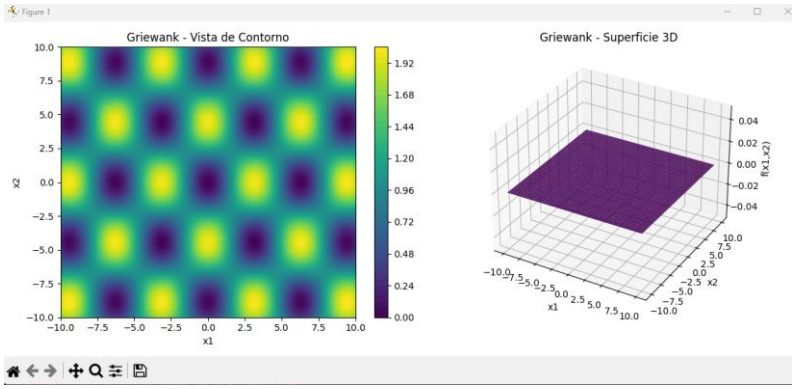

Imagen 16. Grafica de griewank

Primero vamos con el descendente para la función griewank

Iteración 48/50 - Mejor valor (fitness): 0.132109
Iteración 49/50 - Mejor valor (fitness): 0.132109
Iteración 50/50 - Mejor valor (fitness): 0.132109

Mejor solución encontrada: [-3.3740313 5. -0.27245348 -0.02339531
0.34235192]
Mejor valor de la función: 0.13210855630926832

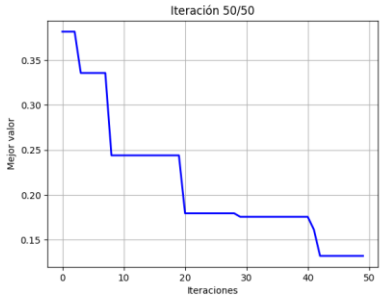

Imagen 17. Descendente griewank

Iteración 48/50 - Mejor valor (fitness): 1.968158
Iteración 49/50 - Mejor valor (fitness): 1.968158
Iteración 50/50 - Mejor valor (fitness): 1.968158

Mejor solución encontrada: [-3.00668479 -4.47817055 -5. -0.10156282
0.16956229]
Mejor valor de la función: 1.9681582934256294

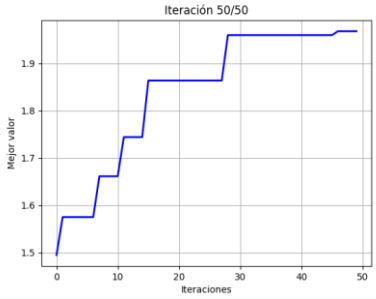

Imagen 18. Ascendente griewank

Así finalizamos todos los ejemplos solicitados con las funciones de prueba

5. Ventajas del Método

Convergencia estable: Tiende a mejorar progresivamente el valor de la función objetivo
Pocos parámetros a ajustar

Buena exploración del espacio de búsqueda: El uso de diferencias entre individuos permite mantener diversidad en la población

Simplicidad de implementación: Su estructura se basa en tres operaciones principales (mutación, cruzamiento y selección)

6. Limitaciones

Sensibilidad a los parámetros: El rendimiento depende fuertemente de la correcta elección del factor de escala

Riesgo de convergencia local: Aunque es menos probable que otros métodos, aún puede quedarse atrapado en óptimos locales

Costo computacional: Al evaluar muchas soluciones por iteración, puede ser más lento que métodos

7. Conclusiones

El algoritmo de Evolución Diferencial (DE) demostró ser una herramienta eficiente para la optimización, no tardaba mucho evaluando, solo hubo una función en donde se quedó atorado, y la realización en ciertas partes se parecía al algoritmo genético, por lo que pude basarme en él o reutilizar su estructura. Además, me pareció interesante cómo con pocos parámetros logra buenos resultados y converge rápido en la mayoría de los casos. También noté que ajustar el factor de mutación y la probabilidad de cruce puede cambiar bastante el comportamiento del algoritmo, así que vale la pena experimentar con diferentes valores.

En general, diría que cumple bien su objetivo y resulta ideal para probar optimización sin complicarse tanto con configuraciones o fórmulas complicadas.

8. Código

https://github.com/santana-robledo/Algorithm-Lab/blob/main/18.%20Evolución%20Diferencial.py

Aquí se encuentra el repositorio