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 descenso de gradiente con reinicios aleatorios en n dimensiones, este presenta solo unos cuantos cambios respecto al anterior, que vamos a cubrir en este reporte, pero utiliza la misma función, solo hay cambios en el for, que ya explicaremos y la graficación, que si tuvo su grado de complejidad.

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
   El código implementa dos funciones, f(x y su derivada):

3. **Componentes del Código**

   Función Objetivo

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


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

4. **Algoritmo Principal**

Cada actualización acerca el vector de parámetros hacia el mínimo global. La tasa de aprendizaje controla la magnitud del paso y el número de iteraciones determina cuánto avanza el proceso.

La implementación del ciclo ocurre aquí:

```
for i in range(num_intentos):
  valor_inicial = np.random.uniform(-5, 5, size=n)
  trayectoria, valores = gradiente_descenso(valor_inicial, tasa_aprendizaje=0.1,
iteraciones=30)
```

sigue la misma lógica que el código anterior, guardando los puntos para graficarlos después

```
indice_mejor = np.argmin(mejores_y)
mejor_x = mejores_x[indice_mejor]
mejor_y = mejores_y[indice_mejor]
```

y obteniendo el mejor valor para graficarlo

El único reto fue graficar, ya que graficar en n dimensiones me sigue pareciendo al inusual, así que los ejemplos los corroboraba en n=1 y n=2, nuevamente encontramos el problema de los

puntos que el código anterior, y esta vez solo pude graficar el mejor punto.

5. Resultados Obtenidos

Vamos a ir con el descendente en n=2

29 [-0.00448932 0.00222388] 0.00002510 [-0.0112233 0.00555969]
30 [-0.00359146 0.0017791] 0.00001606 [-0.00897864 0.00444775]
20 [-2.90115558 1.43714707] [-0.00359146 0.0017791] 0.00001606

*** MEJOR PUNTO FINAL ENCONTRADO ***
Intento: #1
Coordenadas: x = [-0.00109068 -0.00198661], f(x) = 0.000005

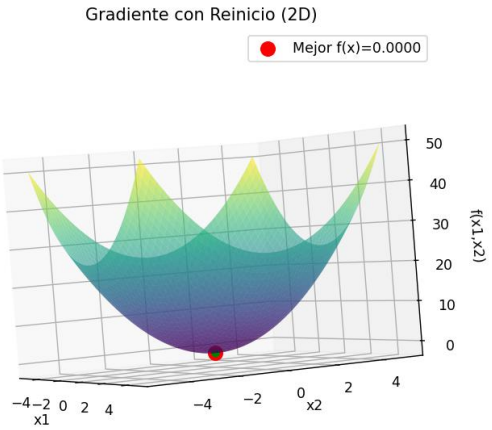

Imagen 1
Gráfica resultante de gradiente descendente

Implementación del gradiente ascendente

29 [-30.26515236 805.80938999] 650244.75243894 [-50.4419206 1343.01564998]
30 [-36.31818283 966.97126798] 936352.44351208 [-60.53030472 1611.61877997]
20 [-0.15299834 4.07357943] [-36.31818283 966.97126798] 936352.44351208

*** MEJOR PUNTO FINAL ENCONTRADO ***
Intento: #4
Coordenadas: x = [-979.16014857 1128.87599701], f(x) = 2233115.613189

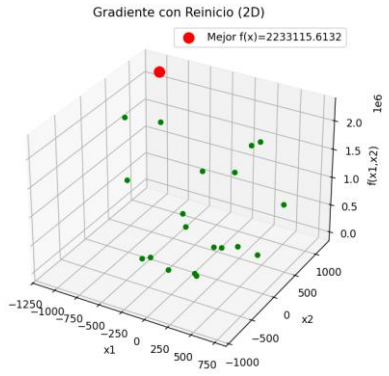

Imagen 2

Gráfica resultante de gradiente ascendente

Recordemos que para cambiar en este código de ascendente a descendente hay que cambiar

n = 2 para cambiar la dimensión

indice_mejor = np.argmax(mejores_y)

x = x + tasa_aprendizaje * g

estas líneas hay que cambiarlas para pasar de descendente a ascendente o al revés.

También algo que me percaté apenas en este reporte pero que vi que esta en el otro reporte es que cuando estas en n=2, cuando cambias de ascendente a descendente cambia el tipo de gráfico. No encontré una razón en el código, pero una rápida investigación me dice que puede ser la función.

Ahora vamos a ir con un n mayor a 2, 3.

Primero descendente

30 [0.00356276 -0.00077038 -0.00454394] 0.00003393 [0.00890689 -0.00192594 -0.01135985]

20 [2.87797232 -0.62230448 -3.67056665] [0.00356276 -0.00077038 -0.00454394] 0.00003393

*** MEJOR PUNTO FINAL ENCONTRADO ***

Intento: #9

Coordenadas: x = [-5.16355451e-05 -1.25036897e-03 -3.70239691e-03], f(x) = 0.000015

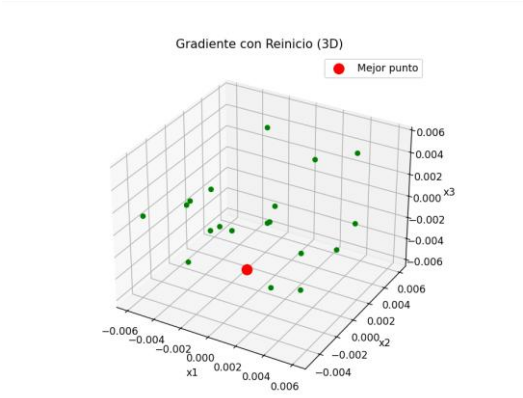

Imagen 3
Gráfica resultante de gradiente descendente n=3

Y ahora ascendente
30 [624.79433973 -826.14922242 705.74900849] 1570972.16764542 [1041.32389955 -1376.9153707 1176.24834748]
20 [2.63208376 -3.48033554 2.97312313] [624.79433973 -826.14922242 705.74900849] 1570972.16764542

*** MEJOR PUNTO FINAL ENCONTRADO ***
Intento: #2
Coordenadas: x = [941.11743486 1095.55394815 1095.05544088], f(x) = 3285086.898084

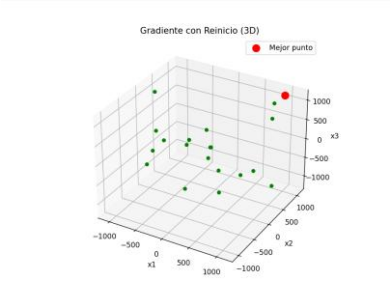

Imagen 3
Gráfica resultante de gradiente ascendente n=3

6. Ventajas del Método

Permite trabajar con cualquier número de dimensiones (n) sin modificar la estructura del código.
Los reinicios aleatorios reducen la dependencia de las condiciones iniciales.

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

La implementación de este código no llevo mucha novedad, era la fusión del código de gradiente en n dimensiones y reinicio de gradiente en 1d, de hecho mucho del código lo copié y pegué, si acaso en la graficación, sigo peleándome con que no aparecen los 20 puntos, y la inusualidad de la gráfica en n=2, pero en general solo fue copiar y pegar código y hacerlo funcionar, ya que las variables no se llamaban igual.

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
        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}")

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

def graficar(mejores_x, mejores_y, mejor_x, mejor_y, funcion):
    n_dim = np.array(mejores_x).shape[1]

    if n_dim == 1:
        # --- Caso 1D ---
        x = np.linspace(-10, 10, 400)
        y_func = [funcion(np.array([xi])) for xi in x]
        plt.plot(x, y_func, label="f(x)", linewidth=2)

        for i in range(len(mejores_x)):
            if i == 0:
                plt.plot(mejores_x[i], mejores_y[i], 'go', label='Puntos')
            else:
                plt.plot(mejores_x[i], mejores_y[i], 'go')

        plt.plot(mejor_x, mejor_y, 'ro', markersize=10,
                 label=f'Mejor punto: f(x) = {mejor_y:.4f}')
        plt.title("Gradiente con Reinicio (1D)", fontsize=14)
        plt.xlabel("X", fontsize=12)
        plt.ylabel("f(X)", fontsize=12)
        plt.grid(True, alpha=0.3)
        plt.legend()
        plt.show()
```

```
elif n_dim == 2:
    # --- Caso 2D ---
    x = np.linspace(-5, 5, 100)
    y = np.linspace(-5, 5, 100)
    X, Y = np.meshgrid(x, y)
    Z = np.zeros_like(X)

    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Z[i, j] = funcion(np.array([X[i, j], Y[i, j]]))

    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection="3d")
    ax.plot_surface(X, Y, Z, cmap="viridis", alpha=0.6)

    for i in range(len(mejores_x)):
        ax.scatter(mejores_x[i][0], mejores_x[i][1], mejores_y[i],
                   c='g', marker='o')

    ax.scatter(mejor_x[0], mejor_x[1], mejor_y,
               c='r', marker='o', s=100, label=f"Mejor f(x)={mejor_y:.4f}")

    ax.set_title("Gradiente con Reinicio (2D)")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("f(x1,x2)")
    ax.legend()
    plt.show()
```

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

    for i in range(len(mejores_x)):
      ax.scatter(mejores_x[i][0], mejores_x[i][1], mejores_x[i][2],
            c='g', marker='o')

    ax.scatter(mejor_x[0], mejor_x[1], mejor_x[2],
          c='r', marker='o', s=100, label="Mejor punto")

    ax.set_title("Gradiente con Reinicio (3D)")
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("x3")
    ax.legend()
    plt.show()

  else:
    # --- Caso n > 3: solo convergencia ---
    plt.plot(mejores_y, "bo-")
    plt.title(f"Convergencia del Gradiente en {n_dim}D")
    plt.xlabel("Intento")
    plt.ylabel("f(x)")
    plt.grid(True, alpha=0.3)
    plt.show()

n = 2
num_intentos = 20
```

```
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 = np.random.uniform(-5, 5, size=n)

  trayectoria, valores = gradiente_descenso(valor_inicial, tasa_aprendizaje=0.1,
iteraciones=30)


  x_final = trayectoria[-1]

  y_final = valores[-1]


  mejores_x.append(x_final)

  mejores_y.append(y_final)


  print(f"{i + 1:<12}{valor_inicial}{x_final}{y_final:.8f}")



indice_mejor = np.argmin(mejores_y)

mejor_x = mejores_x[indice_mejor]

mejor_y = mejores_y[indice_mejor]


print(f"\n*** MEJOR PUNTO FINAL ENCONTRADO ***")

print(f"Intento: #{indice_mejor + 1}")

print(f"Coordenadas: x = {mejor_x}, f(x) = {mejor_y:.6f}")


graficar(mejores_x, mejores_y, mejor_x, mejor_y, funcion)
```