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 del algoritmo de descenso de gradiente en n dimensiones, un método fundamental en optimización matemática y aprendizaje automático. A diferencia de los métodos unidimensionales o bidimensionales, el caso general en n dimensiones extiende el concepto utilizando vectores y gradientes multidimensionales. El algoritmo busca minimizar una función multivariable mediante actualizaciones iterativas en la dirección opuesta al gradiente.

2. **Descripción del código**

   El código implementa el descenso de gradiente para minimizar la función:
   np.sum(x ** 2)
   Dependiendo de la dimensión n definida en el código, la visualización se adapta automáticamente

3. **Componentes del Código**

   ```
   def funcion(x, y):
     return np.sin(5*x) * np.cos(5*y) / 5
   ```

   Esta función crea una superficie sinusoidal compleja con múltiples mínimos y máximos locales, ideal para demostrar el comportamiento del descenso de gradiente en 2D.

   Función Objetivo

   ```
   def funcion(x):
     return np.sum(x ** 2)

   def gradiente(x):
     return 2 * x
   ```

4. **Algoritmo Principal**

   ```
   def gradiente_descenso(x_inicial, tasa_aprendizaje=0.1, iteraciones=30):
     puntos = [x_inicial]
     valores = [funcion(x_inicial)]
     x = x_inicial.copy()

     for _ in range(iteraciones):
       x = x - tasa_aprendizaje * gradiente(x)  # actualización
       puntos.append(x.copy())
       valores.append(funcion(x))

     return np.array(puntos), np.array(valores)
   ```

En cada paso, el algoritmo actualiza la posición en dirección opuesta al gradiente, con un tamaño de paso determinado por la tasa de aprendizaje.

5. Resultados Obtenidos

Dependiendo del número de dimensiones n, se genera una gráfica adecuada.

- 1D y 2D → gráficas geométricas.
- 3D → trayectoria de parámetros.
- n > 3 → curva de convergencia.

Empezaremos con el descendente para n=3

28 [0.00148784 0.00289728 -0.00507049] 0.00003632 [0.00371961 0.00724319 -0.01267622]

29 [0.00119027 0.00231782 -0.00405639] 0.00002324 [0.00297569 0.00579455 -0.01014097]

30 [0.00095222 0.00185426 -0.00324511] 0.00001488 [0.00238055 0.00463564 -0.00811278]

Resultado final:
Punto = [0.00095222 0.00185426 -0.00324511]
f(x) = 0.00001488

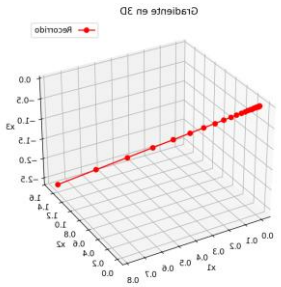

Imagen 1
Gráfica resultante de gradiente descendente

28 [489.61760576 610.26269013 628.1691914] 1006742.48385825 [816.02934293 1017.10448355 1046.94865234]

29 [587.54112691 732.31522815 753.80302968] 1449709.17675588 [979.23521152 1220.52538026 1256.3383828]

30 [705.04935229 878.77827378 904.56363562] 2087581.21452847 [1175.08225382 1464.63045631 1507.60605936]

Resultado final:
Punto = [705.04935229 878.77827378 904.56363562]
f(x) = 2087581.21452847

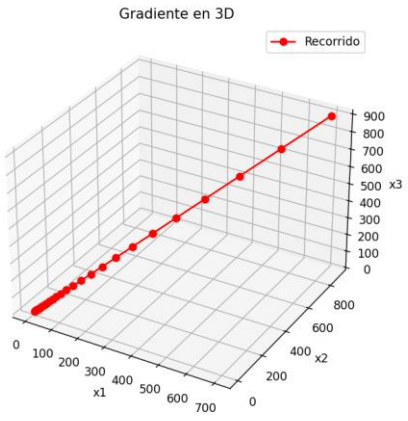

Imagen 2
Gráfica resultante de gradiente ascendente

6. Ventajas del Método

El mismo algoritmo funciona para cualquier número de variables (n), permitiendo optimizar funciones multivariables sin cambios estructurales en el código.

7. Limitaciones

Al igual que en 1D, puede quedar atrapado en mínimos locales, además que hacerlo de más dimensiones parece hacerlo más tardado

8. Conclusiones

Este código fue más difícil que los anteriores, se notó el aumento del grado e dificultad, este fue el primero que mientras veía en clase no sabía como implementaría en código, busqué ejemplos pero no encontraba sobre gradientes en tres dimensiones, sin mencionar que no sabía cómo graficaríamos en más de 3 dimensiones, todo se complicó todavía más cuando en una platica con el profesor me di cuenta que lo había implementado mal, que no era lo que él pedía, así que enfermo y de ultimo minuto aquí ando modificándolo, si bien fue muy retador yo creo que ha sido muy útil para el desarrollo de nuevas habilidades en Python.

9. Código

```
import numpy as np

import matplotlib.pyplot as plt
```

```
def funcion(x):
    return np.sum(x ** 2)



def gradiente(x):
    return 2 * x

def gradiente_descenso(x_inicial, tasa_aprendizaje=0.1, iteraciones=30):

    puntos = [x_inicial]
    valores = [funcion(x_inicial)]
    x = x_inicial.copy()

    print(f"{'Iteración':<10}{'x':<25}{'f(x)':<15}{'gradiente':<25}")
    print("-" * 80)

    for i in range(iteraciones):
        g = gradiente(x)
        x = x + tasa_aprendizaje * g  # descenso
        puntos.append(x.copy())
        valores.append(funcion(x))

        # imprimir cada iteración
        print(f"{i + 1:<10}{str(x):<25}{funcion(x):<15.8f}{str(g):<25}")

    # imprimir resultado final
    print("\nResultado final:")
    print(f" Punto = {x}")
    print(f" f(x)  = {funcion(x):.8f}")
```

```
    return np.array(puntos), np.array(valores)



def graficar(puntos, valores):
    n_dim = puntos.shape[1]

    if n_dim == 1:
        # Caso 1D
        x = np.linspace(-5, 5, 200)
        y = x ** 2
        plt.plot(x, y, label="f(x)")
        plt.plot(puntos[:, 0], valores, "ro-", label="Recorrido")
        plt.title("Gradiente en 1D")
        plt.xlabel("x")
        plt.ylabel("f(x)")
        plt.legend()
        plt.show()

    elif n_dim == 2:
        # Caso 2D con superficie
        x = np.linspace(-5, 5, 100)
        y = np.linspace(-5, 5, 100)
        X, Y = np.meshgrid(x, y)
        Z = X ** 2 + Y ** 2

        fig = plt.figure(figsize=(8, 6))
        ax = fig.add_subplot(111, projection="3d")
```

```
    ax.plot_surface(X, Y, Z, cmap="viridis", alpha=0.6)

    ax.plot(puntos[:, 0], puntos[:, 1], valores, "ro-", label="Recorrido")
    ax.set_title("Gradiente en 2D")
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("f(x,y)")
    ax.legend()
    plt.show()

elif n_dim == 3:
    # Caso 3D (trayectoria en espacio de parámetros)
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection="3d")

    ax.plot(puntos[:, 0], puntos[:, 1], puntos[:, 2], "ro-", label="Recorrido")
    ax.set_title("Gradiente en 3D")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("x3")
    ax.legend()
    plt.show()

else:
    # Caso n > 3 → graficar convergencia
    plt.plot(valores, "bo-")
    plt.title(f"Convergencia del Gradiente en {n_dim}D")
    plt.xlabel("Iteración")
    plt.ylabel("f(x)")
```

```
    plt.show()


n=3
x0 = np.random.uniform(-5, 5, size=n)  # punto inicial aleatorio en nD


puntos, valores = gradiente_descenso(x0, tasa_aprendizaje=0.1, iteraciones=30)
graficar(puntos, valores)
```