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 13 **Expt. No . 13** **Class :-** S.Y.BCS

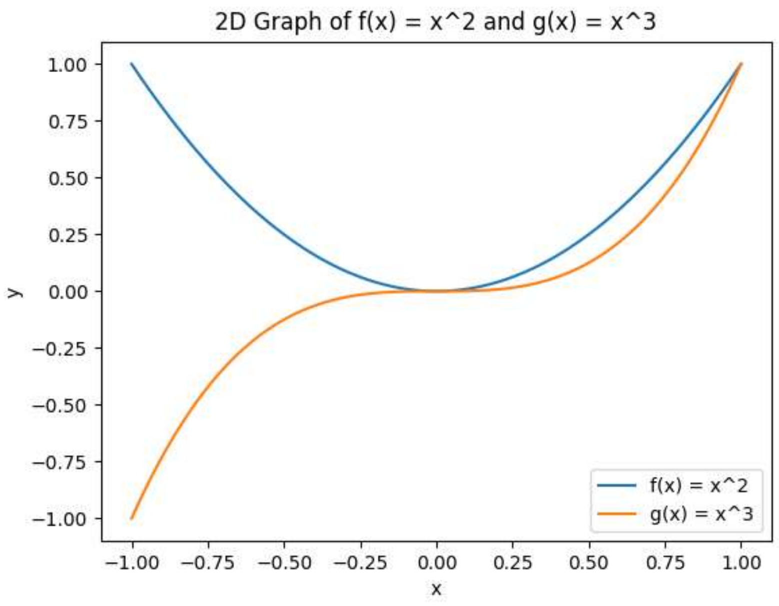
Q.1) Write a Python program to plot 2D graph of the functions f(x) = x^2 and g(x) = x^3 in [-1, 1]

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
def f(x):
   return x**2
def g(x):
   return x**3
# Generate x values in the range [-1, 1]
x = np.linspace(-1, 1, 100)
# Calculate y values for f(x) and g(x)
y_f = f(x)
y_g = g(x)
# Create a figure and axes
fig, ax = plt.subplots()
# Plot f(x) and g(x) on the same graph
ax.plot(x, y_f, label='f(x) = x^2')
ax.plot(x, y_g, label='g(x) = x^3')
# Add labels and legend
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend()
```

```
# Set title
ax.set_title('2D Graph of f(x) = x^2 and g(x) = x^3')
# Show the plot
plt.show()
```

OUTPUT:



Q.2) Using Python, plot the surface plot of parabola z = x**2 + y**2 in -6<x,y<6

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate values for x and y
x = np.linspace(-6, 6, 100)
y = np.linspace(-6, 6, 100)
X, Y = np.meshgrid(x, y)
# Calculate values for z based on the parabola equation
Z = X**2 + Y**2
# Create a 3D figure
```

fig = plt.figure()

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

# Plot the surface plot

surf = ax.plot_surface(X, Y, Z, cmap='viridis')

# Set labels for x, y, z axes

ax.set_xlabel('X')

ax.set_ylabel('Y')

ax.set_zlabel('Z')

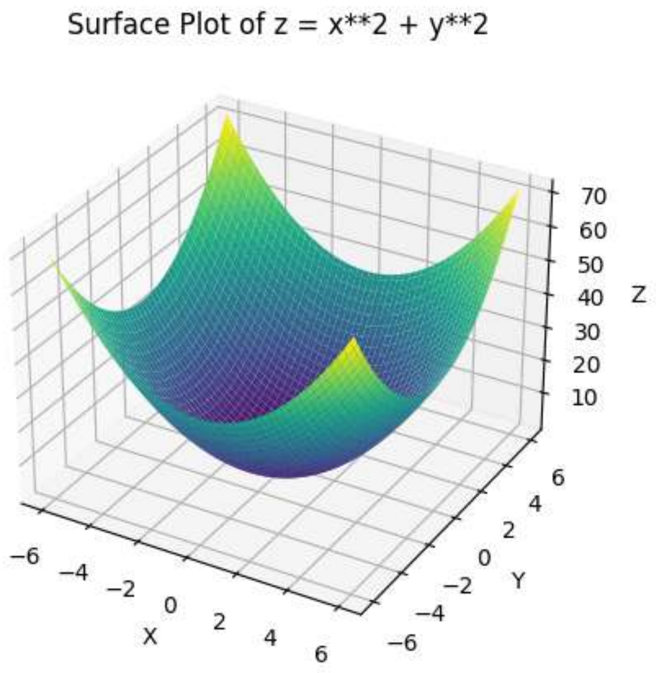
# Set title for the graph

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

# Show the plot

plt.show()

OUTPUT:



Q.3) Write a Python program to plot 3D line graph Whose parametric equation is (cos(2x),sin(2x),x) for 10 <= x <= 20 (in red color), with title of the graph

Syntax:

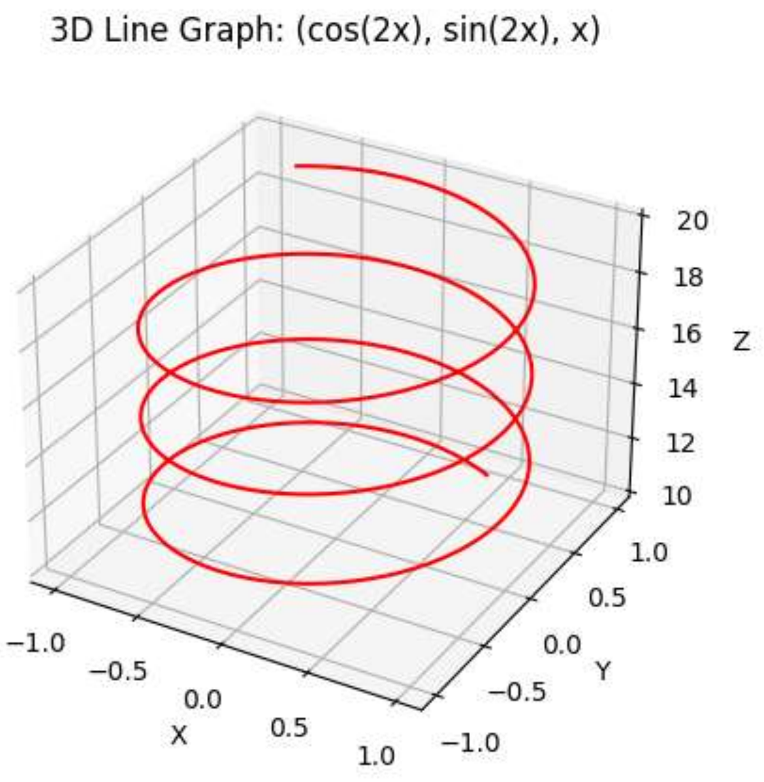
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

```
# Generate values for x
x = np.linspace(10, 20, 500)
# Calculate parametric equations for x, y, z
y = np.sin(2 * x)
z = x
x = np.cos(2 * x)
# Create a 3D figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the 3D line graph
ax.plot(x, y, z, color='red')
# Set title for the graph
ax.set_title("3D Line Graph: (cos(2x), sin(2x), x)")
# Set labels for x, y, z axes
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# Show the plot
plt.show()
```

OUTPUT:



Q.4) Write a python program to reflect the ABC through the line y = 3 where A(1,0),D(2, -1),C(-1,3).

Syntax:

```
def reflect_point(point, line_y):
    x, y = point
    y_reflected = 2 * line_y - y
    return x, y_reflected
# Define the points A, D, and C
A = (1, 0)
D = (2, -1)
C = (-1, 3)
# Define the line of reflection
line_y = 3
# Reflect the points A, D, and C through the line of reflection
A_reflected = reflect_point(A, line_y)
D_reflected = reflect_point(D, line_y)
C_reflected = reflect_point(C, line_y)
# Print the reflected points
print("Original Points:")
print("A:", A)
print("D:", D)
print("C:", C)
print("Reflected Points:")
print("A_reflected:", A_reflected)
print("D_reflected:", D_reflected)
print("C_reflected:", C_reflected)
```

Output:

```
Original Points:
A: (1, 0)
D: (2, -1)
C: (-1, 3)
Reflected Points:
A_reflected: (1, 6)
D_reflected: (2, 7)
C_reflected: (-1, 3)
```

Q.5) Using sympy declare the points P(5, 2), Q(5, -2), R(5, O), 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 *
# Declare the points P, Q, and R
P = Point(5, 2)
Q = Point(5, -2)
R = Point(5, Symbol('O'))
# Check if points P, Q, and R 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.")
# Declarethe ray passing through points P and Q
ray_PQ = Ray(P, Q)
# Find the length of the ray between points P and Q
length_PQ = ray_PQ.length
print("Length of ray PQ between points P and Q:", length_PQ)
# Find the slope of the ray PQ
slope_PQ = ray_PQ.slope
print("Slope of ray PQ:", slope_PQ)
```

OUTPUT:

```
Points P, Q, and R are collinear.
Length of ray PQ between points P and Q: 00
Slope of ray PQ: 00
```

Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[5, 0], C[3,3].

Synatx:

```
import numpy as np

# Define the vertices of the triangle

A = np.array([0, 0])
```

```
B = np.array([4, 0])
C = np.array([3, 3])
# 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: 4.0
Side BC: 3.1622776601683795
Side CA: 4.242640687119285
Area: 6.0000000000000036
Perimeter: 11.404918347287666
```

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 = 3x +2y +5z

subject to

x + 2y + z <= 430

3x + 2z <= 460

x + 4y <= 120

x=>0, y=>0,z=>0

Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("Linear Programming Problem", LpMinimize)
# Define decision variables
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
z = LpVariable('z', lowBound=0)
# Define the objective function
prob += 3 * x + 2 * y + 5 * z
# Define the constraints
```

```
prob += x + 2 * y + z <= 430
prob += 3 * x + 2 * z <= 460
prob += x + 4 * y <= 120
# Solve the problem
prob.solve()
# Print the status of the problem
print("Status:", LpStatus[prob.status])
# If the problem is solved, print the optimal solution and its value
if prob.status == LpStatusOptimal:
    print("Optimal Solution:")
    print("x =", value(x))
    print("y =", value(y))
    print("z =", value(z))
    print("Objective Value =", value(prob.objective))
else:
    print("No optimal solution found.")
```

OUTPUT:

```
Status: Optimal
Optimal Solution:
x = 0.0
y = 0.0
z = 0.0
Objective Value = 0.0
```

Q.9) Apply Python. Program in each of the following transformation on the point P[-2,4]
(I)Shearing in Y direction by 7 units
(II)Scaling in X- and Y co-ordinate by 7/2 and 7 units respectively.
(III) Scaling in X- and Y co-ordinate by 4 and 7 units respectively.
(IV) Rotation about origin by an angle 60^0

Syntax:

```
import math
# Original point P
P = [-2, 4]
# Transformation 1: Shearing in Y direction by 7 units
shear_y = 7
P_sheared_y = [P[0], P[1] + shear_y]
print("Point after Shearing in Y direction by 7 units:", P_sheared_y)
# Transformation 2: Scaling in X- and Y-coordinate by 7/2 and 7 units respectively
scale_x_1 = 7/2
scale_y_1 = 7
P_scaled_1 = [P_sheared_y[0] * scale_x_1, P_sheared_y[1] * scale_y_1]
print("Point after Scaling in X- and Y-coordinate by 7/2 and 7 units respectively:", P_scaled_1)
# Transformation 3: Scaling in X- and Y-coordinate by 4 and 7 units respectively
scale_x_2 = 4
scale_y_2 = 7
P_scaled_2 = [P_scaled_1[0] * scale_x_2, P_scaled_1[1] * scale_y_2]
print("Point after Scaling in X- and Y-coordinate by 4 and 7 units respectively:", P_scaled_2)
# Transformation 4: Rotation about origin by an angle of 60 degrees
angle = 60
angle_rad = math.radians(angle)
P_rotated = [P_scaled_2[0] * math.cos(angle_rad) - P_scaled_2[1] * math.sin(angle_rad), P_scaled_2[0] * math.sin(angle_rad) + P_scaled_2[1] * math.cos(angle_rad)]
print("Point after Rotation about origin by an angle of 60 degrees:", P_rotated)
```

OUTPUT:

```
Point after Shearing in Y direction by 7 units: [-2, 11]
Point after Scaling in X- and Y-coordinate by 7/2 and 7 units respectively: [-7.0, 77]
Point after Scaling in X- and Y-coordinate by 4 and 7 units respectively: [-28.0, 539]
Point after Rotation about origin by an angle of 60 degrees: [-480.7876926398124, 245.25128869403576]
```

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]

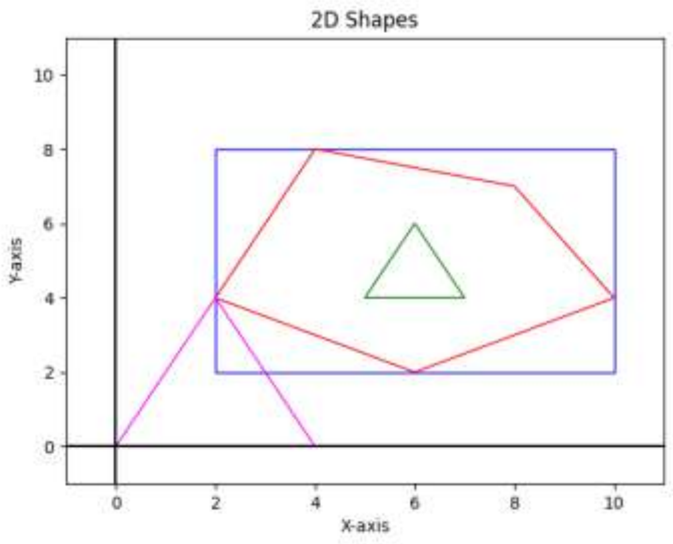
(II) Blue rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8].

(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
X-axis
Y-axis