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

Este reporte presenta la implementación de un algoritmo de gradiente descendente con reinicio, aplicado sobre una función no lineal con múltiples máximos y mínimos locales. A diferencia de funciones más simples (como la cuadrática), aquí se requiere realizar varios intentos desde diferentes puntos iniciales, para esto se encontró una función que cumpliera con estos requisitos de varios máximos y mínimos.

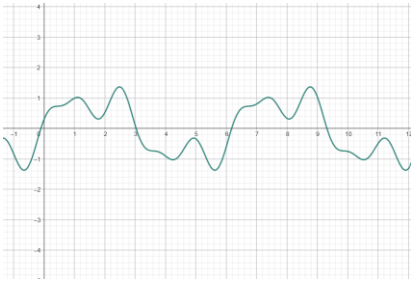

Imagen 1
Gráfica de Función

2. Descripción del código

El código implementa el gradiente descendente con múltiples reinicios. La idea es ejecutar el algoritmo varias veces desde valores iniciales aleatorios y distintos, dentro de un rango definido, y comparar los resultados obtenidos para identificar el mejor punto encontrado.

3. Componentes del Código

Variables y parámetros iniciales

- Rango = [-4, 8] → Rango usado para la gráfica.
- presicion = 0.0001 → Umbral para detener el algoritmo cuando el gradiente es suficientemente pequeño.
- paso = 0.2 → Tasa de aprendizaje (learning rate).
- max_iter = 300 → Número máximo de iteraciones por intento.
- rango_numinicial = [-3, 3] → Rango desde donde se eligen puntos iniciales aleatorios.
- num_intentos = 20 → Número de reinicios del gradiente descendente.

Función Objetivo

```
def funcion(x):
  return np.sin(x) + 0.5 * np.sin(3*x) + 0.3 * np.cos(5*x)
```

Función Derivada

```
def derivada(x):
  return np.cos(x) + 1.5 * np.cos(3*x) - 1.5 * np.sin(5*x)
```

4. Algoritmo Principal

El gradiente descendente sigue este procedimiento:

1. Calcular el gradiente en el punto actual:
   gradiente = derivada(x)
2. Actualizar el valor de x en la dirección contraria al gradiente:
   x = x - paso * gradiente
3. Repetir hasta alcanzar la precisión deseada o llegar al número máximo de iteraciones.

La implementación del gradiente ya quedó cubierta en el reporte anterior, así que en este reporte solo nos centraremos en cubrir la innovación, que es el reinicio, esto quedó muy conveniente, ya que desde el programa anterior el gradiente fue llamado desde una función, por lo que solo tenemos que meterlo de un ciclo de n iteraciones y guardar el mejor valor.

Esto se logra en este código

```
for i in range(num_intentos):
  valor_inicial = random.uniform(rango_numinicial[0], rango_numinicial[1])
  x_final, gradiente = gradiente_descendente(valor_inicial, paso, max_iter)
```

Definimos un valor inicial de números flotantes entre 0 y 1, definí este rango porque era el que más puntos me dejaba graficar, ya que me topé con un problema en la visualización, y es que no podía ver todos los 20 puntos en la gráfica, yo quería ver a donde llegaban todos los intentos, pero no lo logré, si tenia las 20 etiquetas de los puntos, sin embargo no tenía los puntos, intenté de más formas, con códigos de chatgpt, y aun así no lo lograba, así que supuse que es tema de la librería, sin embargo si grafiqué el mejor valor.

5. Resultados Obtenidos

18	-2.113	-2.03604588	-1.02502073
19	1.318	1.78851290	0.31351519
20	-2.618	-2.03604647	-1.02502073

*** MEJOR PUNTO ENCONTRADO ***
Intento: #3
Coordenadas: x = -0.463481, f(x) = -1.142702

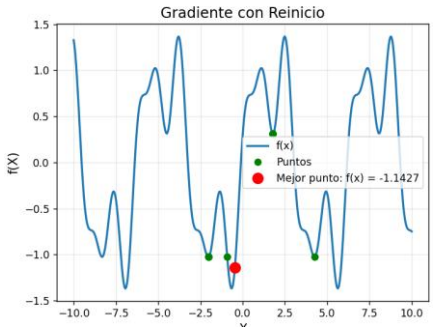

Imagen 2
Gráfica resultante de gradiente descendente

Y ahora los resultados del gradiente ascendente

indice_mejor = np.argmax(mejores_y)

A este código no debes solo cambiarle el signo del gradiente, también debes cambiar esta función de argmax a argmin, o viceversa, para que en el ascendente tome el valor más alto, y en el descendente el más bajo.

18	1.496	1.10554426	1.02502072
19	-1.707	-1.35306257	-0.31351519
20	0.546	1.10554617	1.02502073

*** MEJOR PUNTO ENCONTRADO ***
Intento: #3
Coordenadas: x = -3.605074, f(x) = 1.142702

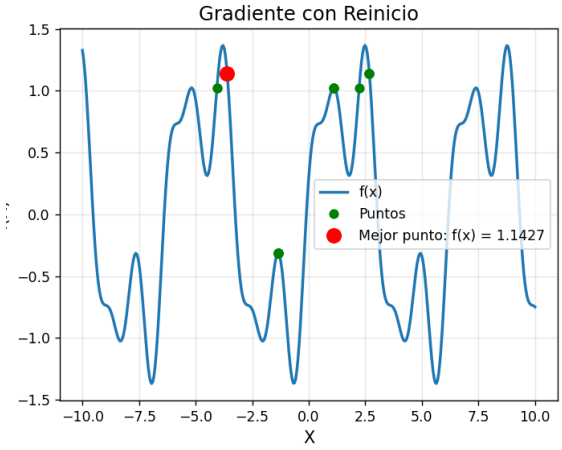

Imagen 3
Gráfica resultante de gradiente ascendente

6. Ventajas del Método

Permite escapar de mínimos locales al usar reinicios aleatorios.

7. Limitaciones

No garantiza encontrar el mínimo global, solo aproxima.

8. Conclusiones

El código implementa correctamente el algoritmo de gradiente ascendente y descendente con reinicios exitosamente, no se me ocurren muchas más conclusiones ya que el cambio fue sencillo, en la parte de reiniciar con nuevas variables no hubo problema, a lo mejor si le dediqué más tiempo a encontrar la mejor forma de guardar el mejor valor, de principio pusé que comprara cada valor nuevo con el anterior para que determine cual es el más

pequeño, sin embargo encontré mejor la forma de agregarlas a listas en cada iteración y tener todos los valores, esto hizo más fácil la graficación de los demás intentos y cuando intentaba resolver lo de los puntos que no aparecían fue una de las variables que vigilé, que estuvieran todos los intentos.

9. Código

```
import matplotlib.pyplot as plt

import numpy as np

import random


# Parámetros iniciales del algoritmo

Rango = [-4, 8]

presicion = 0.0001

paso = 0.2

max_iter = 300

rango_numinicial=[-3,3]

num_intentos=20


# Función objetivo

def funcion(x):

  return np.sin(x) + 0.5 * np.sin(3*x) + 0.3 * np.cos(5*x)


# Derivada

def derivada(x):

  return np.cos(x) + 1.5 * np.cos(3*x) - 1.5 * np.sin(5*x)


# Gradiente descendente

def gradiente_descendente(x_inicial, paso, max_iter):

  puntos_x = []

  puntos_y = []
```

```
    x = x_inicial
    puntos_x.append(x)
    puntos_y.append(funcion(x))

    gradiente = derivada(x)

    for i in range(max_iter):
        if abs(gradiente) > presicion:
            gradiente = derivada(x)
            x = x + (paso * gradiente)
        else:
            break
        puntos_x.append(x)
        puntos_y.append(funcion(x))

    return x, gradiente



print(f"{'Iteracion':<12}{'x_inicial':<25}{'X final':<20}{'f(X final)':<20}")
print("-" * 80)

mejores_x = []
mejores_y = []

for i in range(num_intentos):
    valor_inicial = random.uniform(rango_numinicial[0], rango_numinicial[1])
    x_final, gradiente = gradiente_descendente(valor_inicial, paso, max_iter)

    mejores_x.append(x_final)
```

```
    mejores_y.append(funcion(x_final))

    print(f"{i + 1:<12}{valor_inicial:<20.3f}{x_final:<20.8f}{funcion(x_final):<20.8f}")
##print(len(mejores_x),mejores_x)
#print(len(mejores_y),mejores_y)
indice_mejor = np.argmax(mejores_y)
mejor_x = mejores_x[indice_mejor]
mejor_y = mejores_y[indice_mejor]

print(f"\n*** MEJOR PUNTO ENCONTRADO ***")
print(f"Intento: #{indice_mejor + 1}")
print(f"Coordenadas: x = {mejor_x:.6f}, f(x) = {mejor_y:.6f}")

# Graficar
x = np.linspace(-10, 10, 400)
y_func = funcion(x)

plt.plot(x, y_func, label="f(x)", linewidth=2)
for i in range(len(mejores_x)):
    if i == 0:  # Solo agregar leyenda para el primer punto
        plt.plot(mejores_x[i], mejores_y[i], 'go', label='Puntos')
    else:  # Los demás puntos sin leyenda
        plt.plot(mejores_x[i], mejores_y[i], 'go', label='')

plt.plot(mejor_x, mejor_y, 'ro', markersize=10, label=f'Mejor punto: f(x) = {mejor_y:.4f}')

plt.title("Gradiente con Reinicio", fontsize=14)
plt.xlabel("X", fontsize=12)
plt.ylabel("f(X)", fontsize=12)
```

```
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```