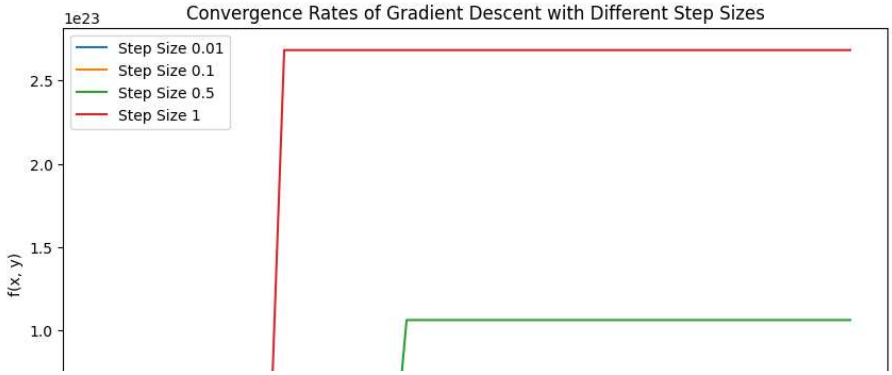
```
#Ngo Le Thien An - ITITDK21030
#Q2_B
import numpy as np
import matplotlib.pyplot as plt

# Define the function f(x, y)
def f(x, y):
    return -9 * x + x**2 + 11 * y + 4 * y**2 - 2 * x * y

# Numerical gradient calculation using central difference method
def numerical_gradient(x, y, h):
    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 descent method
def gradient_descent(learning_rate, num_iterations):
    x, y = 0, 0  # Initial guesses
    history = []  # To store the function values at each iteration

    for iteration in range(num_iterations):
        gradient_x, gradient_y = numerical_gradient(x, y, 1e-5)  # Numerical gradient
        x -= learning_rate * gradient_x
        y -= learning_rate * gradient_y
        value = f(x, y)
        history.append(value)
        print(f"Iteration {iteration + 1}: f({x:.4f}, {y:.4f}) = {value:.4f}")

    return history

# List of different step sizes
learning_rates = [0.01, 0.1, 0.5, 1]

# Number of iterations
num_iterations = 50

# Perform gradient descent with different step sizes and store results
results = {}
for lr in learning_rates:
    history = gradient_descent(lr, num_iterations)
    results[lr] = history

# Plot the value of the function versus iterations for each step size
plt.figure(figsize=(10, 6))
for lr in learning_rates:
    plt.plot(range(num_iterations), results[lr], label=f"Step Size {lr}")
plt.xlabel("Iterations")
plt.ylabel("f(x, y)")
plt.title("Convergence Rates of Gradient Descent with Different Step Sizes")
plt.legend()
plt.show()
```

```
Iteration 1: f(0.0900, -0.1100) = -1.9437
Iteration 2: f(0.1760, -0.2094) = -3.6073
Iteration 3: f(0.2583, -0.2991) = -5.0359
Iteration 4: f(0.3371, -0.3800) = -6.2670
Iteration 5: f(0.4128, -0.4529) = -7.3322
Iteration 6: f(0.4855, -0.5184) = -8.2578
Iteration 7: f(0.5554, -0.5772) = -9.0657
Iteration 8: f(0.6228, -0.6299) = -9.7744
Iteration 9: f(0.6877, -0.6771) = -10.3993
Iteration 10: f(0.7504, -0.7192) = -10.9532
Iteration 11: f(0.8110, -0.7566) = -11.4471
Iteration 12: f(0.8697, -0.7899) = -11.8898
Iteration 13: f(0.9265, -0.8193) = -12.2890
Iteration 14: f(0.9816, -0.8452) = -12.6511
Iteration 15: f(1.0350, -0.8680) = -12.9814
Iteration 16: f(1.0870, -0.8878) = -13.2843
Iteration 17: f(1.1375, -0.9051) = -13.5635
Iteration 18: f(1.1866, -0.9199) = -13.8224
Iteration 19: f(1.2345, -0.9326) = -14.0635
Iteration 20: f(1.2811, -0.9433) = -14.2890
Iteration 21: f(1.3267, -0.9522) = -14.5009
Iteration 22: f(1.3711, -0.9595) = -14.7007
Iteration 23: f(1.4145, -0.9653) = -14.8898
Iteration 24: f(1.4569, -0.9698) = -15.0694
Iteration 25: f(1.4983, -0.9731) = -15.2404
Iteration 26: f(1.5389, -0.9753) = -15.4036
Iteration 27: f(1.5786, -0.9765) = -15.5598
Iteration 28: f(1.6175, -0.9768) = -15.7096
Iteration 29: f(1.6556, -0.9763) = -15.8535
Iteration 30: f(1.6930, -0.9751) = -15.9919
Iteration 31: f(1.7296, -0.9732) = -16.1253
Iteration 32: f(1.7656, -0.9708) = -16.2539
Iteration 33: f(1.8009, -0.9678) = -16.3781
Iteration 34: f(1.8355, -0.9643) = -16.4982
Iteration 35: f(1.8695, -0.9605) = -16.6144
Iteration 36: f(1.9029, -0.9563) = -16.7268
Iteration 37: f(1.9357, -0.9517) = -16.8357
Iteration 38: f(1.9680, -0.9468) = -16.9413
Iteration 39: f(1.9997, -0.9417) = -17.0436
Iteration 40: f(2.0308, -0.9364) = -17.1429
Iteration 41: f(2.0615, -0.9309) = -17.2392
Iteration 42: f(2.0916, -0.9252) = -17.3327
Iteration 43: f(2.1213, -0.9193) = -17.4234
Iteration 44: f(2.1505, -0.9134) = -17.5115
Iteration 45: f(2.1792, -0.9073) = -17.5971
Iteration 46: f(2.2075, -0.9011) = -17.6802
Iteration 47: f(2.2353, -0.8949) = -17.7610
Iteration 48: f(2.2627, -0.8886) = -17.8394
Iteration 49: f(2.2897, -0.8822) = -17.9156
Iteration 50: f(2.3162, -0.8759) = -17.9897
Iteration 1: f(0.9000, -1.1000) = -12.5700
Iteration 2: f(1.4000, -1.1400) = -14.7896
Iteration 3: f(1.7920, -1.0480) = -16.2955
Iteration 4: f(2.1240, -0.9512) = -17.4080
Iteration 5: f(2.4090, -0.8654) = -18.2318
Iteration 6: f(2.6541, -0.7913) = -18.8419
Iteration 7: f(2.8650, -0.7274) = -19.2937
Iteration 8: f(3.0465, -0.6725) = -19.6283
Iteration 9: f(3.2027, -0.6252) = -19.8761
Iteration 10: f(3.3371, -0.5845) = -20.0596
Iteration 11: f(3.4528, -0.5495) = -20.1955
Iteration 12: f(3.5524, -0.5193) = -20.2961
Iteration 13: f(3.6380, -0.4934) = -20.3706
Iteration 14: f(3.7117, -0.4711) = -20.4258
Iteration 15: f(3.7752, -0.4519) = -20.4667
Iteration 16: f(3.8298, -0.4353) = -20.4969
Iteration 17: f(3.8767, -0.4211) = -20.5194
Iteration 18: f(3.9172, -0.4089) = -20.5360
Iteration 19: f(3.9520, -0.3983) = -20.5482
Iteration 20: f(3.9819, -0.3893) = -20.5573
Iteration 21: f(4.0077, -0.3815) = -20.5641
Iteration 22: f(4.0298, -0.3748) = -20.5691
Iteration 23: f(4.0489, -0.3690) = -20.5728
Iteration 24: f(4.0653, -0.3640) = -20.5755
Iteration 25: f(4.0795, -0.3597) = -20.5775
Iteration 26: f(4.0916, -0.3561) = -20.5790
Iteration 27: f(4.1021, -0.3529) = -20.5802
Iteration 28: f(4.1111, -0.3502) = -20.5810
Iteration 29: f(4.1188, -0.3478) = -20.5816
Iteration 30: f(4.1255, -0.3458) = -20.5820
Iteration 31: f(4.1313, -0.3441) = -20.5824
Iteration 32: f(4.1362, -0.3426) = -20.5826
Iteration 33: f(4.1404, -0.3413) = -20.5828
Iteration 34: f(4.1441, -0.3402) = -20.5829
```

```
Iteration 35: f(4.1472, -0.3392) = -20.5830
Iteration 36: f(4.1500, -0.3384) = -20.5831
Iteration 37: f(4.1523, -0.3377) = -20.5832
Iteration 38: f(4.1543, -0.3371) = -20.5832
Iteration 39: f(4.1560, -0.3366) = -20.5832
Iteration 40: f(4.1575, -0.3361) = -20.5833
Iteration 41: f(4.1588, -0.3357) = -20.5833
Iteration 42: f(4.1599, -0.3354) = -20.5833
Iteration 43: f(4.1608, -0.3351) = -20.5833
Iteration 44: f(4.1616, -0.3349) = -20.5833
Iteration 45: f(4.1623, -0.3346) = -20.5833
Iteration 46: f(4.1629, -0.3345) = -20.5833
Iteration 47: f(4.1635, -0.3343) = -20.5833
Iteration 48: f(4.1639, -0.3342) = -20.5833
Iteration 49: f(4.1643, -0.3341) = -20.5833
Iteration 50: f(4.1646, -0.3340) = -20.5833
Iteration 1: f(4.5000, -5.5000) = 89.7500
Iteration 2: f(-1.0000, 15.5000) = 1172.5000
Iteration 3: f(20.0000, -53.0000) = 12993.0000
Iteration 4: f(-48.5000, 173.5000) = 141935.7501
Iteration 5: f(178.0000, -574.5000) = 1548485.5028
Iteration 6: f(-570.0000, 1896.0000) = 16891589.7685
Iteration 7: f(1900.4999, -6263.4999) = 184259183.6880
Iteration 8: f(-6258.9997, 20685.4997) = 2009959612.9961
Iteration 9: f(20690.0007, -68321.0017) = 21925298640.8027
Iteration 10: f(-68316.5186, 225647.6758) = 239168665349.5779
Iteration 11: f(225652.4450, -745264.4946) = 2608926975646.9854
Iteration 12: f(-745258.1996, 2461424.9585) = 28458689138205.4102
Iteration 13: f(2461480.0817, -8129493.0102) = 310434569267427.8750
Iteration 14: f(-8130707.4183, 26848631.9898) = 3386101673099109.0000
Iteration 15: f(26844292.5817, -88663868.0102) = 36925978418047816.0000
Iteration 16: f(-88555707.4183, 292736131.9898) = 402466799733246400.0000
Iteration 17: f(293844292.5817, -968063868.0102) = 4403855150116116992.0000
Iteration 18: f(-960555707.4183, 3204736131.9898) = 48160657178378395648.0000
Iteration 19: f(3340244292.5817, -10312063868.0102) = 505401501622037643264.0000
Iteration 20: f(-9766955707.4183, 32286336131.9898) = 4895701857456957161472.0000
Iteration 21: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 22: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 23: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 24: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 25: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 26: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 27: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 28: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 29: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 30: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 31: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 32: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 33: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 34: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 35: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 36: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 37: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 38: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 39: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 40: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 41: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 42: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 43: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 44: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 45: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 46: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 47: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 48: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 49: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 50: f(42661844292.5817, -151214463868.0103) = 1061854651126028173312000.0000
Iteration 1: f(9.0000, -11.0000) = 561.0000
Iteration 2: f(-22.0000, 84.0000) = 33526.0000
Iteration 3: f(199.0000, -643.0000) = 1940447.0009
Iteration 4: f(-1476.0000, 4888.0000) = 112245179.9411
Iteration 5: f(11261.0007, -37178.9995) = 6492757324.4872
Iteration 6: f(-85609.9938, 282764.1424) = 375570065850.2931
Iteration 7: f(651148.4290, -2150582.4152) = 21724680380512.6562
Iteration 8: f(-4951976.5710, 16356448.8348) = 1256649473217955.0000
Iteration 9: f(37660523.4290, -124406051.1652) = 72696169592853408.0000
Iteration 10: f(-286339476.5710, 945993948.8348) = 4203359338051462144.0000
Iteration 11: f(2171260523.4290, -7220406051.1652) = 244606191580493873152.0000
Iteration 12: f(-15851139476.5710, 51761993948.8348) = 12609447864991393775616.0000
Iteration 13: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
Iteration 14: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
Iteration 15: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
Iteration 16: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
Iteration 17: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
Iteration 18: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
Iteration 19: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400.0000
```

```
Iteration 20: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 21: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 22: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 23: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 24: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 25: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 26: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 27: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 28: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 29: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 30: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 31: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 32: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 33: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 34: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 35: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 36: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 37: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 38: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 39: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 40: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 41: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 42: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 43: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 44: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 45: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 46: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 47: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 48: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 49: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
Iteration 50: f(-15851139476.5710, -262810806051.1652) = 2681976362436760331878400000
```

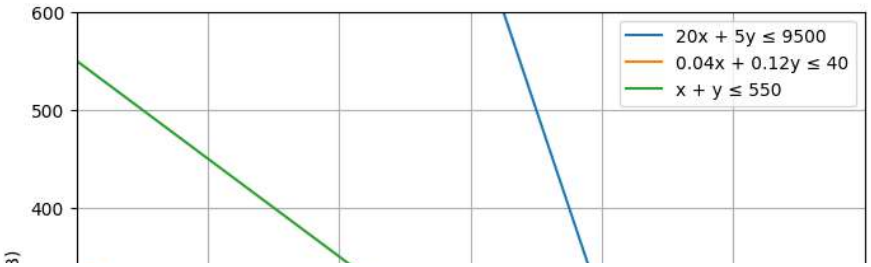



```
#Ngo Le Thien An - ITITDK21030
#Q2_D
import numpy as np

# Define the function f(x, y)
def f(x, y):
    return -9 * x + x**2 + 11 * y + 4 * y**2 - 2 * x * y

# Calculate the gradient of the function
def gradient(x, y):
    df_dx = 2 * x - 9 - 2 * y
    df_dy = 11 + 8 * y - 2 * x
    return np.array([df_dx, df_dy])

# Calculate the Hessian matrix
def hessian(x, y):
    df2_dx2 = 2
    df2_dxdy = -2
    df2_dy2 = 8
    return np.array([[df2_dx2, df2_dxdy], [df2_dxdy, df2_dy2]])

# Perform Newton's method
def newton_optimization(initial_guess, alpha, num_iterations):
    x, y = initial_guess
    history = []  # To store the function values at each iteration

    for _ in range(num_iterations):
        grad = gradient(x, y)
        hess_inv = np.linalg.inv(hessian(x, y))
        direction = np.dot(hess_inv, grad)
        x -= alpha * direction[0]
        y -= alpha * direction[1]
        value = f(x, y)
        history.append(value)
```

```
            print(f"f({x:.4f}, {y:.4f}) = {value:.4f}")

        return history

    # Initial guesses
    initial_guess = [0, 0]

    # Step size (adjust as needed)
    alpha = 1

    # Number of iterations
    num_iterations = 5

    # Perform Newton's method
    history = newton_optimization(initial_guess, alpha, num_iterations)
```

```
f(4.1667, -0.3333) = -20.5833
f(4.1667, -0.3333) = -20.5833
f(4.1667, -0.3333) = -20.5833
f(4.1667, -0.3333) = -20.5833
f(4.1667, -0.3333) = -20.5833
```

```
#Ngo Le Thien An - ITITDK21030
#Q3_B
import matplotlib.pyplot as plt
import numpy as np

# Define the constraints
x = np.linspace(0, 600, 400)
y1 = -4 * x + 1900
y2 = -1/3 * x + 1000/3
y3 = -x + 550

# Plot the constraints
plt.figure(figsize=(8, 6))
plt.plot(x, y1, label='20x + 5y ≤ 9500')
plt.plot(x, y2, label='0.04x + 0.12y ≤ 40')
plt.plot(x, y3, label='x + y ≤ 550')

# Fill the feasible region
plt.fill_between(x, 0, np.minimum(np.minimum(y1, y2), y3), where=(0 <= x) & (x <= 600), color='gray', alpha=0.5)

# Set non-negativity constraints
plt.xlim(0, 600)
plt.ylim(0, 600)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)

# Label axes and constraints
plt.xlabel('x (Product A)')
plt.ylabel('y (Product B)')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```




```
#Ngo Le Thien An - ITITDK21030
#Q3_C
import numpy as np
from scipy.optimize import linprog

# Objective function coefficients
c = [-45, -20, 0, 0, 0]

# Coefficients matrix for constraints
A = np.array([
    [20, 5, 1, 0, 0],    # Raw Material Constraint
    [0.04, 0.12, 0, 1, 0],    # Production Time Constraint
    [1, 1, 0, 0, 1]    # Storage Capacity Constraint
])

# RHS values
b = [9500, 40, 550]

# Variable bounds (x and y should be non-negative by default)
x_bounds = (0, None)
y_bounds = (0, None)

# Solve the linear programming problem
result = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds, (0, None), (0, None), (0, None)])

# Extract and print the results
if result.success:
    print("Optimal Solution:")
    x_optimal, y_optimal = result.x[0], result.x[1]
    optimal_profit = -result.fun  # Negative sign because linprog minimizes by default
    print(f"x (Product A) = {x_optimal:.2f}")
    print(f"y (Product B) = {y_optimal:.2f}")
    print(f"Optimal Profit = ${optimal_profit:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal Solution:
x (Product A) = 450.00
y (Product B) = 100.00
Optimal Profit = $22250.00
```

```
#Ngo Le Thien An - ITITDK21030
#Q3_D
import cvxpy as cp

# Define the decision variables
x = cp.Variable()
y = cp.Variable()

# Define the objective function to maximize
objective = cp.Maximize(45 * x + 20 * y)

# Define the constraints
constraints = [
    20 * x + 5 * y <= 9500,  # Raw Material Constraint
    0.04 * x + 0.12 * y <= 40,  # Production Time Constraint
    x + y <= 550,  # Storage Capacity Constraint
    x >= 0,  # Non-Negativity Constraint for x
    y >= 0,  # Non-Negativity Constraint for y
]

# Create the linear programming problem
problem = cp.Problem(objective, constraints)
```

```
# Solve the problem
problem.solve()

# Print the results
if problem.status == cp.OPTIMAL:
    print("Optimal Solution:")
    print(f"x (Product A) = {x.value:.2f}")
    print(f"y (Product B) = {y.value:.2f}")
    print(f"Optimal Profit = ${problem.value:.2f}")
else:
    print("No optimal solution found.")
```

```
Optimal Solution:
x (Product A) = 450.00
y (Product B) = 100.00
Optimal Profit = $22250.00
```

```
#Ngo Le Thien An - ITITDK21030
#Q3_E
import cvxpy as cp

# Define the decision variables
x = cp.Variable()
y = cp.Variable()

# Define the objective function to maximize
objective = cp.Maximize(45 * x + 20 * y)

# Define the constraints
constraints = [
    20 * x + 5 * y <= 9500,  # Raw Material Constraint
    0.04 * x + 0.12 * y <= 40,  # Production Time Constraint
    x + y <= 550,  # Storage Capacity Constraint
    x >= 0,  # Non-Negativity Constraint for x
    y >= 0,  # Non-Negativity Constraint for y
]

# Create the linear programming problem
problem = cp.Problem(objective, constraints)

# Solve the problem to obtain the optimal solution
problem.solve()

if problem.status == cp.OPTIMAL:
    # Calculate and print the shadow prices (dual values)
    shadow_price_raw_material = constraints[0].dual_value
    shadow_price_production_time = constraints[1].dual_value
    shadow_price_storage_capacity = constraints[2].dual_value

    print(f"Shadow Price (Raw Material): ${shadow_price_raw_material:.2f} per kg")
    print(f"Shadow Price (Production Time): ${shadow_price_production_time:.2f} per hour")
    print(f"Shadow Price (Storage Capacity): ${shadow_price_storage_capacity:.2f} per unit of storage")

    # Determine which option raises profits the most based on the signs of shadow prices
    if shadow_price_raw_material > 0:
        print("Increasing raw material will raise profits.")
    if shadow_price_production_time > 0:
        print("Increasing production time will raise profits.")
    if shadow_price_storage_capacity > 0:
        print("Increasing storage capacity will raise profits.")
else:
    print("No optimal solution found.")
```

```
Shadow Price (Raw Material): $1.67 per kg
Shadow Price (Production Time): $0.00 per hour
Shadow Price (Storage Capacity): $11.67 per unit of storage
Increasing raw material will raise profits.
Increasing production time will raise profits.
Increasing storage capacity will raise profits.
```