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

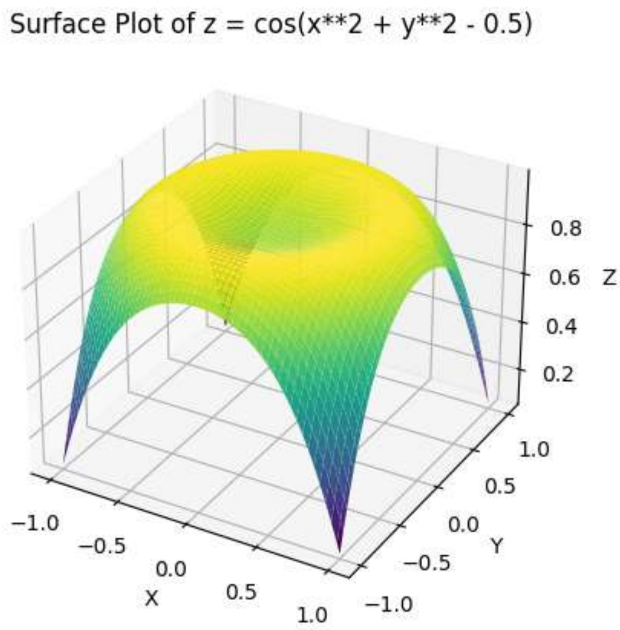
Title of the:- Practical 25 **Expt. No .** 25 **Class :-** S.Y.BCS

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) Write n Python program to generate 3D plot of the function z = six (x) + cos(y) in -10 < x, y < 10.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate x, y values
x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y)
# Calculate z values
Z = np.sin(X) + np.cos(Y)
# Create 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the surface
ax.plot_surface(X, Y, Z, cmap='viridis')
# Set labels and title
```

ax.set_xlabel('X')

ax.set_ylabel('Y')

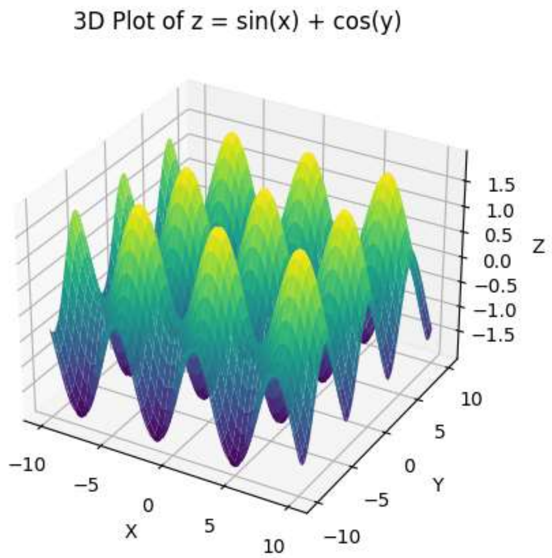
ax.set_zlabel('Z')

ax.set_title('3D Plot of z = sin(x) + cos(y)')

Show the plot

plt.show()

OUTPUT:



Q.3) Using Python plot the graph of function f(x) = sin-1(x) on the interval [-1, l].

Syntax:

import numpy as np

import matplotlib.pyplot as plt

Generate x values

x = np.linspace(-1, 1, 1000)

Calculate f(x) values

f_x = np.arcsin(x)

Create the plot

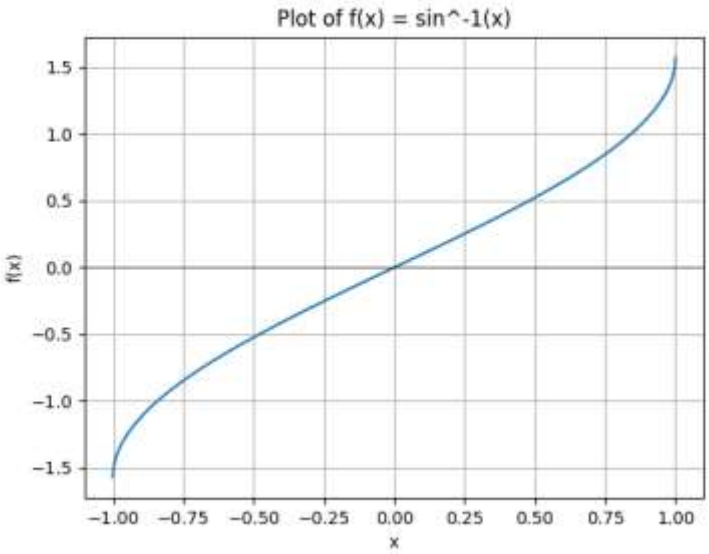
plt.plot(x, f_x)

plt.xlabel('x')

plt.ylabel('f(x)')

```
plt.title('Plot of f(x) = sin^-1(x)')
plt.grid(True)
plt.axhline(0, color='black', lw=0.5) # Add horizontal grid line at y=0
# Show the plot
plt.show()
```

OUTPUT:



Q.4) Rotate the line segment by 180° having endpoints (1,0) and (2,-1).

Syntax:

```
import numpy as np
# Define the endpoints of the line segment
point1 = np.array([1, 0])
point2 = np.array([2, -1])
# Define the rotation matrix for 180 degrees
R = np.array([[-1, 0],[0, -1]])
# Rotate the endpoints of the line segment
rotated_point1 = np.dot(R, point1)
rotated_point2 = np.dot(R, point2)
# Print the rotated coordinates
print("Rotated endpoint 1: ", rotated_point1)
print("Rotated endpoint 2: ", rotated_point2)
```

Output:

```
Rotated endpoint 1:  [-1  0]
Rotated endpoint 2:  [-2  1]
```

Q.5) Using sympy, declare the points P(5, 2), Q(5, -2), R(5, 0), check whether these points are collinear. Declare the ray passing through the points P and Q, find the length of this ray between P and Q. Also find slope of this ray.

Syntax:

```
from sympy import Point, Line
# Declare the points
P = Point(5, 2)
Q = Point(5, -2)
R = Point(5, 0)
# Check if points are collinear
collinear = Point.is_collinear(P, Q, R)
if collinear:
    print("Points P, Q, and R are collinear.")
else:
    print("Points P, Q, and R are not collinear.")
# Declare the ray passing through points P and Q
ray_PQ = Line(P, Q)
# Calculate the length of the ray PQ
length_PQ = P.distance(Q)
print("Length of ray PQ:", length_PQ)
# Calculate the slope of the ray PQ
if ray_PQ.slope == None:
    print("Slope of ray PQ: Undefined (division by zero)")
else:
    print("Slope of ray PQ:", ray_PQ.slope)
```

OUTPUT:

Points P, Q, and R are collinear.

Length of ray PQ: 4

Slope of ray PQ: 00

Q.6) Generate triangle with vertices (0, 0), (4, 0), (1, 4), check whether the triangle is Scalene triangle.

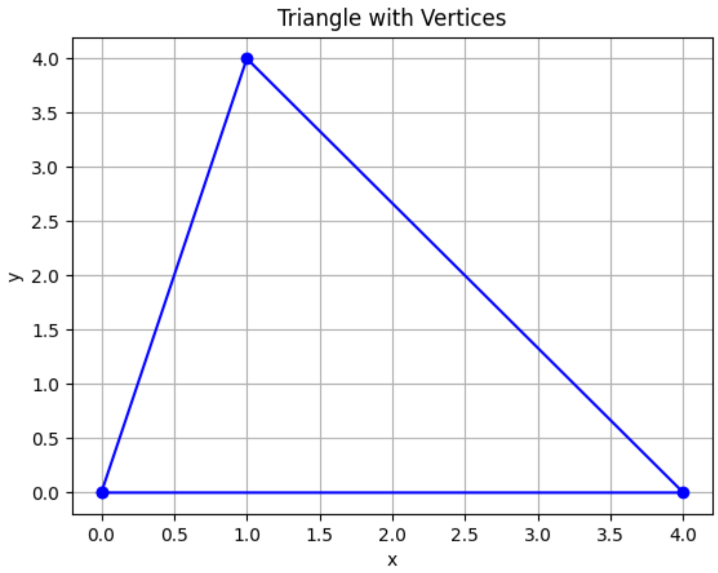
Synatx:

```
import matplotlib.pyplot as plt
# Define the vertices of the triangle
vertices = [(0, 0), (4, 0), (1, 4)]
# Check if the triangle is a scalene triangle
def is_scalene(vertices):
    x1, y1 = vertices[0]
    x2, y2 = vertices[1]
    x3, y3 = vertices[2]
    return (x1 != x2 and x1 != x3 and x2 != x3) and (y1 != y2 and y1 != y3 and y2 != y3)
if is_scalene(vertices):
    print("The triangle is a scalene triangle.")
else:
    print("The triangle is not a scalene triangle.")
# Plot the triangle
x = [point[0] for point in vertices] + [vertices[0][0]]
y = [point[1] for point in vertices] + [vertices[0][1]]
plt.plot(x, y, marker='o', linestyle='-', color='blue')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Triangle with Vertices')
plt.grid(True)
plt.show()
```

OUTPUT:

The triangle is not a scalene triangle.

Plot



Q.7) Write a python program to display the following LPP:

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:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("LPP", LpMinimize)
# Define the decision 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
prob += 4*x + y + 3*z + 5*w, "Z"
```

```
# Define the constraints
prob += 4*x + 6*y - 5*z - 4*w >= 20
prob += -8*x - 3*y + 3*z + 2*w <= 20
prob += -3*x - 2*y + 4*z + w <= 10
# Solve the problem
prob.solve()
# Print the status of the problem
print("Status:", LpStatus[prob.status])
# Print the optimal values of the decision variables
print("Optimal values:")
print("x =", value(x))
print("y =", value(y))
print("z =", value(z))
print("w =", value(w))
# Print the optimal value of the objective function
print("Optimal Z =", value(prob.objective))
```

OUTPUT:

```
Status: Optimal
Optimal values:
x = 0.0
y = 3.3333333
z = 0.0
w = 0.0
Optimal Z = 3.3333333
```

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 = 150x + 75y
subject to
4x+ 6y <= 24
5x + 3y <= 15
x >= 0,y>= 0

Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("LPP", LpMinimize)
# Define the decision variables
x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0)
# Define the objective function
prob += 150*x + 75*y, "Z"
# Define the constraints
prob += 4*x + 6*y <= 24
prob += 5*x + 3*y <= 15
# Solve the problem using the simplex method
prob.solve(PULP_CBC_CMD(msg=False, mip=0))
# Print the status of the problem
print("Status:", LpStatus[prob.status])
# Print the optimal values of the decision variables
print("Optimal values:")
print("x =", value(x))
print("y =", value(y))
# Print the optimal value of the objective function
print("Optimal Z =", value(prob.objective))
```

OUTPUT :

```
Status: Optimal
Optimal values:
x = 0.0
y = 0.0
Optimal Z = 0.0
```

Q.9) Write a python program to apply the following transformation on the point (-2,4) :
(I)Reflection through X-axis.
(II)Scaling in X-coordinate by factor 6.
(III) Shearing in X direction by 4 units.
(IV) Rotate about origin through an angle 30

Syntax:
```
import numpy as np
# Initial point
point = np.array([-2, 4])
# Transformation I: Reflection through X-axis
reflection_x = np.array([[1, 0],[0, -1]])
reflected_point = np.dot(reflection_x, point)
print("Reflection through X-axis:")
print("Initial point:", point)
print("Reflected point:", reflected_point)
# Transformation II: Scaling in X-coordinate by factor 6
scaling_x = np.array([[6, 0],[0, 1]])
scaled_point = np.dot(scaling_x, point)
print("\nScaling in X-coordinate by factor 6:")
print("Initial point:", point)
print("Scaled point:", scaled_point)
# Transformation III: Shearing in X direction by 4 units
shearing_x = np.array([[1, 4],[0, 1]])
sheared_point = np.dot(shearing_x, point)
print("\nShearing in X direction by 4 units:")
print("Initial point:", point)
print("Sheared point:", sheared_point)
# Transformation IV: Rotate about origin through an angle of 30 degrees
angle = np.deg2rad(30)
rotation = np.array([[np.cos(angle), -np.sin(angle)],[np.sin(angle), np.cos(angle)]])
rotated_point = np.dot(rotation, point)
print("\nRotate about origin through an angle of 30 degrees:")
print("Initial point:", point)
print("Rotated point:", rotated_point)
```

OUTPUT:
Reflection through X-axis:
Initial point: [-2 4]
Reflected point: [-2 -4]
Scaling in X-coordinate by factor 6:
Initial point: [-2 4]
Scaled point: [-12   4]
Shearing in X direction by 4 units:
Initial point: [-2 4]
Sheared point: [14 4]
Rotate about origin through an angle of 30 degrees:
Initial point: [-2 4]
Rotated point: [-3.73205081 2.46410162]

Q.10) Write a python program to find the combined transformation of the line segment between the points A[3,2] & B [2,-3] for the following sequence of transformation:

(I) Rotation about origin through an angle pi/6
(II) Scaling in Y – Coordinate by -4 unit.
(III) Uniform scaling by -6.4 units
(IV) Shearing in Y direction by 5 units.

Syntax:

```
import numpy as np
# Initial points
A = np.array([3, 2])
B = np.array([2, -3])
# Transformation I: Rotation about origin through an angle pi/6
angle = np.pi / 6
rotation = np.array([[np.cos(angle), -np.sin(angle)],[np.sin(angle), np.cos(angle)]])
rotated_A = np.dot(rotation, A)
rotated_B = np.dot(rotation, B)
print("Transformation I: Rotation about origin through an angle pi/6")
print("Rotated point A:", rotated_A)
print("Rotated point B:", rotated_B)
# Transformation II: Scaling in Y-coordinate by -4 units
scaling_y = np.array([[1, 0],[0, -4]])
scaled_A = np.dot(scaling_y, rotated_A)
scaled_B = np.dot(scaling_y, rotated_B)
print("\nTransformation II: Scaling in Y-coordinate by -4 units")
```

```
print("Scaled point A:", scaled_A)
print("Scaled point B:", scaled_B)
# Transformation III: Uniform scaling by -6.4 units
uniform_scaling = np.array([[-6.4, 0],[0, -6.4]])
uniform_scaled_A = np.dot(uniform_scaling, scaled_A)
uniform_scaled_B = np.dot(uniform_scaling, scaled_B)
print("\nTransformation III: Uniform scaling by -6.4 units")
print("Uniform scaled point A:", uniform_scaled_A)
print("Uniform scaled point B:", uniform_scaled_B)
# Transformation IV: Shearing in Y direction by 5 units
shearing_y = np.array([[1, 5],[0, 1]])
sheared_A = np.dot(shearing_y, uniform_scaled_A)
sheared_B = np.dot(shearing_y, uniform_scaled_B)
print("\nTransformation IV: Shearing in Y direction by 5 units")
print("Sheared point A:", sheared_A)
print("Sheared point B:", sheared_B)
```

OUTPUT:

```
Transformation I: Rotation about origin through an angle pi/6
Rotated point A: [1.59807621 3.23205081]
Rotated point B: [ 3.23205081 -1.59807621]
Transformation II: Scaling in Y-coordinate by -4 units
Scaled point A: [  1.59807621 -12.92820323]
Scaled point B: [3.23205081 6.39230485]
Transformation III: Uniform scaling by -6.4 units
Uniform scaled point A: [-10.22768775  82.74050067]
Uniform scaled point B: [-20.68512517 -40.91075101]
Transformation IV: Shearing in Y direction by 5 units
Sheared point A: [403.47481562  82.74050067]
Sheared point B: [-225.23888022  -40.91075101]
```