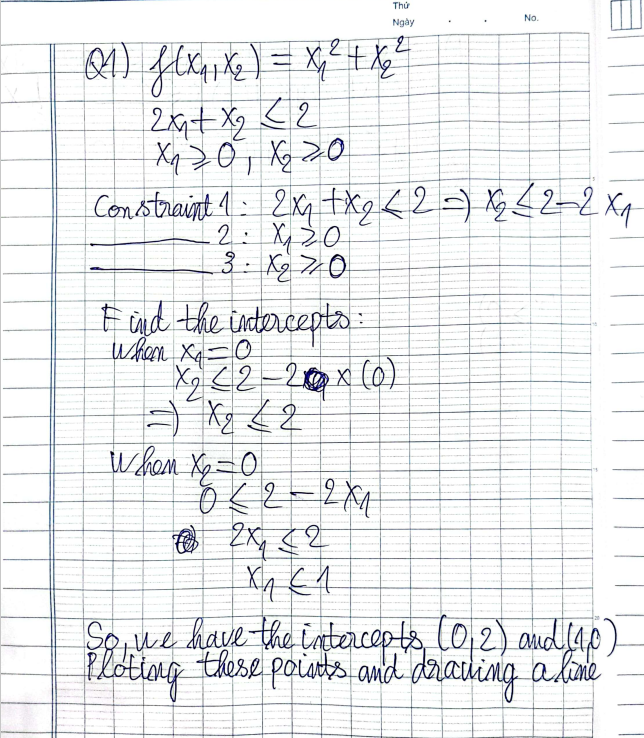
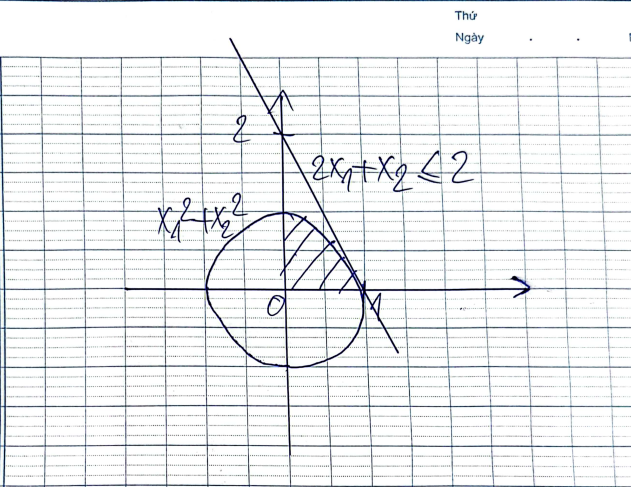
Name: Ngô Lê Thiên Ân

ID: ITITDK21030

Q1)





#Ngo Le Thien An - ITITDK21030

#Q1\_A

import numpy as np

import matplotlib.pyplot as plt

x1 = np.linspace(0, 2, 100)

x2\_constraint = 2 - 2 \* x1  # Rearranging the first constraint

# Plot the feasible region

plt.plot(x1, x2\_constraint, label="2x1 + x2 ≤ 2")

plt.fill\_between(x1, 0, x2\_constraint, where=(x1 >= 0) & (x2\_constraint >= 0), alpha=0.3, color='gray', label="Feasible Region")

# Set the x and y axis labels

plt.xlabel("x1")

plt.ylabel("x2")

# Set the x and y axis limits

plt.xlim(0, 2)

plt.ylim(0, 2)

# Add labels for the constraints

plt.text(0.5, 1.2, "2x1 + x2 ≤ 2", ha="center")

plt.text(0.2, 0.1, "x1 ≥ 0", ha="center")

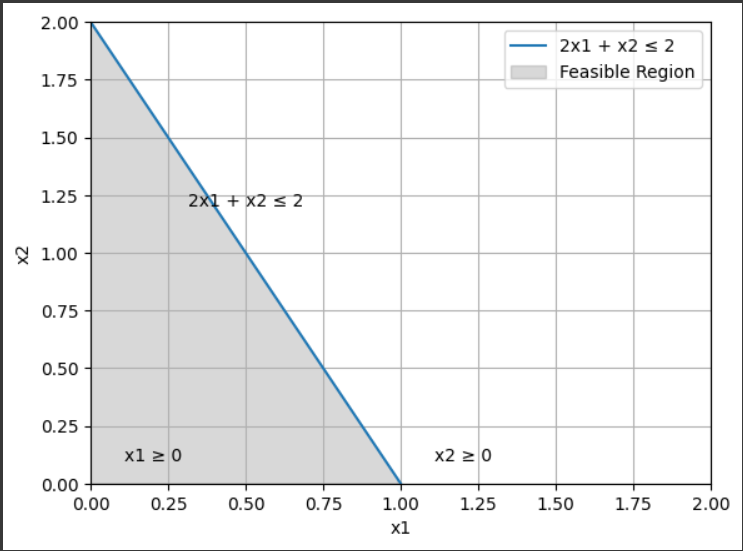
plt.text(1.2, 0.1, "x2 ≥ 0", ha="center")

# Show the plot

plt.legend()

plt.grid(True)

plt.show()



Q2)

#Ngo Le Thien An - ITITDK21030

#Q2\_A

# Define the function f(x, y)

def f(x, y):

    return 3.5 \* x + 2 \* y + x\*\*2 - x\*\*4 + 2 \* x \* y - y\*\*2

# Initialize initial guesses

x = 0

y = 0

# Set the step size

h = 0.1

# Perform three iterations

for iteration in range(3):

    # Calculate the gradient of f(x, y) with respect to x and y

    df\_dx = 3.5 + 2 \* x - 4 \* x\*\*3 + 2 \* y

    df\_dy = 2 - 2 \* y + 2 \* x

    # Update x and y using gradient ascent

    x = x + h \* df\_dx

    y = y + h \* df\_dy

    # Calculate the new value of the function

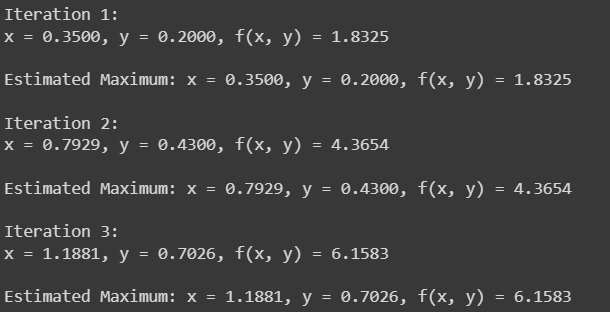
    result = f(x, y)

    # Print the results for each iteration

    print(f'Iteration {iteration + 1}:')

    print(f'x = {x:.4f}, y = {y:.4f}, f(x, y) = {result:.4f}\n')

    print(f'Estimated Maximum: x = {x:.4f}, y = {y:.4f}, f(x, y) = {result:.4f}\n')



#Ngo Le Thien An

#Q2\_B

import numpy as np

import matplotlib.pyplot as plt

# Define the function to maximize

def f(x, y):

    return 3.5 \* x + 2 \* y + x\*\*2 - x\*\*4 + 2 \* x \* y - y\*\*2

# Calculate the gradient numerically using the central difference method

def gradient(f, x, y, h=1e-4):

    df\_dx = (f(x + h, y) - f(x - h, y)) / (2 \* h)

    df\_dy = (f(x, y + h) - f(x, y - h)) / (2 \* h)

    return df\_dx, df\_dy

# Gradient ascent method

def gradient\_ascent(f, x0, y0, step\_size, num\_iterations):

    x\_values = [x0]

    y\_values = [y0]

    function\_values = [f(x0, y0)]

    for \_ in range(num\_iterations):

        df\_dx, df\_dy = gradient(f, x\_values[-1], y\_values[-1])

        x\_values.append(x\_values[-1] + step\_size \* df\_dx)

        y\_values.append(y\_values[-1] + step\_size \* df\_dy)

        function\_values.append(f(x\_values[-1], y\_values[-1]))

    return x\_values, y\_values, function\_values

# Different step sizes to test

step\_sizes = [0.01, 0.1, 0.5, 1]

num\_iterations = 50

plt.figure(figsize=(12, 6))

for step\_size in step\_sizes:

    x\_values, y\_values, function\_values = gradient\_ascent(f, 0, 0, step\_size, num\_iterations)

    plt.plot(range(num\_iterations + 1), function\_values, label=f"Step Size = {step\_size}")

plt.xlabel("Iterations")

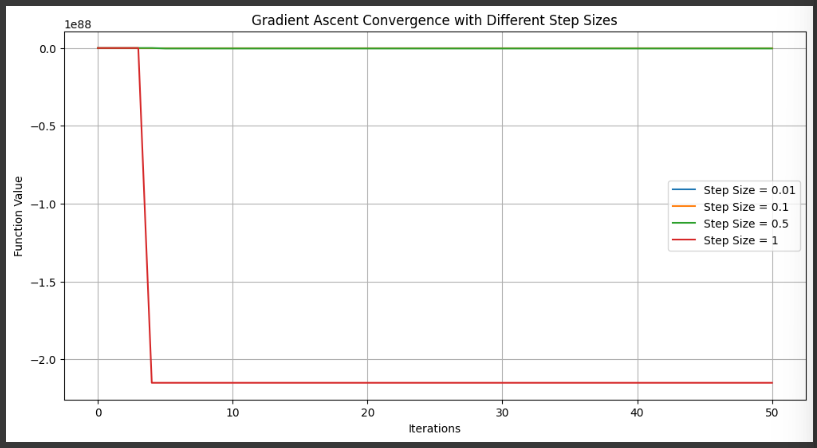
plt.ylabel("Function Value")

plt.title("Gradient Ascent Convergence with Different Step Sizes")

plt.legend()

plt.grid(True)

plt.show()



#Ngo Le Thien An - ITITDK21030

#Q2\_D

import numpy as np

# Define the function to maximize

def f(x, y):

    return 3.5 \* x + 2 \* y + x\*\*2 - x\*\*4 + 2 \* x \* y - y\*\*2

# Calculate the gradient vector of the function

def gradient(x, y):

    df\_dx = 3.5 + 2 \* x + 2 \* y - 4 \* x\*\*3

    df\_dy = 2 - 2 \* y + 2 \* x

    return np.array([df\_dx, df\_dy])

# Calculate the Hessian matrix of the function

def hessian(x, y):

    d2f\_dx2 = 2 - 12 \* x\*\*2

    d2f\_dy2 = -2

    d2f\_dxdy = 2

    return np.array([[d2f\_dx2, d2f\_dxdy], [d2f\_dxdy, d2f\_dy2]])

# Initialize the point (x, y)

x = 0.0

y = 0.0

# Maximum number of iterations

max\_iterations = 100

# Convergence criteria

tolerance = 1e-6

for iteration in range(max\_iterations):

    grad = gradient(x, y)

    hess = hessian(x, y)

    # Calculate the search direction using the inverse Hessian

    search\_direction = -np.linalg.solve(hess, grad)

    # Update the point (x, y)

    x += search\_direction[0]

    y += search\_direction[1]

    # Check for convergence

    if np.linalg.norm(search\_direction) < tolerance:

        break

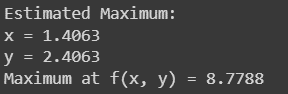
# Updated point is the estimate of the maximum

print("Estimated Maximum:")

print(f"x = {x:.4f}")

print(f"y = {y:.4f}")

print(f"Maximum at f(x, y) = {f(x, y):.4f}")



#Ngo Le Thien An - ITITDK21030

#Q3

import random

# Define the function to maximize

def f(x, y):

    return 3.5 \* x + 2 \* y + x\*\*2 - x\*\*4 - 2 \* x \* y - y\*\*2

# Range for x and y

x\_range = (-2, 2)

y\_range = (-2, 2)

# Number of random samples

num\_samples = 1000

# Initialize variables to store the maximum and corresponding point

max\_value = -float('inf')

max\_point = None

# Perform random search

for \_ in range(num\_samples):

    x = random.uniform(x\_range[0], x\_range[1])

    y = random.uniform(y\_range[0], y\_range[1])

    value = f(x, y)

    if value > max\_value:

        max\_value = value

        max\_point = (x, y)

# Maximum point and value

print("Estimated Maximum:")

print(f"x = {max\_point[0]:.4f}")

print(f"y = {max\_point[1]:.4f}")

print(f"f(x, y) = {max\_value:.4f}")

