UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍA

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

SEMINARIO DE SOLUCION DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL II

Profesor: OLIVA NAVARRO, DIEGO ALBERTO

Nombre: **Medina Herrera Jonathan**

Código: **217131222**

Carrera: **ingeniería en Computación**

Sección**: D05**

Contenido

Gradiente Descendiente:

Instrucciones:

Crea un software que por medio del descenso del gradiente sea capaz de optimizar la función adjunta en la imagen en los limites -1 a 1. Tiene que ser capaz de cambiar el lr (learning rate). Los valores iniciales tienen que ser de forma aleatoria.

Introducción:

El descenso del gradiente es un algoritmo de optimización utilizado en el campo del aprendizaje automático y la optimización. Su objetivo es encontrar el mínimo de una función realizando la iteración hacia abajo por lo largo del gradiente de la función. Busca el punto en que la pendiente de la función es cero, lo cual indica un mínimo local o global, dependiendo de la función. El descenso del gradiente se basa en el cálculo de las derivadas parciales de la función objetivo, con respecto a todas las variables de esta. Estas derivadas parciales nos dirán en qué dirección la función "asciende" más rápido, y, por lo tanto, en qué dirección deberemos ir para subir.

El objetivo de esta actividad es comprender el funcionamiento de este método y aplicarlo en una función para poder asimilar de manera práctica, como es que funciona este algoritmo.

Procedimiento:

Para realizar esta actividad se eligió a Python como lenguaje para desarrollarlo. Lo primero es definir la función a optimizar, en este caso es:

$$f(x_1, x_2) = 10 - e^{-(x_1^2+3X_2^2)}$$

Después obtenemos las derivadas parciales con respecto a cada una de las variables, X_1 y X_2, utilizando el paquete sympy de Python.

```
def obtener_derivadas(variables, funcion):
    # Calcula las derivadas parciales
    derivada_respecto_a_x1 = sympy.diff(funcion, variables[0])
    derivada_respecto_a_x2 = sympy.diff(funcion, variables[1])

    return derivada_respecto_a_x1, derivada_respecto_a_x2
```

Inicializamos los valores y parámetros de manera aleatoria dentro del rango -1 a 1, definimos la tasa de aprendizaje y el número máximo de iteraciones.

```
x1_inicial = np.random.normal(-1, 1)
x2_inicial = np.random.normal(-1, 1)
lr = 0.1
num_iteraciones = 50
```

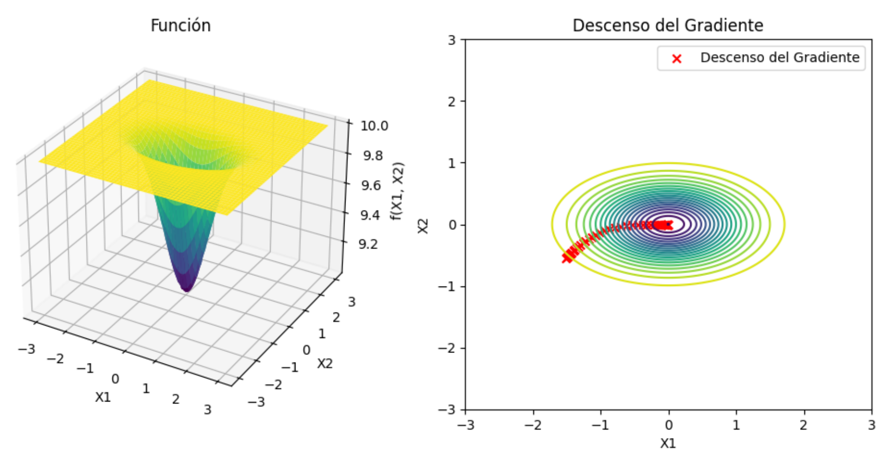
Después aplicamos la función y en cada iteración actualizamos los valores de x_1 x_2,en dirección opuesta al gradiente de la función, multiplicado por la tasa de aprendizaje, y en cada iteración, se registra el valor de las variables y el valor de la función objetivo.

```
historial_x1 = []
historial_x2 = []

for i in range(num_iteraciones):
    grad_x1, grad_x2 = derivadas

    # Actualización de los valores
    x1_inicial -= lr * grad_x1.subs({x1: x1_inicial, x2: x2_inicial})
    x2_inicial -= lr * grad_x2.subs({x1: x1_inicial, x2: x2_inicial})

    historial_x1.append(x1_inicial)
    historial_x2.append(x2_inicial)

    valor_funcion = funcion.subs({x1: x1_inicial, x2: x2_inicial}).evalf()
    print(f"Iter {i + 1}: x1 = {x1_inicial:.6f}, x2 = {x2_inicial:.6f}, f(x1, x2) = {valor_funcion:.6f}")

print(f"Valor mínimo estimado: f({x1_inicial}, {x2_inicial}) = {funcion.subs({x1: x1_inicial, x2: x2_inicial})}")
```

Y por último graficamos la función y la convergencia de error.

Resultados:

Como se puede ver, los valores que optimizan la función son x1 = 0 y x2 = 0, y también se puede observar que el algoritmo fue acercándose de manera correcta al resultado más optimo



También podemos observar como el algoritmo inicia con valores muy alejados al resultado y dentro de las primeras 10 iteraciones estaba lejos del resultado.

```
Iter 1: x1 = -1.512001, x2 = -0.553875, f(x1, x2) = 9.959501
Iter 2: x1 = -1.499754, x2 = -0.539911, f(x1, x2) = 9.956009
Iter 3: x1 = -1.486559, x2 = -0.525087, f(x1, x2) = 9.952022
Iter 4: x1 = -1.472295, x2 = -0.509320, f(x1, x2) = 9.947444
Iter 5: x1 = -1.456819, x2 = -0.492515, f(x1, x2) = 9.942158
Iter 6: x1 = -1.439966, x2 = -0.474566, f(x1, x2) = 9.936017
Iter 7: x1 = -1.421539, x2 = -0.455362, f(x1, x2) = 9.928841
Iter 8: x1 = -1.401308, x2 = -0.434777, f(x1, x2) = 9.920401
Iter 9: x1 = -1.379000, x2 = -0.412684, f(x1, x2) = 9.910415
Iter 10: x1 = -1.354292, x2 = -0.388952, f(x1, x2) = 9.898526
```

Es hasta después de las 27 iteraciones donde el resultado ya está muy cercano al valor optimo.

```
Iter 20: x1 = -0.869836, x2 = -0.078349, f(x1, x2) = 9.539310
Iter 21: x1 = -0.789691, x2 = -0.053612, f(x1, x2) = 9.468597
Iter 22: x1 = -0.705762, x2 = -0.034232, f(x1, x2) = 9.394449
Iter 23: x1 = -0.620287, x2 = -0.020302, f(x1, x2) = 9.320224
Iter 24: x1 = -0.535956, x2 = -0.011173, f(x1, x2) = 9.249956
Iter 25: x1 = -0.455558, x2 = -0.005728, f(x1, x2) = 9.187493
Iter 26: x1 = -0.381529, x2 = -0.002757, f(x1, x2) = 9.135485
Iter 27: x1 = -0.315562, x2 = -0.001260, f(x1, x2) = 9.094786
Iter 28: x1 = -0.258431, x2 = -0.000553, f(x1, x2) = 9.064606
Iter 29: x1 = -0.210084, x2 = -0.000235, f(x1, x2) = 9.043176
Iter 30: x1 = -0.169882, x2 = -0.000098, f(x1, x2) = 9.028447
```

Y finalmente en las ultimas 10 iteraciones el resultado solo tiene margen de error de menos de milésimas.

```
Iter 40: x1 = -0.018607, x2 = -0.000000, f(x1, x2) = 9.000346
Iter 41: x1 = -0.014887, x2 = -0.000000, f(x1, x2) = 9.000222
Iter 42: x1 = -0.011910, x2 = -0.000000, f(x1, x2) = 9.000142
Iter 43: x1 = -0.009529, x2 = -0.000000, f(x1, x2) = 9.000091
Iter 44: x1 = -0.007623, x2 = -0.000000, f(x1, x2) = 9.000058
Iter 45: x1 = -0.006099, x2 = -0.000000, f(x1, x2) = 9.000037
Iter 46: x1 = -0.004879, x2 = -0.000000, f(x1, x2) = 9.000024
Iter 47: x1 = -0.003903, x2 = -0.000000, f(x1, x2) = 9.000015
Iter 48: x1 = -0.003123, x2 = -0.000000, f(x1, x2) = 9.000010
Iter 49: x1 = -0.002498, x2 = -0.000000, f(x1, x2) = 9.000006
Iter 50: x1 = -0.001998, x2 = -0.000000, f(x1, x2) = 9.000004
Valor mínimo estimado: f(-0.00199843400105655, -1.16667183533265E-12) = 9.00000399373048
```

Conclusiones:

El algoritmo de descenso de gradiente es muy útil y versátil para la optimización de funciones. Usando la iteración y ajuste de los parámetros, se puede encontrar la respuesta óptima. Sin embargo, varios factores pueden afectar su eficiencia, como la selección del learning rate y los valores iniciales de los parámetros. Debe realizarse una serie de pruebas para encontrar la mejor combinación que garantice la rápida convergencia de las soluciones a la precisión necesaria. Y en esta actividad se pudo comprobar el funcionamiento y algunas de las limitantes de este algoritmo.