NUMERICAL OPTIMISATION

PRACTICAL FILE

Bhaskar malik

16020

Section a

**Q.1**

import numpy as np

def objective\_function(x):

    return x\*\*2 + 4\*x + 4

def gradient(x):

    return 2\*x + 4

def line\_search(initial\_x, learning\_rate, epsilon):

    x = initial\_x

    iteration = 0

    while True:

        gradient\_x = gradient(x)

        new\_x = x - learning\_rate \* gradient\_x

        # Check for convergence

        if abs(new\_x - x) < epsilon:

            break

        x = new\_x

        iteration += 1

    return x, objective\_function(x), iteration

# Initial parameters

initial\_x = 0.0

learning\_rate = 0.1

epsilon = 1e-6

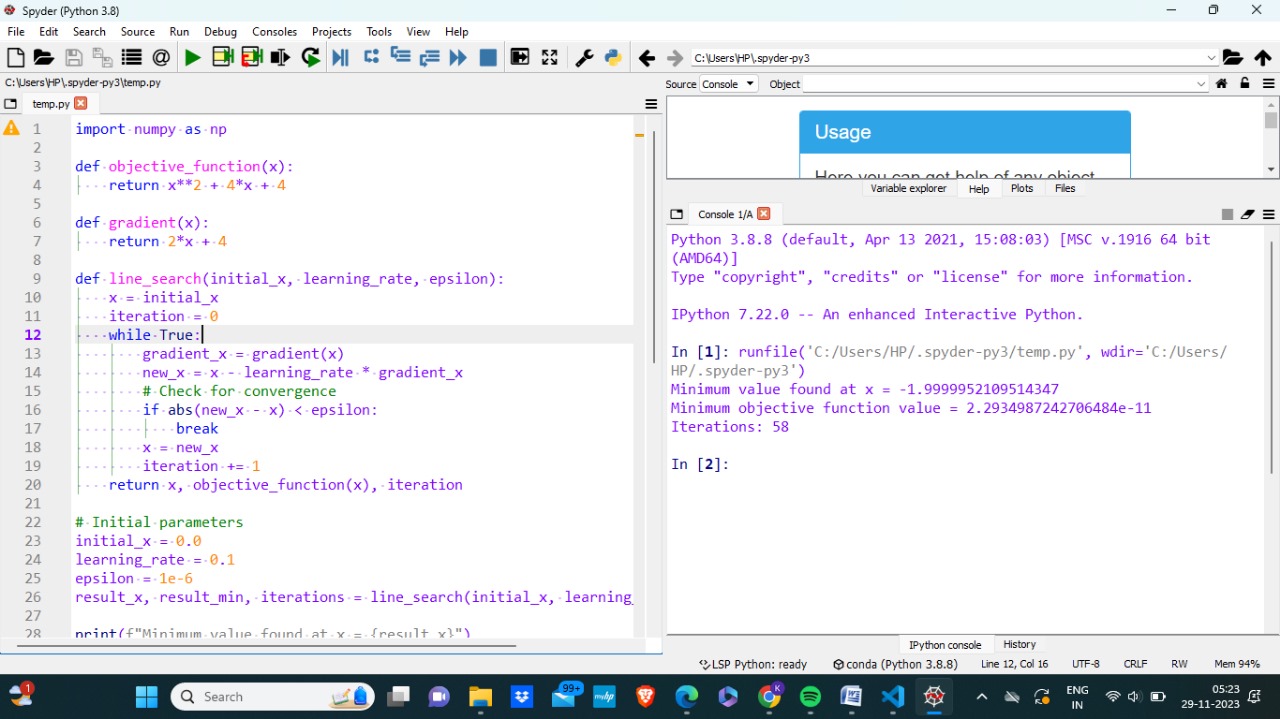
result\_x, result\_min, iterations = line\_search(initial\_x, learning\_rate, epsilon)

print(f"Minimum value found at x = {result\_x}")

print(f"Minimum objective function value = {result\_min}")

print(f"Iterations: {iterations}")

**OUTPUT:**



**Q.2**

import matplotlib.pyplot as plt

# define constants of function

c1=4

c2=5

points=[[0,0],[4,0],[0,3],[2,1]]

z=[]

for i in points:

value=c1\*i[0]+c2\*i[1]

z.append(value)

print("value at ",i,"is : ",value)

Max=points[z.index(max(z))]

print("optimal point is :",Max)

# x and y are define

x = [i for i in range(0,10)]

y = [(6-2\*i) for i in x]

y1= [(9-2\*i) for i in x]

# plot x and y

plt.plot(x, y)

plt.plot(x, y1)

# to make grid in graph

plt.grid(True)

# to limit axis of x and y

plt.xlim(0,12)

plt.ylim(-26,13)

# to stlye line in figure

plt.plot(x,y,ls="dotted",marker="s",c="b",lw="2")

plt.plot(x,y1,ls="dotted",marker=".",c="r",lw="2")

# to label axis

plt.xlabel('x - axis')

plt.ylabel('y - axis')

# to label graph

plt.title('My first graph!')

# to show graph

plt.show()

**OUTPUT:** A screenshot of a computer

Description automatically generated

**Q.3**

import sympy as sp

x1,x2=sp.symbols ('x1 x2')

function=100\* (x2-x1\*2) \*\*2+(1-x1)\*2

gradient=[sp.diff (function, x1), sp.diff (function, x2)]

hessian=[[sp.diff (gradient [0], x1), sp.diff (gradient [0],x2)], [sp.diff (gradient [1], x1), sp.diff (gradient[1],x2)]]

print("Gradient:")

print(gradient)

print("\nHessian:")

print(hessian)

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**Q.4**

import numpy as np

from scipy.optimize import differential\_evolution

def objective\_function(x):

    return -10 \* np.cos(np.pi \* x - 2.2) + (x + 1.5) \* x

bounds = [(-10, 10)]

result = differential\_evolution(objective\_function, bounds)

min\_x = result.x

global\_min\_val = result.fun

print("Global Optimal Solution:")

print(f"x = {min\_x[0]}")

print(f"f(x) = {global\_min\_val}")

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**Q.5**

import numpy as np

import matplotlib.pyplot as plt

# Define the function f(x)

def objective\_function(x):

return -10 \* np.cos(np.pi \* x - 2.2) + (x + 1.5) \* x

# Generate x values

x = np.linspace(-5, 5, 20)

print(x)

y = objective\_function(x)

print(y)

plt.plot(x, y, label='f(x) = -10Cos(pi x - 2.2) + (x + 1.5) \* x')

plt.xlabel('x')

plt.ylabel('f(x)')

plt.title(' Function f(x)')

plt.grid(True)

min\_y = min(y)

min\_x = x[np.argmin(y)]

plt.scatter(min\_x, min\_y, color='blue', label=f'Minimum: ({min\_x}, {min\_y})')

plt.legend()

plt.show()

print("Global optimal solution is", min\_x)

print(" Optimal function value is", min\_y)

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**Q.6**

import pulp

import matplotlib.pyplot as plt

lp\_problem = pulp.LpProblem("LPP", pulp.LpMaximize)

x = pulp.LpVariable("x", lowBound=0)

y = pulp.LpVariable("y", lowBound=0)

lp\_problem += 3 \* x + 2 \* y

lp\_problem += x <= 4

lp\_problem += y <= 6

lp\_problem += 2 \* x + y <= 12

lp\_problem.solve()

print("Status:", pulp.LpStatus[lp\_problem.status])

print("x =", x.varValue)

print("y =", y.varValue)

print("Optimal Value =", pulp.value(lp\_problem.objective))

plt.plot(x.varValue, y.varValue, 'ro', label="Optimal Value")

plt.fill([0, 4, 4, 3, 0], [0, 0, 4, 6, 6], 'b', alpha=0.2)

plt.xlabel("x")

plt.ylabel("y")

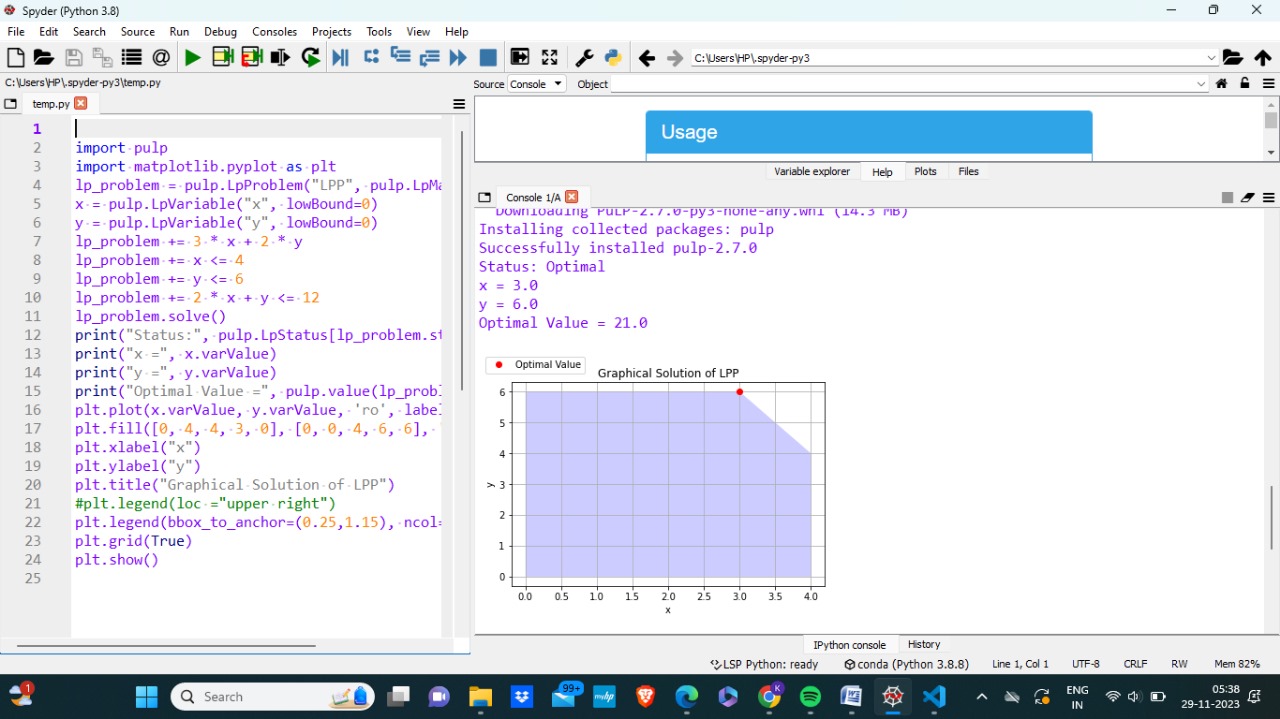
plt.title("Graphical Solution of LPP")

#plt.legend(loc ="upper right")

plt.legend(bbox\_to\_anchor=(0.25,1.15), ncol=7)

plt.grid(True)

**OUTPUT:**



import numpy as np

from matplotlib import pyplot as plt

plt.rcParams["figure.figsize"] = [10.50, 3.50]

plt.rcParams["figure.autolayout"] = True

c1=2

c2=3

def f(x):

return x\*\*2

x = np.linspace(-5, 1, 5)

z=[]

for i in x:

value=c1\*i+c2\*f(x)

z.append(value)

print("value at ",i,"is : ",value)

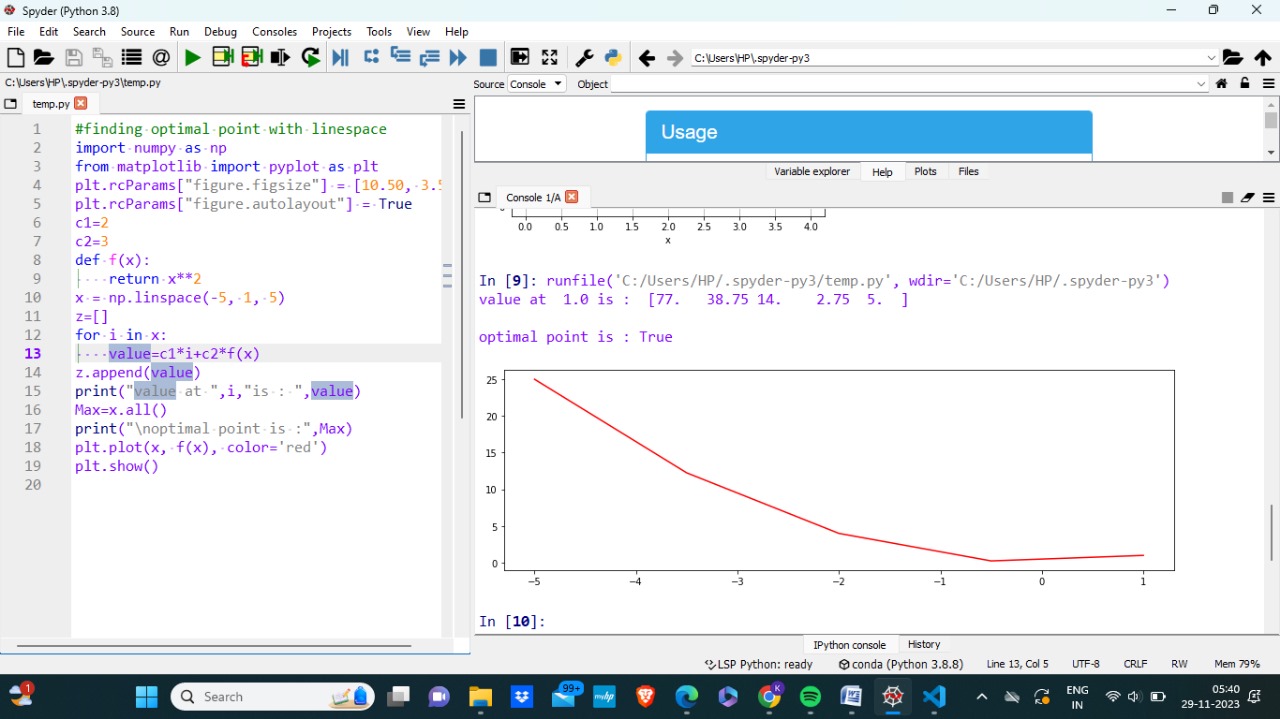
Max=x.all()

print("\noptimal point is :",Max)

plt.plot(x, f(x), color='red')

plt.show()

**OUTPUT:**



**Q.7**

from sympy import symbols, diff, solve, Matrix

import numpy as np

from scipy.optimize import minimize

# Define symbols

x, y, l = symbols('x y lambda')

# Define the objective function and constraint

f = x\*\*2 + y\*\*2

g = x + y - 1

# Define the Lagrangian

L = f - l \* g

# Compute partial derivatives

partials = [diff(L, var) for var in (x, y, l)]

# Solve the system of equations

solution = solve(partials, (x, y, l), dict=True)[0]

# Extract the optimal values

optimal\_x = solution[x]

optimal\_y = solution[y]

# Compute the Hessian matrix

hessian\_list = [[diff(L.diff(var1), var2) for var1 in (x, y, l)] for var2 in (x, y, l)]

hessian\_matrix = Matrix(hessian\_list)

# Display the Hessian matrix

print("Hessian Matrix:")

print(hessian\_matrix)

# Determine the nature of the stationary point

hessian\_determinant = hessian\_matrix.det()

if hessian\_determinant > 0:

    print("Stationary point is a local minimum.")

elif hessian\_determinant < 0:

    print("Stationary point is a local maximum.")

else:

    print("Second-order test inconclusive (saddle point or test fails)")

# Display the optimal solution

print("\nOptimal solution:")

print(f"x: {optimal\_x}")

print(f"y: {optimal\_y}")

# Finding optimal solution using scipy.optimize.minimize

# Define the objective function

def objective\_function(x):

    return x[0]\*\*2 + x[1]\*\*2

# Initial guess

initial\_guess = [0.0, 0.0]

# Minimize the objective function

result = minimize(objective\_function, initial\_guess, method='BFGS')

# Extract the optimal solution

optimal\_solution = result.x

minimum\_value = result.fun

# Print the results

print("\nOptimal Solution (using scipy.optimize.minimize):", optimal\_solution)

print("Minimum Value:", minimum\_value)

**OUTPUT:**

