Practica - Parte 1 Resuelve el siguiente problema de optimizacion con el m´etodo de Newton. Observa detenidamente la siguiente

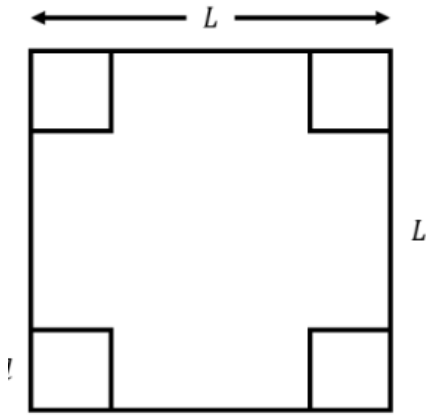


figura:

Para maximizar el volumen de la figura, debemos encontrar el valor óptimo de (l) que maximice la función objetivo (f(x) = (20 - 2x)(20 - 2x)x). Utilizaremos los límites dados para definir el espacio de búsqueda:

(x_l = 0) (x_u = 10) Primero, calculemos la derivada de (f(x)) con respecto a (x):

[f'(x) = \frac{d}{dx}[(20 - 2x)(20 - 2x)x]]

Aplicando la regla del producto y la regla de la cadena, obtenemos:

[f'(x) = (20 - 2x)(20 - 2x) + x(-4)(20 - 2x) + (20 - 2x)(-4x)]

Simplificando:

[f'(x) = 4x^2 - 80x + 400]

Para encontrar los puntos críticos, igualamos la derivada a cero:

[4x^2 - 80x + 400 = 0]

Resolviendo esta ecuación cuadrática, encontramos dos soluciones:

(x_1 = 5) (x_2 = 5) Dado que ambos valores son iguales, tenemos un punto crítico único. Ahora evaluemos la segunda derivada o jasiano o algo asi dijo el profe de (f(x)) en (x = 5):

[f''(x) = \frac{d2}{dx2}[(20 - 2x)(20 - 2x)x]]

Aplicando la regla del producto y la regla de la cadena nuevamente, obtenemos:

[f''(x) = 12x - 80]

Evaluando en (x = 5):

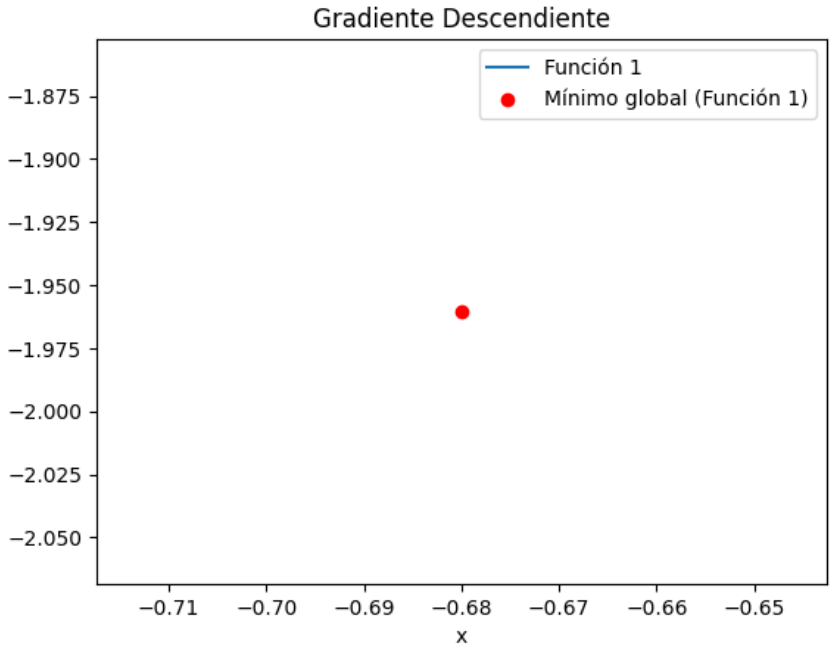
[f''(5) = 12(5) - 80 = 40 > 0]

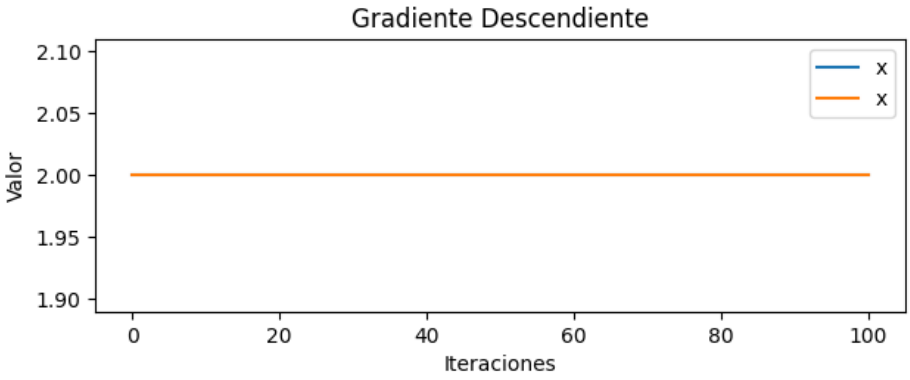
Como la segunda derivada es positiva en (x = 5), tenemos un mínimo local en ese punto. Por lo tanto, el valor de (l) que maximiza el volumen es (l = 5) centímetros.

En la parte 2 se nos pidio realizar un codigo para encontrar el minimo global de 2 funciones.

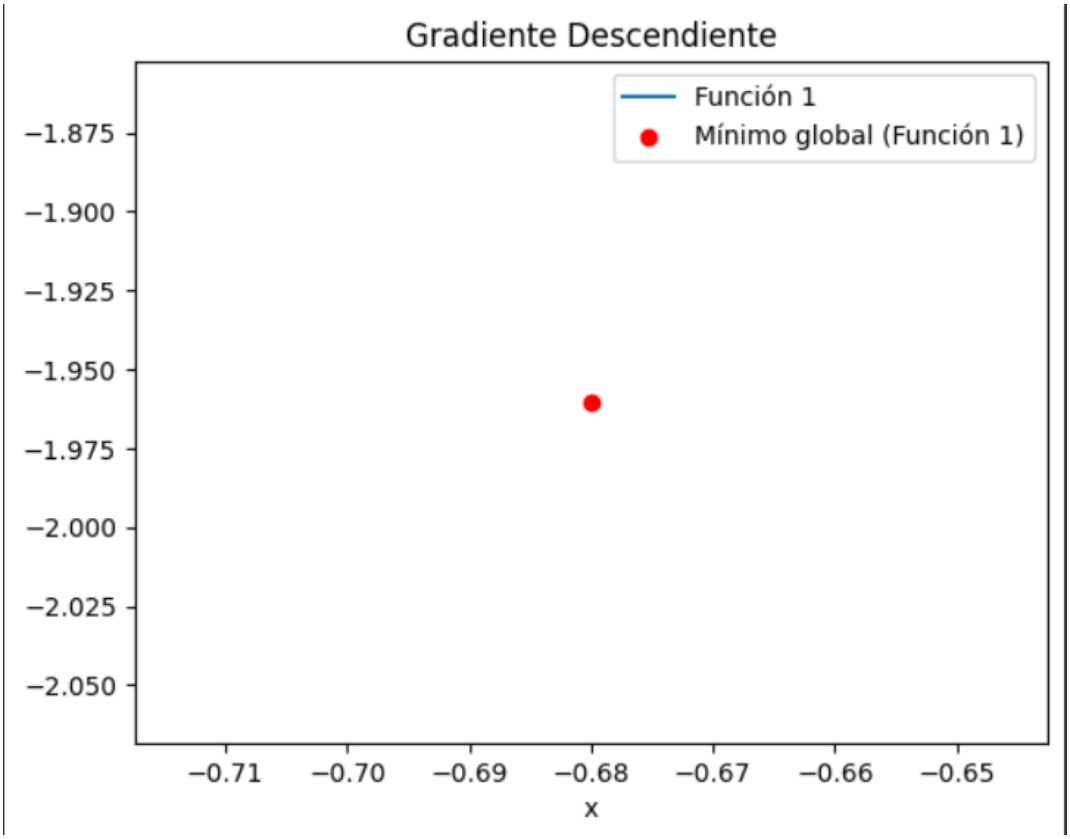
```
import numpy as np
import matplotlib.pyplot as plt

# Función 1: f(x, y)
def func1(x, y):
    return x * np.exp(-x**2 - y**2)

# Gradiente de la función 1
def grad_func1(x, y):
    df_dx = (1 - 2*x**2) * np.exp(-x**2 - y**2)
    df_dy = 2 * x * y * np.exp(-x**2 - y**2)
    return np.array([df_dx, df_dy])

# Función 2: f(x)
def func2(x):
    return np.sum((x - 2)**2)
```

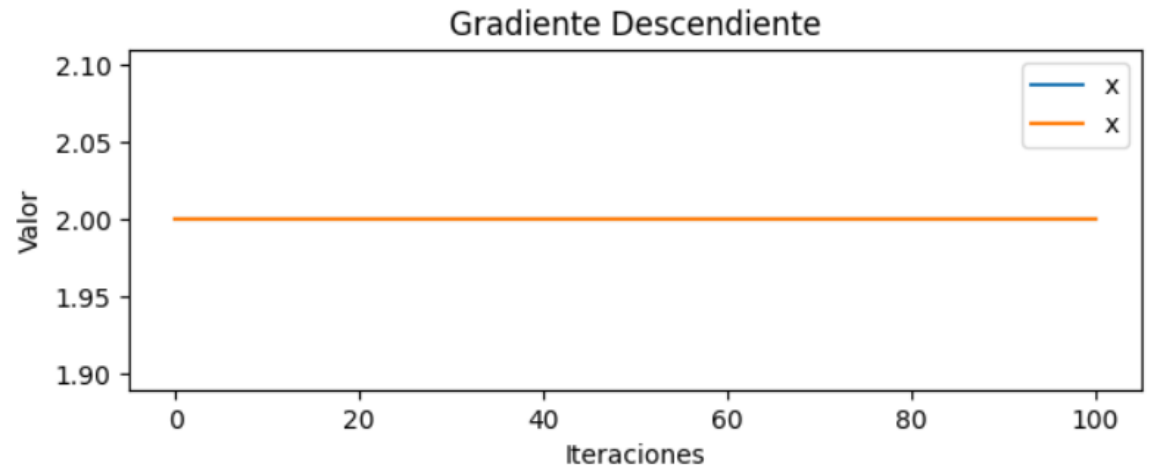
```
# Gradiente de la función 2
def grad_func2(x):
    return 2 * (x - 2)

# Método del Gradiente Descendiente
def gradient_descent(func1, grad_func, initial_x, learning_rate, num_iterations):
    x = initial_x
    history = [x]
    for _ in range(num_iterations):
        gradient = grad_func(x)  # Corregido aquí
        x -= learning_rate * gradient
        history.append(x)
    return x, history

# Método del Gradiente Descendiente
def gradient_descenty(func1, grad_func, initial_y,initial_x, learning_rate, num_iterations):
    x = initial_x
    history = [x]
    y = initial_y
    history = [y]
    for _ in range(num_iterations):
        gradient = grad_func(*x)  # Corregido aquí
        y -= learning_rate * gradient
        x -= learning_rate * gradient
        history.append(x)
        history.append(y)
    return x,history

# Parámetros
initial_x1 = np.array([-0.42888, -0.70711])  # Inicialización para función 1
initial_y1 = np.array([-0.42888, -0.70711])  # Inicialización para función 1
initial_x2 = np.array([1.5, 1.5])  # Inicialización para función 2
learning_rate = 0.1
num_iterations = 100

# Aplicar Gradiente Descendiente
x_min1, history1 = gradient_descenty(func1, grad_func1, initial_x1, initial_y1, learning_rate, num_iterations)
x_min2, history2 = gradient_descent(func2, grad_func2, initial_x2, learning_rate, num_iterations)

# Gráficas funcion 1
plt.plot(*zip(*history1), label="Función 1")
plt.scatter(*x_min1, color='red', marker='o', label="Mínimo global (Función 1)")
plt.xlabel("x")
plt.title("Gradiente Descendiente")
plt.legend()
plt.show()

# Gráfica para Función 2
plt.subplot(2, 1, 2)
plt.plot(history2, label='x')
plt.xlabel('Iteraciones')
plt.ylabel('Valor')
plt.title('Gradiente Descendiente')
plt.legend()

plt.tight_layout()
plt.show()
```





Por si no se ven las graficas:





Resultados de la primera grafica 0.6797, -1.9614

Resultados segunda grafica es (2,2)