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

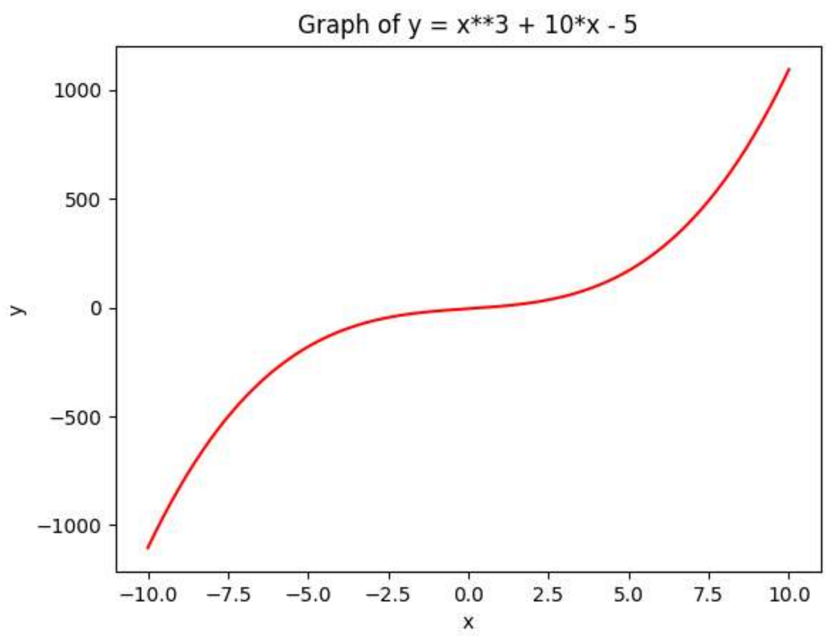
Title of the:- Practical 12 **Expt. No .** 12 **Class :-** S.Y.BCS

Q.1) write a python program to plot the graph of y = x**3 + 10*x - 5, for x belongs [-10, 10] in red color.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the equation y = x**3 + 10*x - 5
def equation(x):
    return x**3 + 10*x - 5
# Generate x values in the range [-10, 10]
x = np.linspace(-10, 10, 500)
# Evaluate the y values using the equation
y = equation(x)
# Create the plot
plt.plot(x, y, color='red')
# Set the plot title and axis labels
plt.title("Graph of y = x**3 + 10*x - 5")
plt.xlabel("x")
plt.ylabel("y")
# Show the plot
plt.show()
```

OUTPUT:



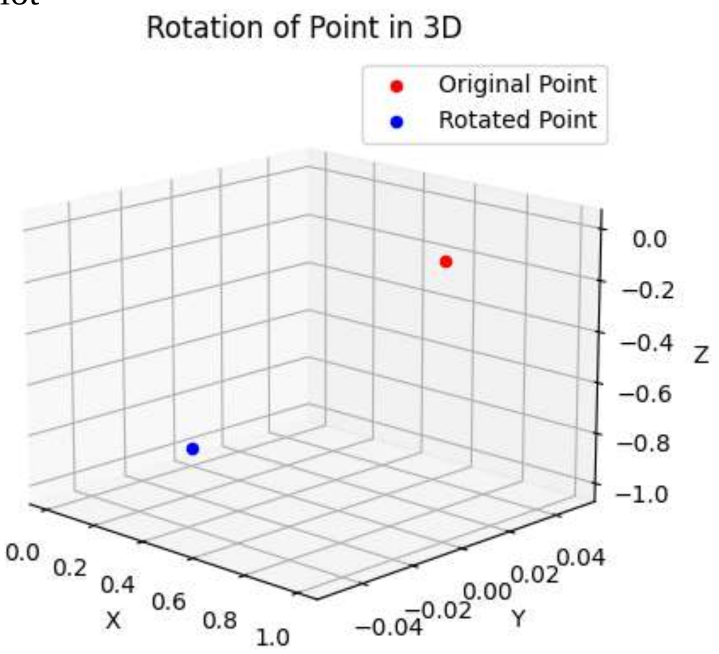
Q.2) write a python program in 3D to rotate the point (1, 0, 0) through XZ- plane in clockwise direction (rotation through Y- axis by an angle of 90°).

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the point to rotate
point = np.array([1, 0, 0])
# Define the rotation angle in radians
theta = np.radians(90)
# Create the 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the original point
ax.scatter(point[0], point[1], point[2], color='red', label='Original Point')
# Perform the rotation
rotated_point = np.dot(np.array([[np.cos(theta), 0, np.sin(theta)],
```

```
                    [0, 1, 0],
                    [-np.sin(theta), 0, np.cos(theta)]]), point)
# Plot the rotated point
ax.scatter(rotated_point[0], rotated_point[1], rotated_point[2], color='blue',
label='Rotated Point')
# Set the plot title and axis labels
ax.set_title('Rotation of Point in 3D')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# Add a legend
ax.legend()
# Show the plot
plt.show()
```

OUTPUT:



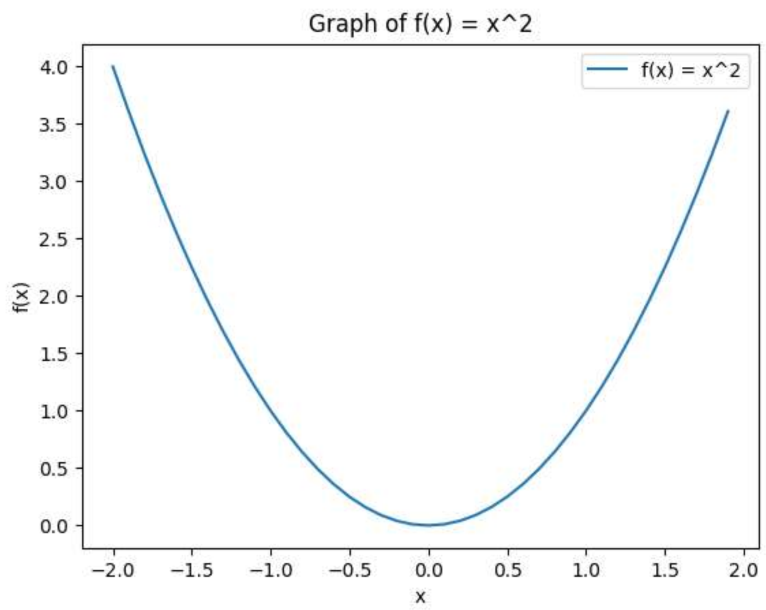
Q.3) Using Python plot the graph of function f(x) = x**2 on the interval (-2,2).

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the function f(x) = x^2
```

```
def f(x):
    return x**2
# Generate x values in the range (-2,2) with a step of 0.1
x = np.arange(-2, 2, 0.1)
# Calculate y values using the function f(x)
y = f(x)
# Create the plot
plt.plot(x, y, label='f(x) = x^2')
# Set the plot title and axis labels
plt.title('Graph of f(x) = x^2')
plt.xlabel('x')
plt.ylabel('f(x)')
# Add a legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:



Q.4) Write a python program to rotate the segment by 180° having endpoints (1,0) and (2,-1)

Syntax:

```
import math
# Define the endpoints of the line segment
x1, y1 = 1, 0
x2, y2 = 2, -1
# Perform the rotation
x1_rotated = -x1
y1_rotated = -y1
x2_rotated = -x2
y2_rotated = -y2
# Print the original and rotated endpoints
print("Original Endpoint 1: ({}, {})".format(x1, y1))
print("Original Endpoint 2: ({}, {})".format(x2, y2))
print("Rotated Endpoint 1: ({}, {})".format(x1_rotated, y1_rotated))
print("Rotated Endpoint 2: ({}, {})".format(x2_rotated, y2_rotated))
```

Output:

```
Original Endpoint 1: (1, 0)
Original Endpoint 2: (2, -1)
Rotated Endpoint 1: (-1, 0)
Rotated Endpoint 2: (-2, 1)
```

Q.5) Write a python program to draw a polygon with 8 sides and radius 5 centered at origin and find its area and perimeter

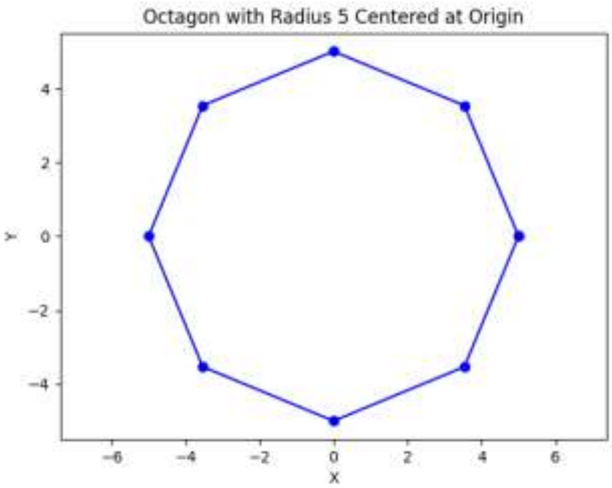
Syntax:

```
import matplotlib.pyplot as plt

import numpy as np

# Number of sides in the polygon

num_sides = 8

# Radius of the polygon

radius = 5

# Calculate the angle between each pair of vertices
```

```
angle = 2 * np.pi / num_sides
# Generate the x and y coordinates of the vertices
x = [radius * np.cos(i * angle) for i in range(num_sides)]
y = [radius * np.sin(i * angle) for i in range(num_sides)]
# Add the first vertex again to close the polygon
x.append(x[0])
y.append(y[0])
# Plot the polygon
plt.plot(x, y, 'bo-') # 'bo-' specifies blue color, circle marker, and solid line
# Set the aspect ratio to 'equal' to ensure the polygon is displayed as a regular shape
plt.axis('equal')
# Set the labels for the axes
plt.xlabel('X')
plt.ylabel('Y')
# Set the title of the plot
plt.title('Octagon with Radius 5 Centered at Origin')
# Show the plot
plt.show()
# Calculate the area of the polygon
area = 0.5 * num_sides * radius ** 2 * np.sin(angle)
# Calculate the perimeter of the polygon
perimeter = num_sides * radius
# Print the calculated area and perimeter
print('Area of the octagon:', area)
print('Perimeter of the octagon:', perimeter)
```

Output:

```
Area of the octagon: 70.71067811865476
Perimeter of the octagon: 40
```



Q.6) Write a python program to find the area and perimeter of the XYZ, where X(l, 2), Y(2, -2), Z(-1,2).

Synatx:

```
import math

# Input coordinates

X = [1, 2]

Y = [2, -2]

Z = [-1, 2]

# Calculate distances between points

def distance(p1, p2):

    return math.sqrt((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2)

# Calculate lengths of sides

XY = distance(X, Y)

YZ = distance(Y, Z)

XZ = distance(X, Z)

# Calculate perimeter

perimeter = XY + YZ + XZ

# Calculate area using Heron's formula

s = perimeter / 2

area = math.sqrt(s * (s - XY) * (s - YZ) * (s - XZ))

# Print results

print("Length of XY: ", XY)
```

```
print("Length of YZ: ", YZ)
print("Length of XZ: ", XZ)
print("Perimeter: ", perimeter)
print("Area: ", area))
```

OUTPUT:

Length of XY: 4.123105625617661

Length of YZ: 5.0

Length of XZ: 2.0

Perimeter: 11.123105625617661

Area: 4.000000000000003

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

Max Z = 3.5x +2y
Subjected to
x + y >= 5
x >= 4
y <= 2
x > 0 , y > 0

Syntax:

```
from pulp import *
# Create the problem
prob = LpProblem("Linear Programming Problem", LpMaximize)
# Define the decision variables
x = LpVariable("x", lowBound=0)
y = LpVariable("y", lowBound=0)
# Define the objective function
objective = 3.5 * x + 2 * y
prob += objective
# Define the constraints
prob += x + y >= 5
prob += x >= 4
prob += y <= 2
# 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("Optimal Objective Value: Z =", value(objective))
```

OUTPUT:

Status: Unbounded
Optimal Solution:
x = 5.0
y = 0.0
Optimal Objective Value: Z = 17.5

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

Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("Minimization Problem", LpMinimize)
# Define decision variables
x = LpVariable("x", lowBound=0, cat='Continuous')
y = LpVariable("y", lowBound=0, cat='Continuous')
z = LpVariable("z", lowBound=0, cat='Continuous')
# Define the objective function
prob += 3*x + 5*y + 4*z, "Z"
# Define the constraints
prob += 2*x + 3*y <= 8, "Constraint 1"
prob += 2*y + 5*z <= 10, "Constraint 2"
prob += 3*x + 2*y + 4*z <= 15, "Constraint 3"
# Solve the problem
prob.solve()
# Print the status of the problem
print("Status:", LpStatus[prob.status])
# Print the optimal solution
```

```
print("Optimal Solution:")
print("x =", value(x))
print("y =", value(y))
print("z =", value(z))
# Print the optimal objective value
print("Z =", value(prob.objective))
```

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

Q.9) Write a python program lo apply the following transformation on the point (-2, 4)

(I) Reflection through y – axis
(II) Scaling in X – coordinate by 6 factor
(III) Scaling in Y – coordinate by factor 4.1
(IV) Shearing in X Direction by 7/2 units

Syntax:

```
# Initial point
x = -2
y = 4
# (I) Reflection through y-axis
print("Point after reflection through y-axis:")
x = -x
y = y
print("x =", x)
print("y =", y)
# (II) Scaling in X-coordinate by 6 factor
print("\nPoint after scaling in X-coordinate by 6 factor:")
x = x * 6
y = y
print("x =", x)
print("y =", y)
# (III) Scaling in Y-coordinate by factor 4.1
print("\nPoint after scaling in Y-coordinate by factor 4.1:")
x = x
y = y * 4.1
print("x =", x)
print("y =", y)
```

```
# (IV) Shearing in X Direction by 7/2 units
print("\nPoint after shearing in X Direction by 7/2 units:")
x = x + (7/2) * y
y = y
print("x =", x)
print("y =", y)
```

OUTPUT:

```
Point after reflection through y-axis:
x = 2
y = 4
Point after scaling in X-coordinate by 6 factor:
x = 12
y = 4
Point after scaling in Y-coordinate by factor 4.1:
x = 12
y = 16.4
Point after shearing in X Direction by 7/2 units:
x = -55.7
y = 16.4
```

Q.10) Find the combined transformation on line segment between the point A[4,1] & B[-3,0] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi/4.
(II) Uniform scaling by 7.3units
(III) Scaling in X Coordinate by 3 units.
(IV) Shearing in X – Direction by 1/2 unit.

Syntax:

```
import numpy as np
# Initial points
A = np.array([4, 1])
B = np.array([-3, 0])
# (I) Rotation about origin through an angle pi/4
theta = np.pi/4
rot_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                      [np.sin(theta), np.cos(theta)]])
A = np.dot(rot_matrix, A)
B = np.dot(rot_matrix, B)
print("Points after rotation about origin through angle pi/4:")
print("A =", A)
print("B =", B)
```

```
# (II) Uniform scaling by 7.3 units
scale_factor = 7.3
A = A * scale_factor
B = B * scale_factor
print("\nPoints after uniform scaling by 7.3 units:")
print("A =", A)
print("B =", B)
# (III) Scaling in X Coordinate by 3 units
scale_x = 3
A[0] = A[0] * scale_x
B[0] = B[0] * scale_x
print("\nPoints after scaling in X Coordinate by 3 units:")
print("A =", A)
print("B =", B)
# (IV) Shearing in X Direction by 1/2 unit
shear_x = 1/2
A[0] = A[0] + shear_x * A[1]
B[0] = B[0] + shear_x * B[1]
print("\nPoints after shearing in X Direction by 1/2 unit:")
print("A =", A)
print("B =", B)
```

OUTPUT:

```
Points after rotation about origin through angle pi/4:
A = [2.12132034 3.53553391]
B = [-2.12132034 -2.12132034]
Points after uniform scaling by 7.3 units:
A = [15.48563851 25.80939751]
B = [-15.48563851 -15.48563851]
Points after scaling in X Coordinate by 3 units:
A = [46.45691552 25.80939751]
B = [-46.45691552 -15.48563851]
Points after shearing in X Direction by 1/2 unit:
A = [59.36161428 25.80939751]
B = [-54.19973478 -15.48563851]
```