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 8 **Expt. No .** 8 **Class :-** S.Y.BCS

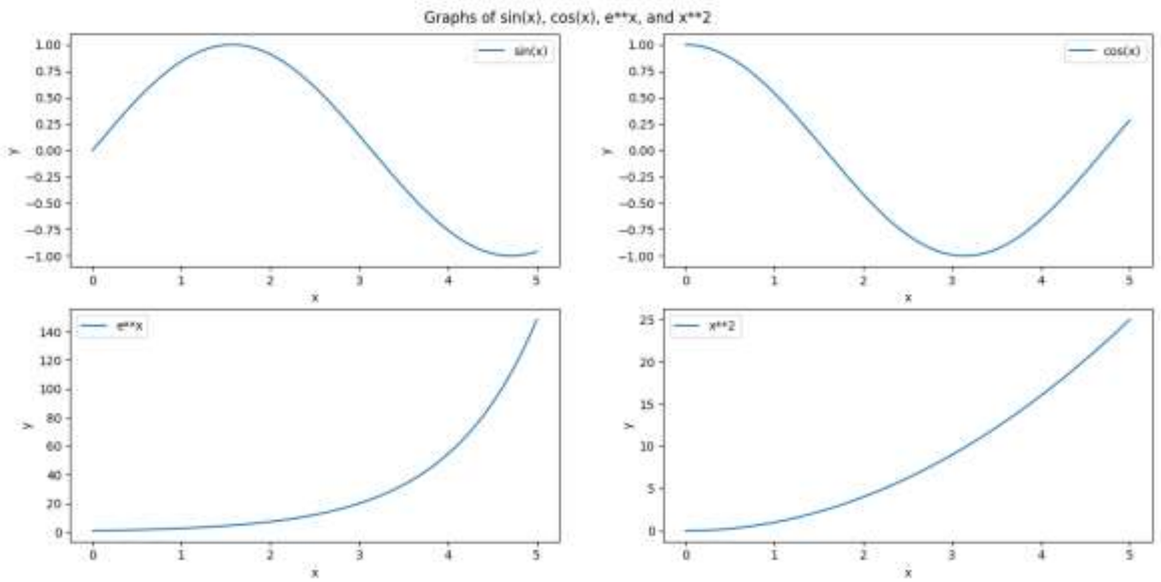
Q.1) Plot the graphs of sin x, cos x, e**x and x**2 in [0, 5] in one figure with (2 x 2) subplot

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate x values
x = np.linspace(0, 5, 500)
# Compute y values for sin(x), cos(x), e**x, x**2
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.exp(x)
y4 = x**2
# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
fig.suptitle('Graphs of sin(x), cos(x), e**x, and x**2')
# Plot sin(x)
axs[0, 0].plot(x, y1, label='sin(x)')
axs[0, 0].legend()
# Plot cos(x)
axs[0, 1].plot(x, y2, label='cos(x)')
axs[0, 1].legend()
# Plot e**x
axs[1, 0].plot(x, y3, label='e**x')
axs[1, 0].legend()
```

```
# Plot x**2
axs[1, 1].plot(x, y4, label='x**2')
axs[1, 1].legend()
# Set x and y axis labels for all subplots
for ax in axs.flat:
    ax.set_xlabel('x')
    ax.set_ylabel('y')
# Adjust spacing between subplots
fig.tight_layout()
# Show the plot
plt.show()
```

OUTPUT:



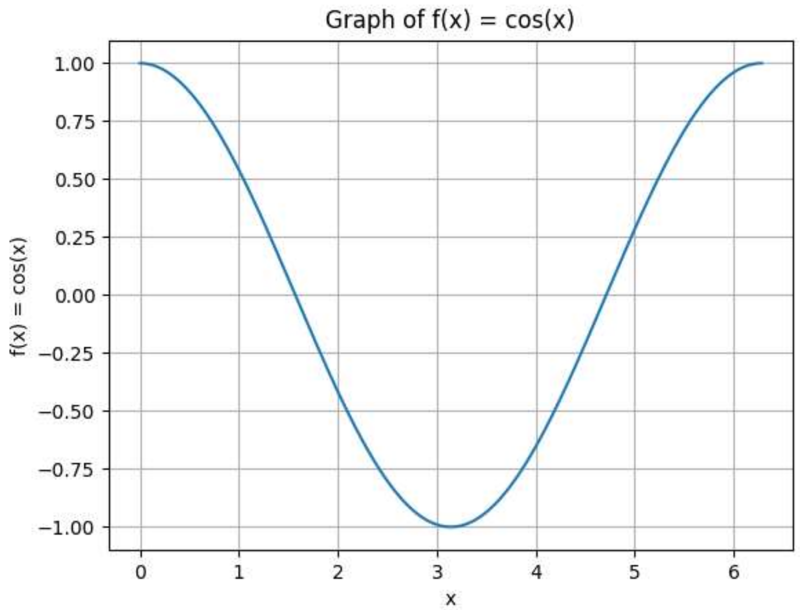
Q.2) Using Python plot the graph of function f(x) = cos(x) in the interval [0, 2*pi].

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate x values
x = np.linspace(0, 2*np.pi, 500)
# Compute y values for cos(x)
```

```
y = np.cos(x)
# Plot f(x) = cos(x)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x) = cos(x)')
plt.title('Graph of f(x) = cos(x)')
plt.grid(True)
plt.show()
```

OUTPUT:



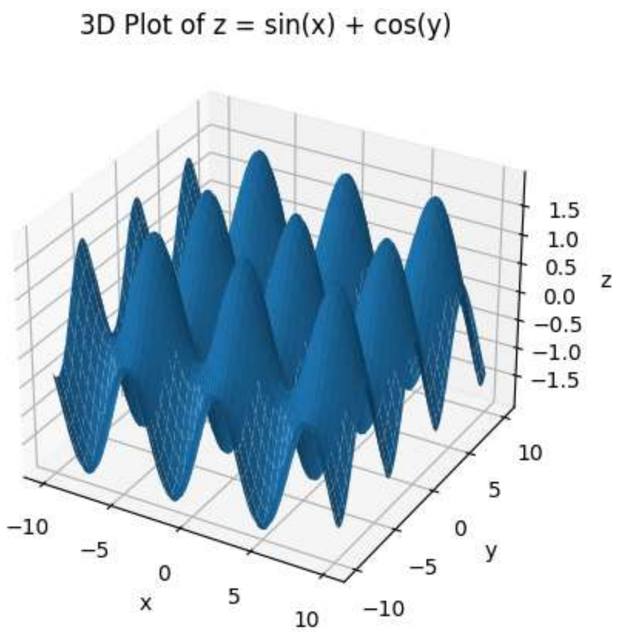
Q.3) Write a Python program to generate 3D plot of the functions z = sinx + cosy in -10 < x, y < 10.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate x and y values
x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
```

```
# Create a meshgrid of x and y values
X, Y = np.meshgrid(x, y)
# Compute z values for the function z = sin(x) + cos(y)
Z = np.sin(X) + np.cos(Y)
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D Plot of z = sin(x) + cos(y)')
plt.show()
```

OUTPUT:



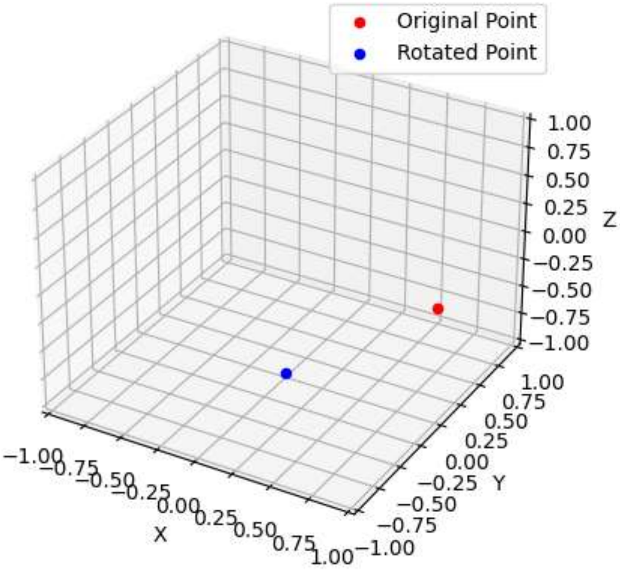
Q.4) Write a Python program in 3D to rotate the point (1, 0, 0) through XZ Plane in anticlockwise direction (Rotation through Y axis) by an angle of 90°.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Point to rotate
point = np.array([1, 0, 0])
# Rotation angle in radians
angle = np.deg2rad(90)
# Rotation matrix for Y axis (anticlockwise)
rotation_matrix = np.array([
    [np.cos(angle), 0, np.sin(angle)],
    [0, 1, 0],
    [-np.sin(angle), 0, np.cos(angle)]
])
# Apply rotation
rotated_point = np.dot(rotation_matrix, point)
# Create 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot original point
ax.scatter(point[0], point[1], point[2], c='r', marker='o', label='Original Point')
# Plot rotated point
ax.scatter(rotated_point[0], rotated_point[1], rotated_point[2], c='b', marker='o',
label='Rotated Point')
# Set plot limits
ax.set_xlim([-1, 1])
ax.set_ylim([-1, 1])
ax.set_zlim([-1, 1])
# Set plot labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# Add legnd
ax.legend()
```

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

Output:

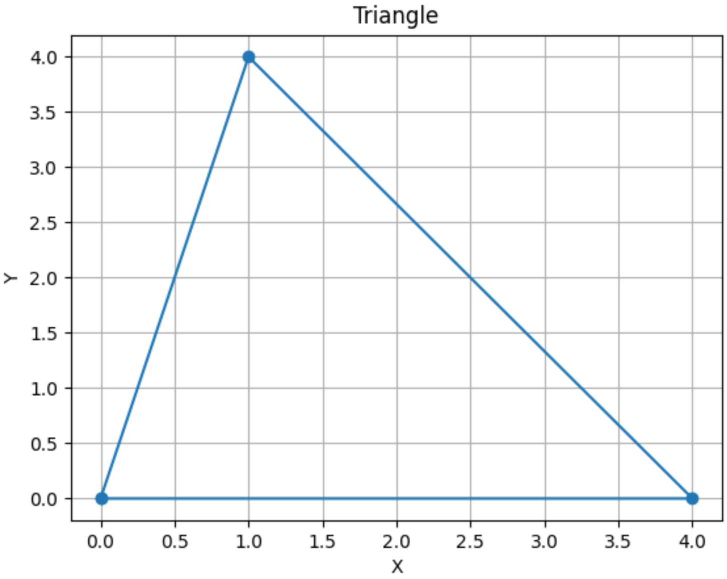


Q.5) Using python, generate triangle with vertices (0, 0), (4, 0), (1, 4), check whether the triangle is Scalene triangle.

Syntax:

```
import matplotlib.pyplot as plt
# Vertices of the triangle
vertex1 = (0, 0)
vertex2 = (4, 0)
vertex3 = (1, 4)
# Plot the triangle
plt.plot(*zip(vertex1, vertex2, vertex3, vertex1), marker='o')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Triangle')
plt.grid(True)
plt.show()
```

OUTPUT:



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:

```
Triangle ABC:
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 = 150x + 75y

Subjected to

4x + 6y <= 24

5x + 3y <= 15

x > 0 , y > 0

Syntax:

```
from pulp import *
# Create the LP problem as a maximization problem
problem = LpProblem("LPP", LpMaximize)
# Define the decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
```

```
y = LpVariable('y', lowBound=0, cat='Continuous')
# Define the objective function
problem += 150 * x + 75 * y, "Z"
# Define the constraints
problem += 4 * x + 6 * y <= 24, "Constraint1"
problem += 5 * x + 3 * y <= 15, "Constraint2"
# Solve the LP problem
problem.solve()
# Print the status of the solution
print("Status:", LpStatus[problem.status])
# Print the optimal values of x and y
print("Optimal x =", value(x))
print("Optimal y =", value(y))
# Print the optimal value of the objective function
print("Optimal Z =", value(problem.objective
```

OUTPUT:

```
Status: Optimal
Optimal x = 3.0
Optimal y = 0.0
Optimal Z = 450.0
```

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 = 3x + 5y + 4z
subject to
2x + 3y <= 8
2y + 5z <= 10
3x + 2y + 5z <= 15
x=>0, y=>0

Syntax:

```
from pulp import *
```

```
# Create a PuLP minimization problem
prob = LpProblem("LPP", LpMinimize)
# Define decision variables
x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0)
z = LpVariable("z", lowBound=0)
# Define objective function
prob += 3*x + 5*y + 4*z, "Z"
# Define constraints
prob += 2*x + 3*y <= 8, "Constraint1"
prob += 2*y + 5*z <= 10, "Constraint2"
prob += 3*x + 2*y + 5*z <= 15, "Constraint3"
# Solve the problem
prob.solve()
# Print the status of the solution
print("Status:", LpStatus[prob.status])
# If the problem is solved, print the optimal solution and the optimal value of Z
if prob.status == LpStatusOptimal:
    print("Optimal Solution:")
    print("x =", value(x))
    print("y =", value(y))
    print("z =", value(z))
    print("Z =", value(prob.objective))
```

OUTPUT:
Status: Optimal
Optimal Solution:
x = 0.0
y = 0.0
z = 0.0
Z = 0.0

Q.9) Apply Python. Program in each of the following transformation on the point P[4,-2]
(I)Refection through Y-axis.
(II)Scaling in X-co-ordinate by factor 5.
(III) Rotation about origin through an angle pi/2..
(IV)Shearing in X direction by 7/2 units
Syntax:

```
import numpy as np
# Initial point P
```

```
P = np.array([4, -2])
# (I) Reflection through Y-axis
P_reflect_y = np.array([-P[0], P[1]])
# (II) Scaling in X-coordinate by factor 5
P_scale_x = np.array([5 * P[0], P[1]])
# (III) Rotation about origin through an angle pi/2
angle = np.pi / 2
P_rotate = np.array([P[0] * np.cos(angle) - P[1] * np.sin(angle), P[0] * np.sin(angle) + P[1] * np.cos(angle)])
# (IV) Shearing in X-direction by 7/2 units
shearing_factor = 7 / 2
P_shear_x = np.array([P[0] + shearing_factor * P[1], P[1]])
# Print the transformed points
print("Initial point P:", P)
print("Reflection through Y-axis:", P_reflect_y)
print("Scaling in X-coordinate by factor 5:", P_scale_x)
print("Rotation about origin through an angle pi/2:", P_rotate)
print("Shearing in X-direction by 7/2 units:", P_shear_x)
```

OUTPUT:

```
Initial point P: [ 4 -2]
Reflection through Y-axis: [-4 -2]
Scaling in X-coordinate by factor 5: [20 -2]
Rotation about origin through an angle pi/2: [2. 4.]
Shearing in X-direction by 7/2 units: [-3. -2.]
```

Q.10) Find the combined transformation of the line segment between the point A[7, -2] & B[6, 2] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi/3.
(II) Scaling in X-Coordinate by 7 units.
(III) Uniform scaling by -4 units
(IV) Reflection through the line X - axis

Syntax:

```
import numpy as np
# Define the point P
P = np.array([4, -2])
# Define the transformation functions
def rotate_about_origin(point, angle):
    # Rotation about origin through an angle
    cos_theta = np.cos(angle)
    sin_theta = np.sin(angle)
```

```
    x = point[0]
    y = point[1]
    x_rotated = x * cos_theta - y * sin_theta
    y_rotated = x * sin_theta + y * cos_theta
    return np.array([x_rotated, y_rotated])
def scale_x(point, factor):
    # Scaling in X-coordinate
    x_scaled = point[0] * factor
    y_scaled = point[1]
    return np.array([x_scaled, y_scaled])
def uniform_scale(point, factor):
    # Uniform scaling
    x_scaled = point[0] * factor
    y_scaled = point[1] * factor
    return np.array([x_scaled, y_scaled])
def reflect_x_axis(point):
    # Reflection through X-axis
    x_reflected = point[0]
    y_reflected = -point[1]
    return np.array([x_reflected, y_reflected])
# Apply the transformations to the point P
angle = np.pi / 3
P_rotated = rotate_about_origin(P, angle)
P_scaled_x = scale_x(P_rotated, 7)
P_uniform_scaled = uniform_scale(P_scaled_x, -4)
P_reflected_x_axis = reflect_x_axis(P_uniform_scaled)
# Print the transformed points
print("Point P: ", P)
print("After Rotation: ", P_rotated)
print("After Scaling in X-Coordinate: ", P_scaled_x)
print("After Uniform Scaling: ", P_uniform_scaled)
print("After Reflection through X-axis: ", P_reflected_x_axis)
```

OUTPUT:

```
Point P:  [ 4 -2]
After Rotation:  [3.73205081 2.46410162]
After Scaling in X-Coordinate:  [26.12435565  2.46410162]
After Uniform Scaling:  [-104.49742261   -9.85640646]
After Reflection through X-axis:  [-104.49742261    9.85640646]
```