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

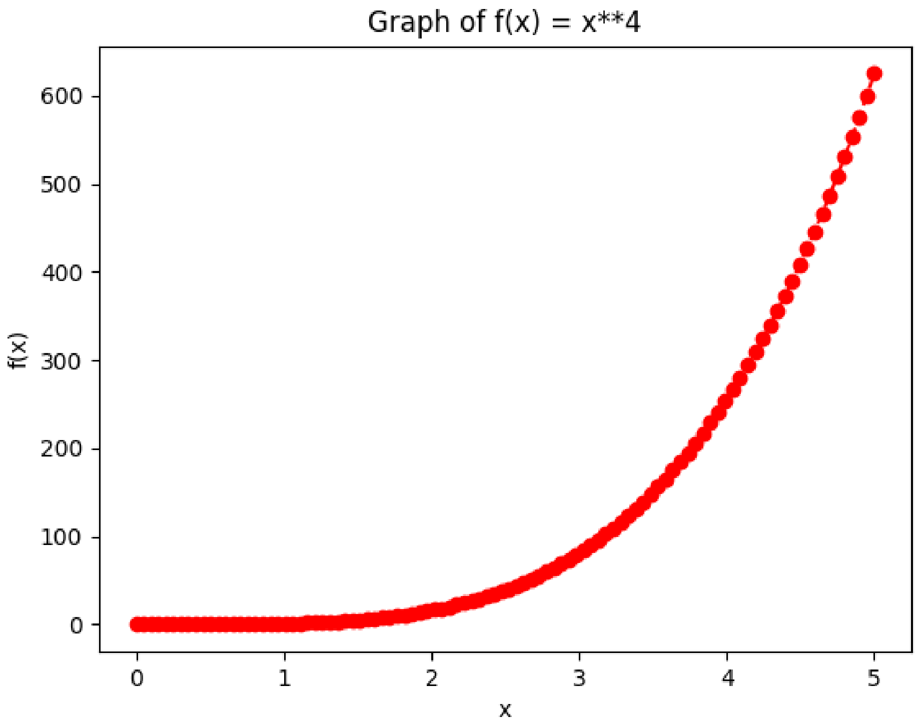
Title of the:- Practical 7 **Expt. No .** 7 **Class :-** S.Y.BCS

Q.1) Plot the graph of f(x) = x**4 in [0, 5] with red dashed line with circle markers.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the function f(x) = x**4
def f(x):
    return x**4
# Generate x values in the interval [0, 5]
x = np.linspace(0, 5, 100)
# Generate y values using the function f(x)
y = f(x)
# Plot the graph with red dashed line and circle markers
plt.plot(x, y, 'r--o', markersize=6)
# Set x-axis label
plt.xlabel('x')
# Set y-axis label
plt.ylabel('f(x)')
# Set title
plt.title('Graph of f(x) = x**4')
# Show the plot
plt.show()
```

OUTPUT:



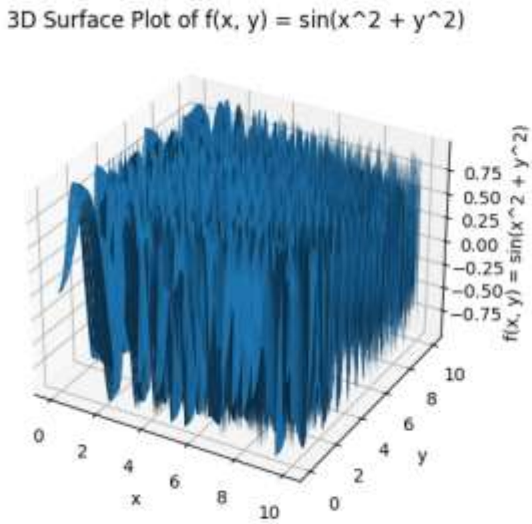
Q.2) Using python, generate 3D surface Plot for the function $f(x) = \sin(x^2 + y^2)$ in the interval [0,10]

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate x and y values in the interval [0,10]
x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)
# Create a grid of x and y values
X, Y = np.meshgrid(x, y)
# Compute z values using the function f(x, y) = sin(x^2 + y^2)
Z = np.sin(X**2 + Y**2)
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
ax.plot_surface(X, Y, Z)
# Set labels and title
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y) = sin(x^2 + y^2)')
ax.set_title('3D Surface Plot of f(x, y) = sin(x^2 + y^2)')
# Show the plot
plt.show()
```

OUTPUT:



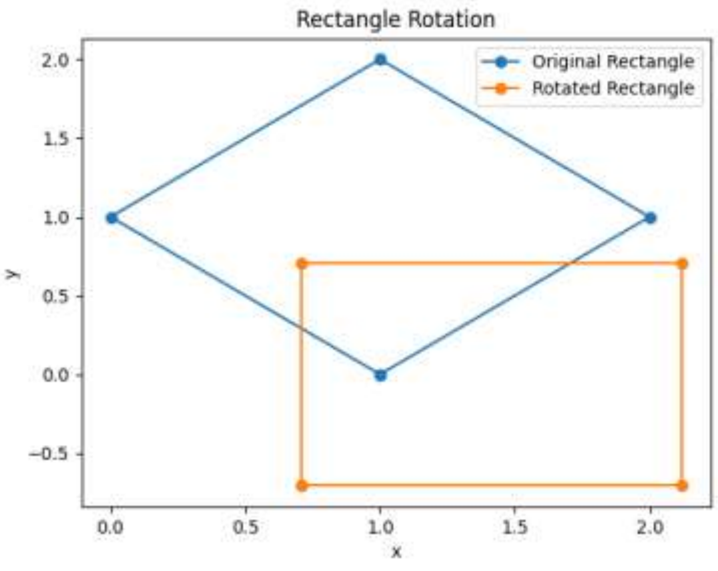
Q.3) Write python program to draw rectangle with vertices [1, 0], [2, 1], [1, 2] and [0, 1], its rotation.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the rectangle vertices
vertices = np.array([[1, 0], [2, 1], [1, 2], [0, 1], [1, 0]])
# Extract x and y coordinates of the vertices
x = vertices[:, 0]
y = vertices[:, 1]
```

```
# Plot the rectangle
plt.plot(x, y, '-o', label='Original Rectangle')
# Calculate the rotation angle in radians
theta = np.radians(45)
# Rotate the rectangle vertices
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                            [np.sin(theta), np.cos(theta)]])
rotated_vertices = np.dot(vertices, rotation_matrix)
# Extract x and y coordinates of the rotated vertices
rotated_x = rotated_vertices[:, 0]
rotated_y = rotated_vertices[:, 1]
# Plot the rotated rectangle
plt.plot(rotated_x, rotated_y, '-o', label='Rotated Rectangle')
# Set x-axis label
plt.xlabel('x')
# Set y-axis label
plt.ylabel('y')
# Set title
plt.title('Rectangle Rotation')
# Add legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:

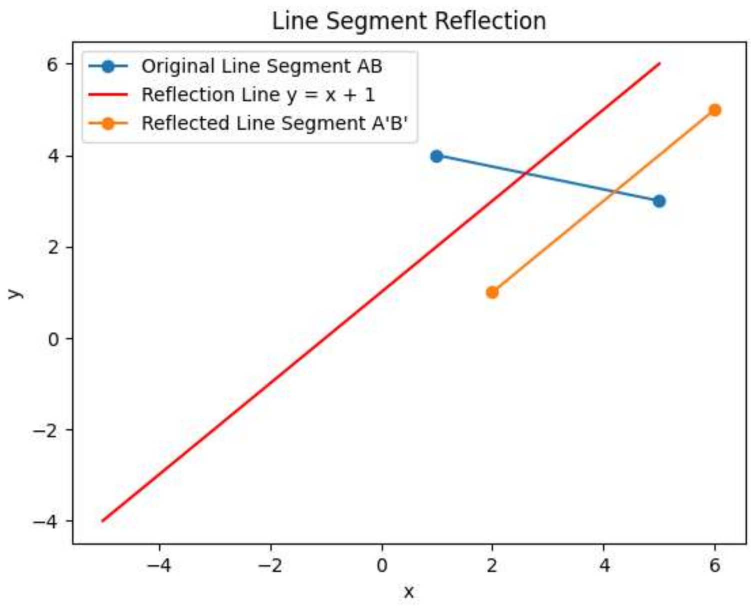


Q.4) Write a Python program to reflect the line segment joining the points A[5, 3] & B[1, 4] through the line y = x + 1.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the points A and B
A = np.array([5, 3])
B = np.array([1, 4])
# Define the equation of the reflection line
reflection_line = lambda x: x + 1
# Plot the original line segment AB
plt.plot([A[0], B[0]], [A[1], B[1]], '-o', label='Original Line Segment AB')
# Plot the reflection line
x_vals = np.linspace(-5, 5, 100) # Generate x values for the plot
plt.plot(x_vals, reflection_line(x_vals), '-r', label='Reflection Line y = x + 1')
# Calculate the reflected points
reflected_A = np.array([reflection_line(A[0]), A[0]])
reflected_B = np.array([reflection_line(B[0]), B[0]])
# Plot the reflected line segment A'B'
plt.plot([reflected_A[0], reflected_B[0]], [reflected_A[1], reflected_B[1]], '-o',
label='Reflected Line Segment A\'B\'')
# Set x-axis label
plt.xlabel('x')
# Set y-axis label
plt.ylabel('y')
# Set title
plt.title('Line Segment Reflection')
# Add legend
plt.legend()
# Show the plot
plt.show()
```

Output:



Q.5) Using sympy declare the points P(5, 2), Q(5, -2), R(5, 0), check whether these points arc 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
# Define the points P, Q, and R
P = Point(5, 2)
Q = Point(5, -2)
R = Point(5, 0)
# Check if points P, Q, and R are collinear
line_PQ = Line(P, Q)
line_PR = Line(P, R)
collinear = line_PQ.is_parallel(line_PR)
# Print the result
if collinear:
    print("Points P, Q, and R are collinear")
else:
    print("Points P, Q, and R are not collinear")
```

```
# Calculate the length of the ray PQ
length_PQ = P.distance(Q)
# Calculate the slope of the ray PQ
slope_PQ = (Q.y - P.y) / (Q.x - P.x)
# Print the length and slope of the ray PQ
print("Length of the ray PQ:", length_PQ)
print("Slope of the ray PQ:", slope_PQ)
```

OUTPUT:

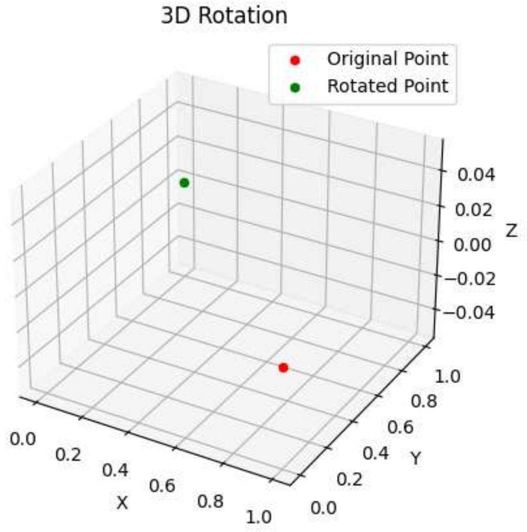
```
 Points P, Q, and R are collinear
Length of the ray PQ: 4
Slope of the ray PQ: zoo
```

Q.6) Write a Python program in 3D to rotate the point (1, 0, 0) through X Plane in anticlockwise direction (Rotation through Z axis) by an angle of 90°.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the point to be rotated
point = np.array([1, 0, 0])
# Define the rotation angle in degrees
angle = np.radians(90)
# Define the rotation matrix for rotating around the Z axis
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle), 0],
                            [np.sin(angle), np.cos(angle), 0],
                            [0, 0, 1]])
# Perform the rotation
rotated_point = np.dot(rotation_matrix, point)
# Create a 3D plot
fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')
# Plot the original point
ax.scatter(point[0], point[1], point[2], c='r', marker='o', label='Original Point')
# Plot the rotated point
ax.scatter(rotated_point[0], rotated_point[1], rotated_point[2], c='g', marker='o', label='Rotated Point')
# Set the axes labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# Set the plot title
ax.set_title('3D Rotation')
# Set the plot legend
ax.legend()
# Show the plot
plt.show()
```

OUTPUT:



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

Max Z = 3.5x +2 y

Subjected to

x + y >= 5

x >= 4

y <= 2

x > 0 , y > 0

Syntax:

```
import numpy as np
from scipy.optimize import linprog
# Coefficients of the objective function
c = [-3.5, -2]
# Coefficients of the inequality constraints
A = [[-1, -1], [-1, 0], [0, 1]]
b = [-5, -4, 2]
# Bounds on the variables
x_bounds = (0, None)
y_bounds = (0, None)
# Solve the linear programming problem
result = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds])
if result.success:
    print("Optimal solution found:")
    print("x =", result.x[0])
    print("y =", result.x[1])
    print("Maximum value of Z =", -result.fun)
else:
    print("Optimal solution not found.")
```

OUTPUT:

Optimal solution not found.

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 = x + 2y + z
subject to
x + 2y + 2x <= 1
3x + 2y + z >= 8
x + y <= 11
x>= 0, y>= 0 , z>=0

Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("LP Problem", LpMinimize)
# Define decision variables
x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0)
z = LpVariable("z", lowBound=0)
# Objective function
prob += x + 2 * y + z, "Z"
# Constraints
prob += x + 2 * y + 2 * x <= 1, "constraint1"
prob += 3 * x + 2 * y + z >= 8, "constraint2"
prob += x + y <= 11, "constraint3"
# Solve the problem using the simplex method
prob.solve(PULP_CBC_CMD())
# Print the results
print("Status:", LpStatus[prob.status])
if prob.status == LpStatusOptimal:
    print("Optimal Solution:")
    print("x =", value(x))
    print("y =", value(y))
    print("z =", value(z))
    print("Optimal Objective Value (Z) =", value(prob.objective))
```

OUTPUT:

```
Status: Optimal
Optimal Solution:
x = 0.33333333
y = 0.0
z = 7.0
```

Optimal Objective Value (Z) = 7.33333333

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 3.
(III) Rotation about origin through an angle pi
(IV) Shearing in both X and Y Direction by -2 and 4 unit Respectively.
Syntax:

```
import numpy as np
# Point P
P = np.array([4, -2])
# (I) Reflection through y-axis
reflection_y_axis = np.array([-1, 1])  # Reflection matrix through y-axis
P_reflected_y_axis = np.dot(reflection_y_axis, P)
print("Reflection through y-axis:", P_reflected_y_axis)
# (II) Scaling in X-coordinate by factor 3
scaling_x = np.array([3, 1])  # Scaling matrix in X-coordinate by factor 3
P_scaled_x = np.dot(scaling_x, P)
print("Scaling in X-coordinate by factor 3:", P_scaled_x)
# (III) Rotation about origin through an angle pi
angle_pi = np.pi  # Angle in radians
rotation_pi = np.array([[np.cos(angle_pi), -np.sin(angle_pi)],
                        [np.sin(angle_pi),  np.cos(angle_pi)]])  # Rotation matrix about origin by angle pi
P_rotated_pi = np.dot(rotation_pi, P)
print("Rotation about origin through angle pi:", P_rotated_pi)
# (IV) Shearing in both X and Y Direction by -2 and 4 units respectively
shear_x = np.array([1, -2])  # Shearing matrix in X-direction by -2 units
shear_y = np.array([4, 1])  # Shearing matrix in Y-direction by 4 units
P_sheared = np.dot(shear_x, P) + np.dot(shear_y, P)
print("Shearing in both X and Y Direction by -2 and 4 units respectively:", P_sheared)
```

OUTPUT:

```
Reflection through y-axis: -6
Scaling in X-coordinate by factor 3: 10
Rotation about origin through angle pi: [-4.  2.]
Shearing in both X and Y Direction by -2 and 4 units respectively: 22
```

Q.10) Find the combined transformation of line segment between the points A[4,-1] & B[3,2] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi/4.

(II) Shearing in Y direction by 7 units.

(III) Scaling in X – direction by 5 units

(IV) Reflection through y – axis

Syntax:

```
import numpy as np
# Points A and B
A = np.array([4, -1])
B = np.array([3, 2])
# (I) Rotation about origin through an angle pi/4
angle_pi_4 = np.pi / 4  # Angle in radians
rotation_pi_4 = np.array([[np.cos(angle_pi_4), -np.sin(angle_pi_4)],
                          [np.sin(angle_pi_4), np.cos(angle_pi_4)]])  # Rotation matrix about origin by
angle pi/4
A_rotated = np.dot(rotation_pi_4, A)
B_rotated = np.dot(rotation_pi_4, B)
# (II) Shearing in Y direction by 7 units
shear_y_7 = np.array([0, 7])  # Shearing matrix in Y-direction by 7 units
A_sheared = A_rotated + np.dot(shear_y_7, A_rotated)
B_sheared = B_rotated + np.dot(shear_y_7, B_rotated)
# (III) Scaling in X direction by 5 units
scaling_x_5 = np.array([5, 1])  # Scaling matrix in X-direction by 5 units
A_scaled_x = np.dot(scaling_x_5, A_sheared)
B_scaled_x = np.dot(scaling_x_5, B_sheared)
# (IV) Reflection through y-axis
reflection_y_axis = np.array([-1, 1])  # Reflection matrix through y-axis
A_reflected_y_axis = np.dot(reflection_y_axis, A_scaled_x)
B_reflected_y_axis = np.dot(reflection_y_axis, B_scaled_x)
print("Line segment after applying the sequence of transformations:")
print("A:", A_reflected_y_axis)
print("B:", B_reflected_y_axis)
```

OUTPUT:

```
Line segment after applying the sequence of transformations:
A: [-108.8944443  108.8944443]
B: [-155.56349186  155.56349186]
```