Sahakar Maharshi Bhausaheb Santuji Thorat

College Sangamner

DEPARTMENT OF COMPUTER SCIENCE

MATHEMATICS

Remark
Demonstrators
Signature
Date :- / /2023

Name :- Prem Vijay Vajare **Batch No. :- D** **Roll No:- 75** **Date:-** / /2023

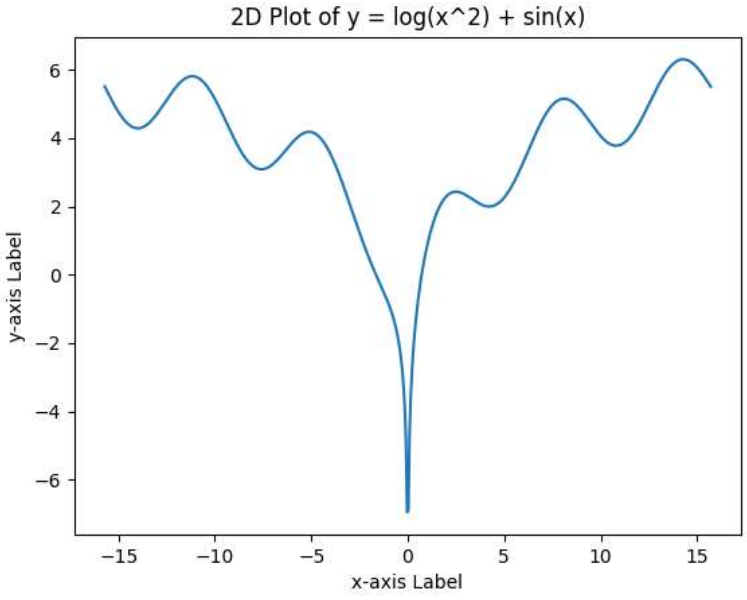
Title of the:- Practical 22 **Expt. No .** 22 **Class :-** S.Y.BCS

Q.1) Write a python program to draw 2D plot y = log(x^2) + sin(x) with suitable label in the x axis , y axis and a tittle in [-5*pi,5*pi]

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate x values from -5*pi to 5*pi
x = np.linspace(-5 * np.pi, 5 * np.pi, 500)
# Compute y values using the given function
y = np.log(x**2) + np.sin(x)
# Create the plot
plt.plot(x, y)
plt.xlabel('x-axis Label') # Label for x-axis
plt.ylabel('y-axis Label') # Label for y-axis
plt.title('2D Plot of y = log(x^2) + sin(x)') # Title of the plot
# Show the plot
plt.show()
```

OUTPUT:

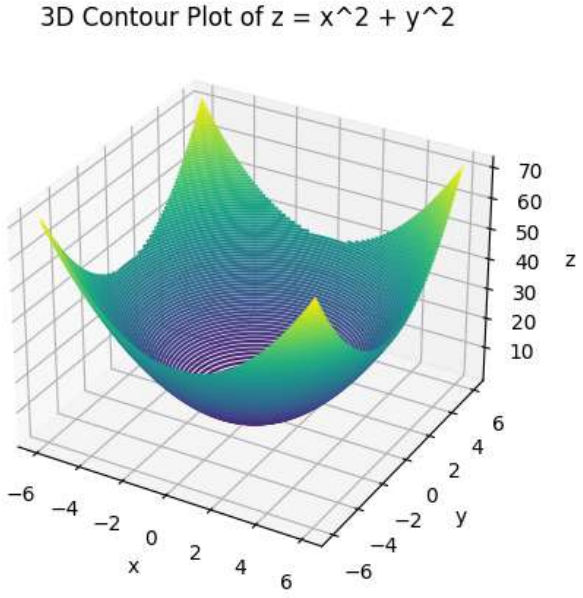


Q.2) Write a python program to plot 3D dimensional Contour plot of parabola z = x^2 + y^2 , -6 < x,y < 6

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
# Generate x and y values
x = np.linspace(-6, 6, 100)
y = np.linspace(-6, 6, 100)
# Create a grid of x and y values
X, Y = np.meshgrid(x, y)
# Compute z values using the given function
Z = X**2 + Y**2
# Create a 3D contour plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.contour3D(X, Y, Z, 100, cmap='viridis')
# Set labels and title
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D Contour Plot of z = x^2 + y^2')
# Show the plot
plt.show()
```

OUTPUT:

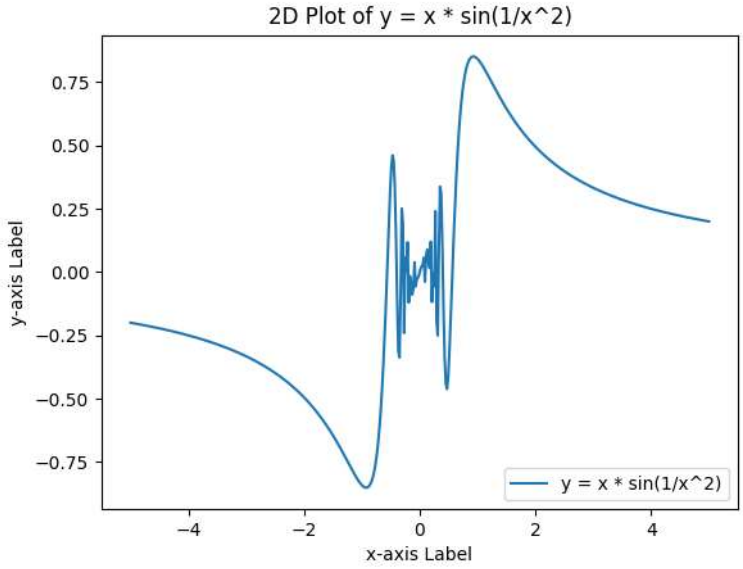


Q.3) Write a python program to draw 2D plot y = x sin(1/x^2) in [-5,5] with suitable label in the x axis, y axis a title and location of legend to lower right corner.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate x values from -5 to 5
x = np.linspace(-5, 5, 500)
# Compute y values using the given function
y = x * np.sin(1 / x**2)
# Create the plot
plt.plot(x, y, label='y = x * sin(1/x^2)')
plt.xlabel('x-axis Label') # Label for x-axis
plt.ylabel('y-axis Label') # Label for y-axis
plt.title('2D Plot of y = x * sin(1/x^2)') # Title of the plot
plt.legend(loc='lower right') # Legend in lower-right corner
# Show the plot
plt.show()
```

OUTPUT:



Q.4) Write n python program to find the angle at each vertices of the triangle ABC where A[0,0] B[2,2] and C[0,2].

Syntax:

```
import numpy as np
# Given coordinates of points A, B, and C
A = np.array([0, 0])
B = np.array([2, 2])
C = np.array([0, 2])
# Compute the sides of the triangle
AB = np.linalg.norm(B - A)  # Length of side AB
BC = np.linalg.norm(C - B)  # Length of side BC
AC = np.linalg.norm(C - A)  # Length of side AC
# Compute the angles using the Law of Cosines
angle_A = np.arccos((AB**2 + AC**2 - BC**2) / (2 * AB * AC))
angle_B = np.arccos((-AB**2 + AC**2 + BC**2) / (2 * AC * BC))
angle_C = np.pi - angle_A - angle_B
# Convert angles from radians to degrees
angle_A_deg = np.degrees(angle_A)
angle_B_deg = np.degrees(angle_B)
angle_C_deg = np.degrees(angle_C)
# Print the angles in degrees
print("Angle at vertex A: {:.2f} degrees".format(angle_A_deg))
print("Angle at vertex B: {:.2f} degrees".format(angle_B_deg))
```

print("Angle at vertex C: {:.2f} degrees".format(angle_C_deg))
Output:
Angle at vertex A: 45.00 degrees
Angle at vertex B: 90.00 degrees
Angle at vertex C: 45.00 degrees

Q.5) Write a Python program to Reflect the Point P[3,6] through the line x - 2y + 4 = 0.

Syntax:

```
import numpy as np
# Given point P
P = np.array([3, 6])
# Given line parameters
a = 1  # Coefficient of x in the line equation
b = -2  # Coefficient of y in the line equation
c = 4  # Constant term in the line equation
# Compute the perpendicular distance from point P to the line
dist = abs(a * P[0] + b * P[1] + c) / np.sqrt(a**2 + b**2)
# Compute the reflected point using the formula
reflected_x = P[0] - 2 * a * dist / (a**2 + b**2)
reflected_y = P[1] - 2 * b * dist / (a**2 + b**2)
# Create the reflected point as a NumPy array
reflected_P = np.array([reflected_x, reflected_y])
# Print the coordinates of the reflected point
print("Reflected Point: ({:.2f}, {:.2f})".format(reflected_P[0], reflected_P[1]))
```

OUTPUT:

Reflected Point: (2.11, 7.79)

Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[5, 0], C[3,3].

Synatx:

```
import numpy as np
# Define the vertices of the triangle
A = np.array([0, 0])
```

```
B = np.array([5, 0])
C = np.array([3, 3])
# Calculate the side lengths of the triangle
AB = np.linalg.norm(B - A)
BC = np.linalg.norm(C - B)
CA = np.linalg.norm(A - C)
# Calculate the semiperimeter
s = (AB + BC + CA) / 2
# Calculate the area using Heron's formula
area = np.sqrt(s * (s - AB) * (s - BC) * (s - CA))
# Calculate the perimeter
perimeter = AB + BC + CA
# Print the results
print("Triangle ABC:")
print("Side AB:", AB)
print("Side BC:", BC)
print("Side CA:", CA)
print("Area:", area)
print("Perimeter:", perimeter)
```

OUTPUT:

```
Triangle ABC:
Side AB: 5.0
Side BC: 3.605551275463989
Side CA: 4.242640687119285
Area: 7.5000000000000036
Perimeter: 12.848191962583275
```

Q.7) write a Python program to solve the following LPP

Max Z = x + y

Subjected to

x + y <= 11

x >= 6

y >= 6

x > 0, y > 0

Syntax:

```
from pulp import *

# Create the problem

prob = LpProblem("Maximization Problem", LpMaximize)

# Define the variables

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

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

# Define the objective function

prob += x + y, "Objective Function"

# Define the constraints

prob += x + y <= 11, "Constraint 1"

prob += x >= 6, "Constraint 2"

prob += y >= 6, "Constraint 3"

# Solve the problem

prob.solve()

# Print the results

print("Status: ", LpStatus[prob.status])

print("Optimal Solution:")

print("x = ", value(x))

print("y = ", value(y))

print("Max Z = ", value(prob.objective))
```

OUTPUT:

Status: Infeasible

Optimal Solution:

x = 5.0

y = 6.0

Max Z = 11.0

Q.8) Write a python program to display the following LPP by using pulp module and simplex method. Find its optimal solution if exist.

Min Z = 4x+y+3z+5w
subject to
4x+6y-5z-4w >= -20
-8x-3y+3z+2w <= 20
-3x -2y + 4z + w <= 10
x >= 0,y>= 0,z>= 0,w>= 0

Syntax:

#BY using Pulp Module

```
from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpStatus, value
# Create the LPP problem
problem = LpProblem("LPP", LpMinimize)
# Define the variables
x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0)
z = LpVariable("z", lowBound=0)
w = LpVariable("w", lowBound=0)
# Define the objective function
objective = 4 * x + y + 3 * z + 5 * w
problem += objective
# Define the constraints
constraint1 = 4 * x + 6 * y - 5 * z - 4 * w >= -20
constraint2 = -8 * x - 3 * y + 3 * z + 2 * w <= 20
constraint3 = -3*x -2*y + 4*z + w  <= 10
problem += constraint1
problem += constraint2
problem += constraint3
# Solve the LPP problem using the simplex method
```

```
problem.solve()
# Check if the optimization was successful
if LpStatus[problem.status] == "Optimal":
    print("Optimal solution found:")
    print("x =", value(x))
    print("y =", value(y))
    print("z =", value(z))
    print("w =", value(w))
    print("Minimum value of Z =", value(objective))
else:
    print("Optimization failed.")
```

OUTPUT :

```
Optimal solution found:
x = 0.0
y = 0.0
z = 0.0
w = 0.0
Minimum value of Z = 0.0
```

Q.9) Write the python program for each of the following
(I)Rotate the point (1,1) about (1,4) through angle pi / 2
(II) Find Distance between two points (0,0) and (1,0)
(III) Find the shearing of the point (3,4) in X direction by 3 units.
(IV) Represent two dimensional points using point function (-2,5)
Syntax:

```
import math
# Rotate point (x, y) about (cx, cy) through angle theta (in radians)
def rotate_point(x, y, cx, cy, theta):
    dx = x - cx
    dy = y - cy
    cos_theta = math.cos(theta)
    sin_theta = math.sin(theta)
    new_x = cx + dx * cos_theta - dy * sin_theta
    new_y = cy + dx * sin_theta + dy * cos_theta
    return new_x, new_y
# Find distance between two points (x1, y1) and (x2, y2)
def distance_between_points(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
# Shear point (x, y) in X direction by shx units
```

```
def shear_point_x(x, y, shx):
    new_x = x + shx * y
    return new_x, y
# Define a class for representing two-dimensional points
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
# Rotate point (1,1) about (1,4) through angle pi/2
x1, y1 = rotate_point(1, 1, 1, 4, math.pi / 2)
print("Rotated point: ({}, {})".format(x1, y1))
# Find distance between two points (0,0) and (1,0)
x2, y2 = 0, 0
x3, y3 = 1, 0
distance = distance_between_points(x2, y2, x3, y3)
print("Distance between points: {}".format(distance))
# Find the shearing of the point (3,4) in X direction by 3 units
x4, y4 = shear_point_x(3, 4, 3)
print("Sheared point: ({}, {})".format(x4, y4))
# Represent two-dimensional points using Point class
p = Point(-2, 5)
print("Point: ({}, {})".format(p.x, p.y))
```

OUTPUT:
Rotated point: (4.0, 4.0)
Distance between points: 1.0
Sheared point: (15, 4)
Point: (-2, 5)

Q.10) A company has 3 production facilities S1, S2, and S3 with production capacity of 7,9 and 18 units (in 100's) per week of a product, respectively. These units are to be shipped to 4 warehouse D1,D2,D3,D4 with requirement of 5,6,7 and 14 units (in 100's) per week, respectively. The transportation costs (in rupee) per units between factories to warehouse are given in the table below.

	D1	D2	D3	D4	Supply
S1	19	30	50	10	7
S2	70	30	40	60	9
S3	40	8	70	20	18
Demand	5	8	7	14	34

Write a python program to solve transportation problem for minimizing the costs of whole operation.

Syntax:

```
from pulp import *
# Define the factories, warehouses, and their respective capacities
factories = ['S1', 'S2', 'S3']
warehouses = ['D1', 'D2', 'D3', 'D4']
capacities = {'S1': 7, 'S2': 9, 'S3': 18}
requirements = {'D1': 5, 'D2': 6, 'D3': 7, 'D4': 14}
# Define the transportation costs between factories and warehouses
transportation_costs = {
    ('S1', 'D1'): 19, ('S1', 'D2'): 30, ('S1', 'D3'): 50, ('S1', 'D4'): 10,
    ('S2', 'D1'): 70, ('S2', 'D2'): 30, ('S2', 'D3'): 40, ('S2', 'D4'): 60,
    ('S3', 'D1'): 40, ('S3', 'D2'): 8, ('S3', 'D3'): 70, ('S3', 'D4'): 20
}
# Crete a binary variable to represent the shipment decision
shipment = LpVariable.dicts('Shipment', (factories, warehouses), lowBound=0, cat='Integer')
# Create the LP problem
prob = LpProblem('Transportation Problem', LpMinimize)
# Define the objective function
prob += lpSum(shipment[f][w] * transportation_costs[f, w] for f in factories for w in warehouses)
# Add the capacity constraints for factories
for f in factories:
    prob += lpSum(shipment[f][w] for w in warehouses) <= capacities[f]
# Add the demand constraints for warehouses
for w in warehouses:
    prob += lpSum(shipment[f][w] for f in factories) >= requirements[w]
# Solve the LP problem
prob.solve()
# Print the optimal solution
print('Optimal Solution:')
for f in factories:
    for w in warehouses:
        if shipment[f][w].varValue > 0:
            print('Ship {} units from {} to {}'.format(shipment[f][w].varValue, f, w))
# Print the total cost of the optimal solution
print('Total Cost: Rs. {}'.format(value(prob.objective)))
```

OUTPUT:
Optimal Solution:
Ship 5.0 units from S1 to D1
Ship 2.0 units from S1 to D4
Ship 7.0 units from S2 to D3
Ship 6.0 units from S3 to D2
Ship 12.0 units from S3 to D4
Total Cost: Rs. 683.0