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)

2 import numpy as np

from scipy.optimize import differential\_evolution

defobjective\_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 min x: ",min\_x)

print("Global Optimal Solution:")

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

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

3 import numpy as np

defobjective\_function(x):

return x\*\*2 + 4\*x + 4

def gradient(x):

return 2\*x + 4

defline\_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}")

4 from sympy import symbols, diff, solve, Matrix

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

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

# Compute the Hessian matrix using a list of lists

hessian\_list = []

# Iterate over var2

for var2 in (x, y, l):

# Initialize a row for var2

row = []

# Iterate over var1

for var1 in (x, y, l):

# Calculate the second-order partial derivative and append to the row

row.append(diff(L.diff(var1), var2))

# Append the row to the Hessian list

hessian\_list.append(row)

# Create an instance of the Matrix class from the list of lists

hessian\_matrix = Matrix(hessian\_list)

# Display the Hessian matrix

print(hessian\_matrix)

hessian\_determinant = hessian\_matrix.det()

if hessian\_determinant> 0:

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

elifhessian\_determinant< 0:

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

else:6

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

# Display the result

print("Optimal solution:")

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

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

5!pip install pulp

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()

plt.grid(True)

plt.show()