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:- 04** **Date:-** / /2023

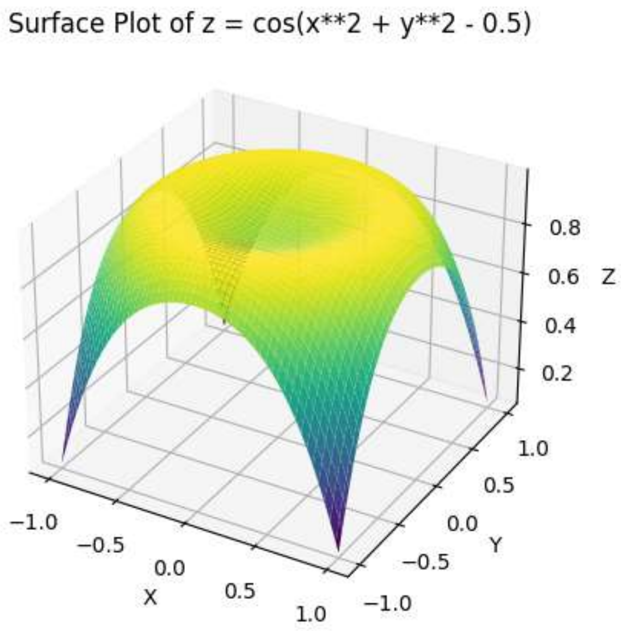
Title of the:- Practical 5 **Expt. No .** 5 **Class :-** S.Y.BCS

Q.1) Using Python plot the surface plot of function z = cos (x**2 + y**2 - 0.5) in the interval from -1 < x,y < 1.

Syntax:

```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

# Define the function

def func(x, y):

    return np.cos(x**2 + y**2 - 0.5)

# Generate x, y values in the interval from -1 to 1

x = np.linspace(-1, 1, 100)

y = np.linspace(-1, 1, 100)

X, Y = np.meshgrid(x, y)  # Create a grid of x, y values

Z = func(X, Y)  # Compute z values using the function

# Create a 3D plot

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, Z, cmap='viridis')  # Plot the surface

ax.set_xlabel('X')

ax.set_ylabel('Y')

ax.set_zlabel('Z')

ax.set_title('Surface Plot of z = cos(x**2 + y**2 - 0.5)')

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

OUTPUT:

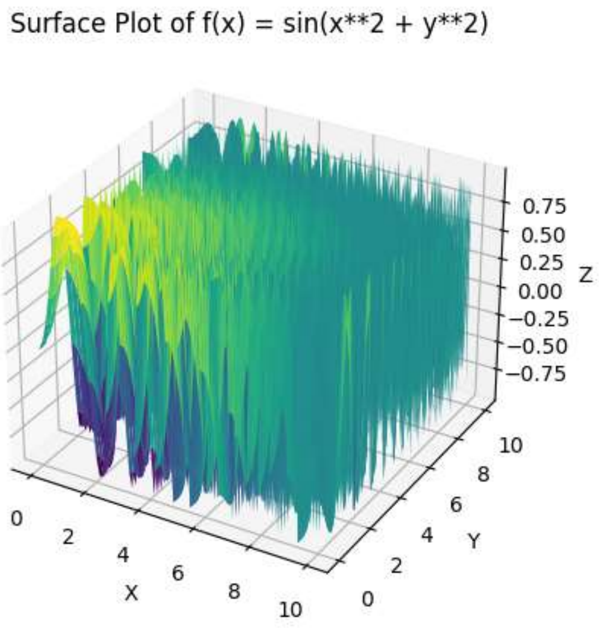


Q.2) Generate 3D surface Plot for the function f(x) = sin (x**2 + y**2) in the interval [0, 10].

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the function
def func(x, y):
    return np.sin(x**2 + y**2)
# Generate x, y values in the interval from 0 to 10
x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)
X, Y = np.meshgrid(x, y)  # Create a grid of x, y values
Z = func(X, Y)  # Compute z values using the function
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')  # Plot the surface
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Surface Plot of f(x) = sin(x**2 + y**2)')
plt.show()  # Show the plot
```

OUTPUT:



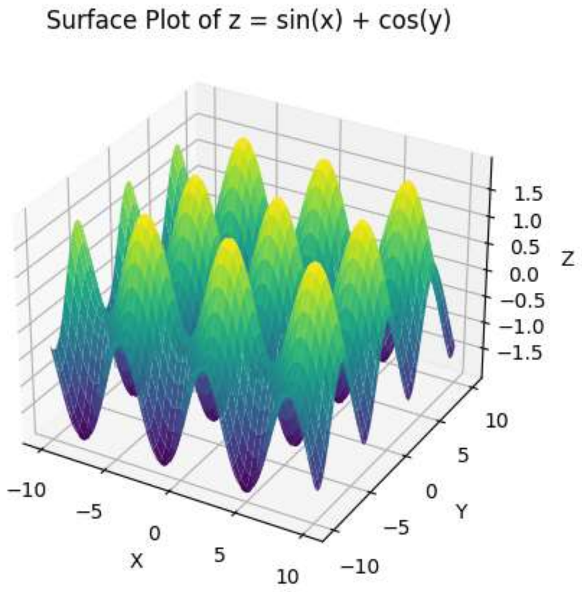
Q.3) Write a Python program to generate 3D plot of the functions z = sin x + cos y in the interval -10 < x,y < 10.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the function
def func(x, y):
    return np.sin(x) + np.cos(y)
# Generate x, y values in the interval from -10 to 10
x = np.linspace(-10, 10, 100)
```

```
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)  # Create a grid of x, y values
Z = func(X, Y)  # Compute z values using the function
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')  # Plot the surface
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Surface Plot of z = sin(x) + cos(y)')
plt.show()  # Show the plot
```

OUTPUT:

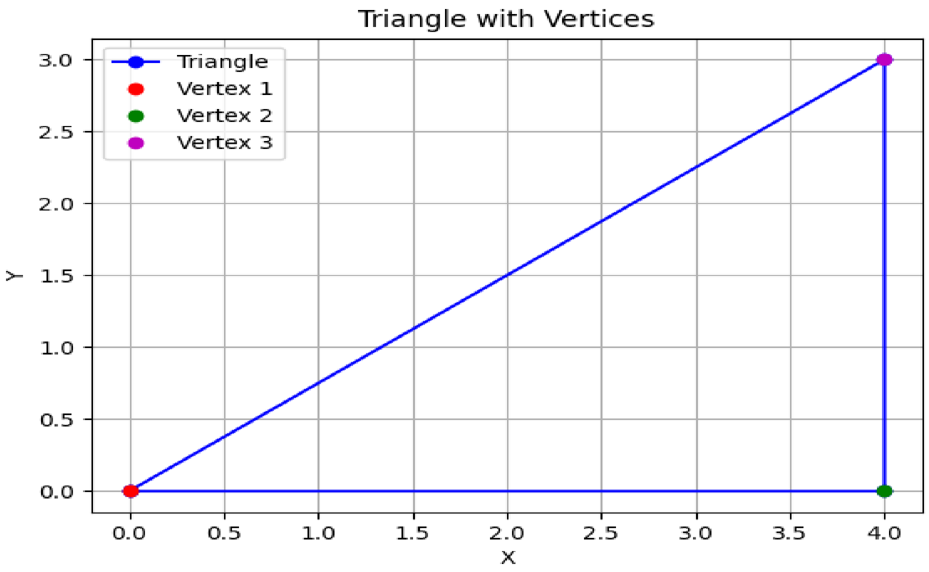


Q.4) Using python, generate triangle with vertices (0, 0), (4, 0), (4, 3) check whether the triangle is Right angle triangle.

Syntax:

```
import matplotlib.pyplot as plt
# Define the vertices of the triangle
vertex1 = (0, 0)
vertex2 = (4, 0)
vertex3 = (4, 3)
# Extract x and y coordinates of the vertices
x = [vertex1[0], vertex2[0], vertex3[0], vertex1[0]]
y = [vertex1[1], vertex2[1], vertex3[1], vertex1[1]]
# Plot the triangle
plt.plot(x, y, 'b-o', label='Triangle')
plt.plot(vertex1[0], vertex1[1], 'ro', label='Vertex 1')
plt.plot(vertex2[0], vertex2[1], 'go', label='Vertex 2')
plt.plot(vertex3[0], vertex3[1], 'mo', label='Vertex 3')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Triangle with Vertices')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



Q.5) Generate vector x in the interval [-7, 7] using numpy package with 50 subintervals.

Syntax:

```
import numpy as np
# Define the interval and number of subintervals
start = -7
end = 7
num_subintervals = 50
# Generate the vector x
x = np.linspace(start, end, num=num_subintervals+1)
# Print the vector x
print("Vector x:")
print(x)
```

OUTPUT:

```
Vector x:
[-7.0000000e+00 -6.7200000e+00 -6.4400000e+00 -6.1600000e+00
 -5.8800000e+00 -5.6000000e+00 -5.3200000e+00 -5.0400000e+00
 -4.7600000e+00 -4.4800000e+00 -4.2000000e+00 -3.9200000e+00
 -3.6400000e+00 -3.3600000e+00 -3.0800000e+00 -2.8000000e+00
 -2.5200000e+00 -2.2400000e+00 -1.9600000e+00 -1.6800000e+00
 -1.4000000e+00 -1.1200000e+00 -8.4000000e-01 -5.6000000e-01
 -2.8000000e-01  8.8817842e-16  2.8000000e-01  5.6000000e-01
  8.4000000e-01  1.1200000e+00  1.4000000e+00  1.6800000e+00
  1.9600000e+00  2.2400000e+00  2.5200000e+00  2.8000000e+00
  3.0800000e+00  3.3600000e+00  3.6400000e+00  3.9200000e+00
  4.2000000e+00  4.4800000e+00  4.7600000e+00  5.0400000e+00
  5.3200000e+00  5.6000000e+00  5.8800000e+00  6.1600000e+00
  6.4400000e+00  6.7200000e+00  7.0000000e+00]
```

Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[6, 0], C[4,4].

Synatx:

```
import numpy as np
# Define the vertices of the triangle
A = np.array([0, 0])
B = np.array([6, 0])
C = np.array([4, 4])
# 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:

```
Side AB: 6.0
Side BC: 4.47213595499958
Side CA: 5.656854249492381
Area: 11.999999999999998
Perimeter: 16.12899020449196
```

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

Max Z = 5x + 3y

Subjected to

x + y <= 7

2x + 5y <= 1

x > 0

y > 0

Syntax:

```
from scipy.optimize import linprog
# Objective function coefficients
c = [-5, -3]
# Coefficient matrix of inequality constraints
A = [[1, 1],
     [2, 5]]
# Right-hand side of inequality constraints
b = [7, 1]
# Bounds on variables
x_bounds = (0, None)
y_bounds = (0, None)
# Solve the linear programming problem
res = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds])
# Check if the optimization was successful
if res.success:
    print("Optimal solution found:")
    print("x =", res.x[0])
    print("y =", res.x[1])
    print("Maximum value of Z =", -res.fun)
else:
    print("Optimization failed. Message:", res.message)
```

```
OUTPUT:
Optimal solution found:
x = 0.5
y = 0.0
Maximum value of Z = 2.5
```

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 >= 10
-8x-3y+3z+2w <= 20
x + y <= 11
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 >= 10
constraint2 = -8 * x - 3 * y + 3 * z + 2 * w <= 20
constraint3 = x + y <= 11
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 = 1.6666667
z = 0.0
w = 0.0
Minimum value of Z = 1.6666667
```

Q.9) Apply Python. Program in each of the following transformation on the point P[3,8]

(I)Refection through y-axis.

(II)Scaling in X-co-ordinate by factor 6.

(III) Rotation about origin through an angle 30^0

(IV) Reflection through the line y =- x

Syntax:

```
import numpy as np
# Point P
P = np.array([3, 8])
```

```
# Transformation I: Reflection through y-axis
P_reflection_y_axis = np.array([-P[0], P[1]])
# Transformation II: Scaling in X-coordinate by factor 6
P_scaling_x = np.array([6 * P[0], P[1]])
# Transformation III: Rotation about origin through an angle of 30 degrees
theta = np.deg2rad(30)  # Convert angle from degrees to radians
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                            [np.sin(theta), np.cos(theta)]])
P_rotation = np.dot(rotation_matrix, P)
# Transformation IV: Reflection through the line y = -x
reflection_line = np.array([[0, -1],
                            [-1, 0]])
P_reflection_line = np.dot(reflection_line, P)
# Print the results
print("Original Point P:", P)
print("Transformation I - Reflection through y-axis:", P_reflection_y_axis)
print("Transformation II - Scaling in X-coordinate by factor 6:", P_scaling_x)
print("Transformation III - Rotation about origin through an angle of 30 degrees:", P_rotation)
print("Transformation IV - Reflection through the line y = -x:", P_reflection_line)
```

OUTPUT:

```
Original Point P: [3 8]
Transformation I - Reflection through y-axis: [-3  8]
Transformation II - Scaling in X-coordinate by factor 6: [18  8]
Transformation III - Rotation about origin through an angle of 30 degrees: [-1.40192379  8.42820323]
Transformation IV - Reflection through the line y = -x: [-8 -3]
```

Q.10) Write a python program to Plot 2D X-axis and Y-axis in black color. In the same diagram plot:-

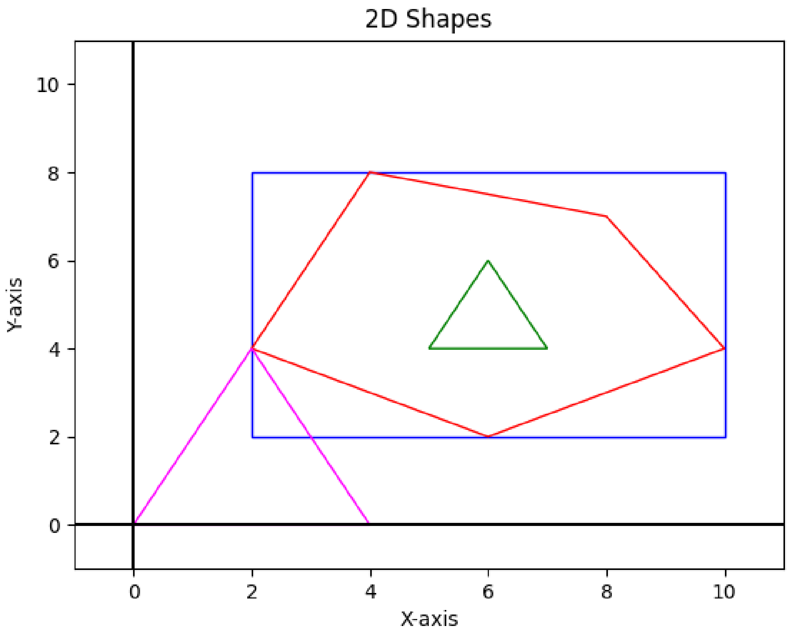
(I) Green Triangle with vertices [5,4],[7,4],[6,6]
(II) Blue rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8].
(III) Red polygon with vertices [6, 2], [10, 4], [8, 7], [4, 8], [2, 4].
(IV) Isosceles triangle with vertices [0, 0], [4, 0], [2, 4].

Syntax:

```
import matplotlib.pyplot as plt
# Create figure and axis
fig, ax = plt.subplots()
# Plot X-axis and Y-axis in black color
```

```
ax.axhline(0, color='black')
ax.axvline(0, color='black')
# Green Triangle with vertices [5,4],[7,4],[6,6]
green_triangle = plt.Polygon([[5, 4], [7, 4], [6, 6]], edgecolor='green', facecolor='none')
ax.add_patch(green_triangle)
# Blue Rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8]
blue_rectangle = plt.Polygon([[2, 2], [10, 2], [10, 8], [2, 8]], edgecolor='blue', facecolor='none')
ax.add_patch(blue_rectangle)
# Red Polygon with vertices [6, 2], [10, 4], [8, 7], [4, 8], [2, 4]
red_polygon = plt.Polygon([[6, 2], [10, 4], [8, 7], [4, 8], [2, 4]], edgecolor='red', facecolor='none')
ax.add_patch(red_polygon)
# Isosceles Triangle with vertices [0, 0], [4, 0], [2, 4]
isosceles_triangle = plt.Polygon([[0, 0], [4, 0], [2, 4]], edgecolor='magenta', facecolor='none')
ax.add_patch(isosceles_triangle)
# Set axis limits
ax.set_xlim([-1, 11])
ax.set_ylim([-1, 11])
# Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('2D Shapes')
# Show the plot
plt.show()
```

OUTPUT:


2D Shapes
Y-axis
X-axis