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

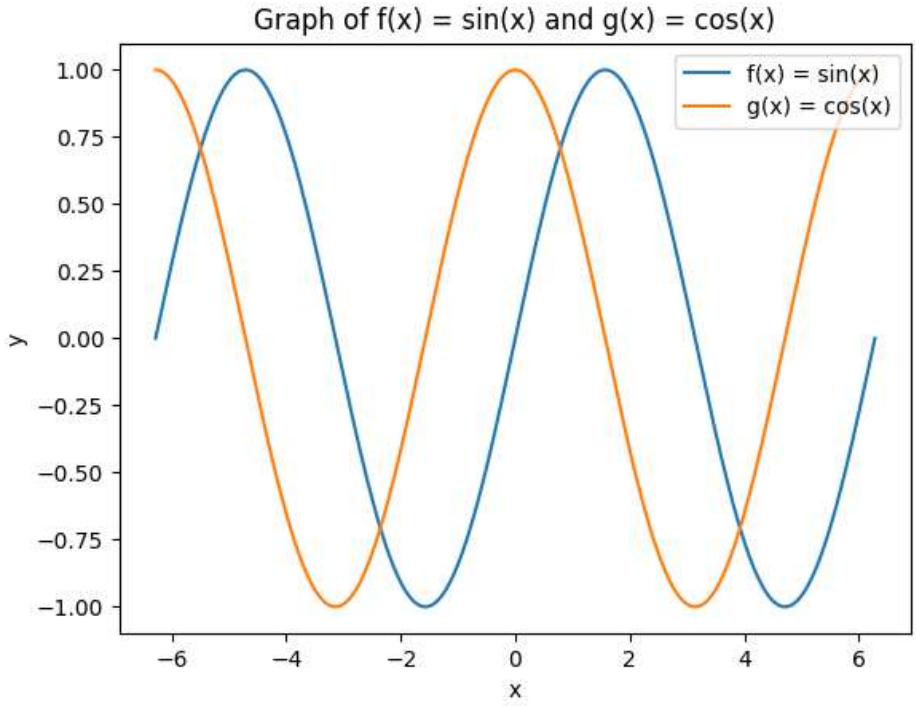
Title of the:- Practical 20 **Expt. No .** 20 **Class :-** S.Y.BCS

Q.1) Write a Python program to plot 2D graph of the function f(x)= sin (x) and g(x) = cos (x) in [-2*pi,2*pi]

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the range of x values
x = np.linspace(-2 * np.pi, 2 * np.pi, 1000)
# Compute the y values for f(x) = sin(x) and g(x) = cos(x)
f_x = np.sin(x)
g_x = np.cos(x)
# Create a figure and axis
fig, ax = plt.subplots()
# Plot f(x) = sin(x)
ax.plot(x, f_x, label='f(x) = sin(x)')
# Plot g(x) = cos(x)
ax.plot(x, g_x, label='g(x) = cos(x)')
# Set the title and labels for x and y axes
ax.set_title('Graph of f(x) = sin(x) and g(x) = cos(x)')
ax.set_xlabel('x')
ax.set_ylabel('y')
# Add a legend
ax.legend()
# Show the plot
plt.show()
```

OUTPUT:



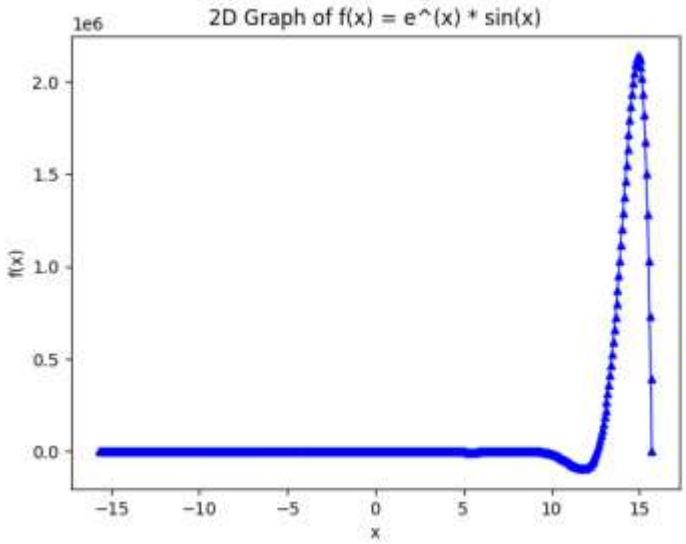
Q.2) Write n Python program to plot the 2D graph of the function f(x)=e(x)sin(x) in [-5*pi,5*pi] with blue points line with upward point line with upward pointing triangle.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the function f(x)
def f(x):
    return np.exp(x) * np.sin(x)
# Generate x values in the range [-5*pi, 5*pi]
x = np.linspace(-5*np.pi, 5*np.pi, 500)
# Calculate y values using the function f(x)
y = f(x)
# Plot the graph with blue points and a line with upward pointing triangles
```

```
plt.plot(x, y, 'b^-', linewidth=1, markersize=4)
# Set x and y axis labels
plt.xlabel('x')
plt.ylabel('f(x)')
# Set the title of the graph
plt.title('2D Graph of f(x) = e^(x) * sin(x)')
# Show the graph
plt.show()
```

OUTPUT:



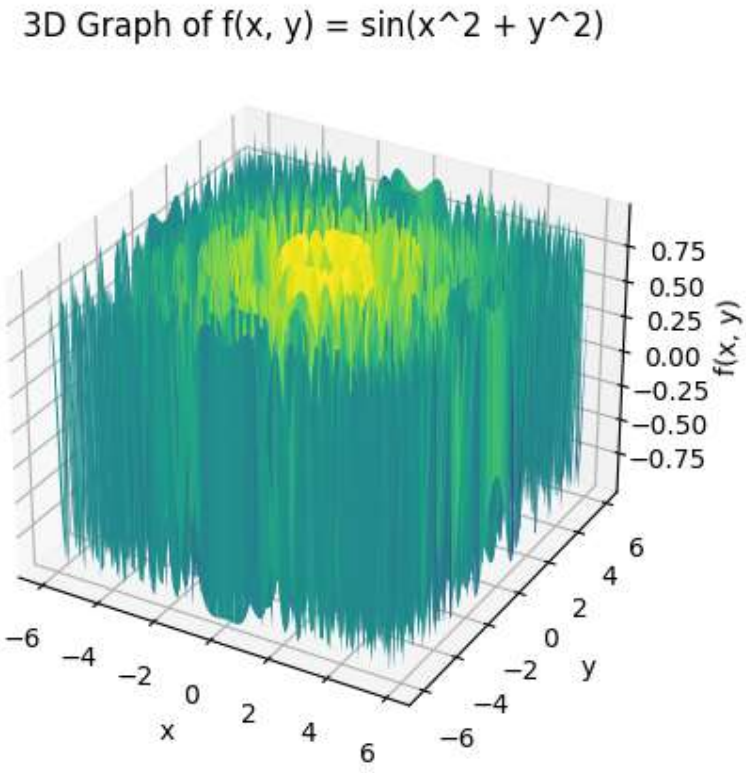
Q.3) Write a Python program to plot the 3D graph of the function f(x) = sin(x^2 + y^2), -6< x,y < 6.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the function f(x, y)
def f(x, y):
    return np.sin(x**2 + y**2)
# Generate x, y values in the range -6 to 6 with a step of 0.1
x = np.arange(-6, 6, 0.1)
```

```
y = np.arange(-6, 6, 0.1)
# Create a meshgrid from x, y values
X, Y = np.meshgrid(x, y)
# Calculate z values using the function f(x, y)
Z = f(X, Y)
# Create a 3D figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the 3D surface
ax.plot_surface(X, Y, Z, cmap='viridis')
# Set x, y, z axis labels
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
# Set the title of the graph
ax.set_title('3D Graph of f(x, y) = sin(x^2 + y^2)')
# Show the graph
plt.show()
```

OUTPUT:

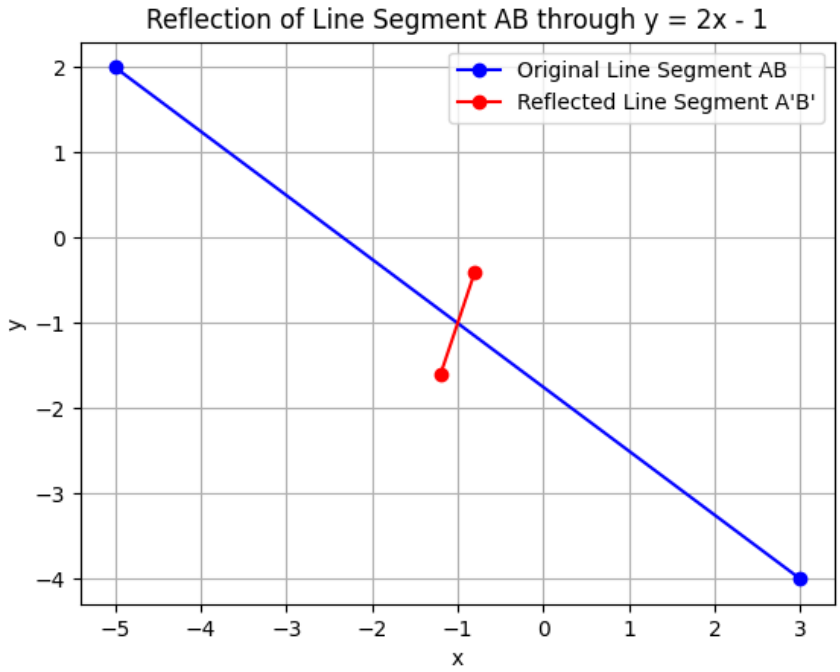


Q.4) Write a python program to reflect the line segment joining the points A[-5, 2], B[3, -4] through the line y = 2x - 1.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the line segment endpoints A and B
A = np.array([-5, 2])
B = np.array([3, -4])
# Define the reflection line y = 2x - 1
m = 2  # slope of the reflection line
c = -1  # y-intercept of the reflection line
# Calculate the midpoint of the line segment AB
midpoint = (A + B) / 2
# Calculate the direction vector of the reflection line
direction = np.array([1, m])
# Calculate the projection of the midpoint onto the reflection line
projection = (2 * midpoint.dot(direction) - 2 * c * direction) / (1 + m**2)
# Calculate the reflected point of A with respect to the reflection line
reflected_A = midpoint + (projection - midpoint)
# Calculate the reflected point of B with respect to the reflection line
reflected_B = midpoint - (projection - midpoint)
# Plot the original line segment AB and the reflected line segment A'B'
plt.plot([A[0], B[0]], [A[1], B[1]], 'bo-', label='Original Line Segment AB')
plt.plot([reflected_A[0], reflected_B[0]], [reflected_A[1], reflected_B[1]], 'ro-', label='Reflected Line Segment A\'B\'')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.title('Reflection of Line Segment AB through y = 2x - 1')
plt.grid()
plt.show()
```

Output:



Q.5) Write a Python program lo find the area and perimeter of a polygon with vertices (0, 0), (-2, 0), (5,5), (1, -1)

Syntax:

```
import math
# Define the vertices of the polygon
vertices = [(0, 0), (-2, 0), (5, 5), (1, -1)]
# Calculate the area of the polygon using Shoelace formula
def calculate_area(vertices):
    area = 0
    for i in range(len(vertices)):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % len(vertices)]
        area += (x1 * y2 - x2 * y1)
    return abs(area) / 2
# Calculate the perimeter of the polygon
def calculate_perimeter(vertices):
    perimeter = 0
    for i in range(len(vertices)):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % len(vertices)]
        perimeter += math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
```

```
    return perimeter
# Call the functions to calculate area and perimeter
area = calculate_area(vertices)
perimeter = calculate_perimeter(vertices)
# Print the results
print("Area of the polygon: ", area)
print("Perimeter of the polygon: ", perimeter)
```

OUTPUT:

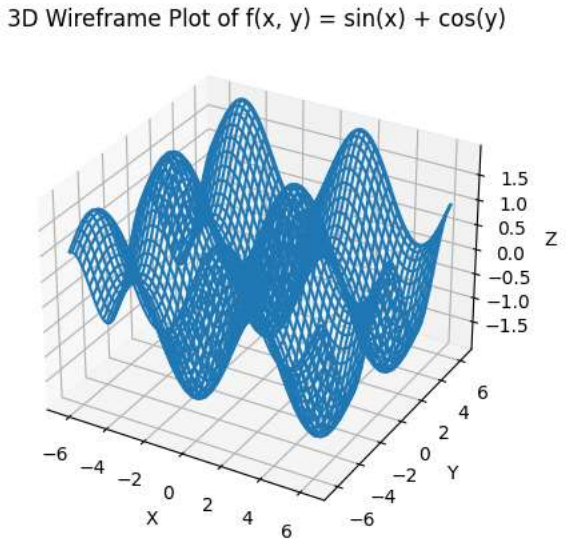
```
Area of the polygon:  10.0
Perimeter of the polygon:  19.2276413803437
```

Q.6) Write a. Python program to plot the 3D graph of the function f(x, y) = sin x + cos y, x, y belongs [-2*pi,2*pi] using wireframe plot.

```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

# Define the range of x and y values

x = np.linspace(-2 * np.pi, 2 * np.pi, 100)

y = np.linspace(-2 * np.pi, 2 * np.pi, 100)

# Create a meshgrid from x and y

X, Y = np.meshgrid(x, y)

# Calculate the Z values using the function f(x, y) = 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')

# Create a wireframe plot

ax.plot_wireframe(X, Y, Z)

# Set labels and title

ax.set_xlabel('X')

ax.set_ylabel('Y')
```

```
ax.set_zlabel('Z')
ax.set_title('3D Wireframe Plot of f(x, y) = sin(x) + cos(y)')
# 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+ y
subject to
x - y >= 1
x + y >= 2
x>= 0, y>= 0

Syntax:

```
from pulp import *
# Create a LP Minimization problem
problem = LpProblem("LPP", LpMinimize)
# Define the decision variables
x = LpVariable("x", lowBound=0)  # x >= 0
y = LpVariable("y", lowBound=0)  # y >= 0
# Define the objective function
```

```
problem += x + y
# Define the constraints
problem += x - y >= 1
problem += x + y >= 2
# Solve the problem using the simplex method
status = problem.solve()
# Check if the problem has an optimal solution
if status == LpStatusOptimal:
    # Print the optimal solution
    print("Optimal Solution:")
    print("x =", value(x))
    print("y =", value(y))
    print("Z =", value(problem.objective))
else:
    print("No Optimal Solution")
```

OUTPUT:

Optimal Solution:

x = 2.0

y = 0.0

Z = 2.0

Q.9) Apply Python. Program in each of the following transformation on the point P[3,-2]

(I) Scaling in y direction by 4 unit.

(II) Reflection through y axis.

(III) Rotation about origin by angle 45° .

(IV) Reflection through the line y = x.

Syntax:

```
import numpy as np
# Given point P
P = np.array([3, -2])
# Transformation I: Scaling in y direction by 4 units
scaling_matrix = np.array([[1, 0], [0, 4]])
P_scaled_y = np.dot(scaling_matrix, P)
print("After Scaling in y direction by 4 units:")
print("Point P_scaled_y:", P_scaled_y)
# Transformation II: Reflection through y axis
reflection_y_axis_matrix = np.array([[-1, 0], [0, 1]])
P_reflected_y_axis = np.dot(reflection_y_axis_matrix, P)
print("After Reflection through y axis:")
```

```
print("Point P_reflected_y_axis:", P_reflected_y_axis)
# Transformation III: Rotation about origin by angle 45 degrees
angle_rad = np.deg2rad(45)  # Convert angle to radians
rotation_matrix = np.array([[np.cos(angle_rad), -np.sin(angle_rad)],
[np.sin(angle_rad), np.cos(angle_rad)]])
P_rotated = np.dot(rotation_matrix, P)
print("After Rotation about origin by angle 45 degrees:")
print("Point P_rotated:", P_rotated)
# Transformation IV: Reflection through the line y = x
reflection_line_matrix = np.array([[0, 1], [1, 0]])
P_reflected_line = np.dot(reflection_line_matrix, P)
print("After Reflection through the line y = x:")
print("Point P_reflected_line:", P_reflected_line)
```

OUTPUT:

```
Point P_scaled_y: [ 3 -8]
After Reflection through y axis:
Point P_reflected_y_axis: [-3 -2]
After Rotation about origin by angle 45 degrees:
Point P_rotated: [3.53553391 0.70710678]
After Reflection through the line y = x:
Point P_reflected_line: [-2  3]
```

Q.10) Apply the following transformation on the point P[3, -2]

(I) Shearing in x direction by -2 units.

(II) Scaling in X and y direction by -2 and 2 units respectively

(III) Reflection through x axis.

(IV) Reflection through the line y = -x

Syntax:

```
import numpy as np
# Given point P
P = np.array([3, -2])
# Transformation I: Shearing in x direction by -2 units
shearing_x_matrix = np.array([[1, -2], [0, 1]])
P_sheared_x = np.dot(shearing_x_matrix, P)
print("After Shearing in x direction by -2 units:")
print("Point P_sheared_x:", P_sheared_x)
# Transformation II: Scaling in x and y direction by -2 and 2 units respectively
scaling_matrix = np.array([[-2, 0], [0, 2]])
P_scaled_xy = np.dot(scaling_matrix, P)
print("After Scaling in x and y direction by -2 and 2 units respectively:")
```

```
print("Point P_scaled_xy:", P_scaled_xy)
# Transformation III: Reflection through x axis
reflection_x_axis_matrix = np.array([[1, 0], [0, -1]])
P_reflected_x_axis = np.dot(reflection_x_axis_matrix, P)
print("After Reflection through x axis:")
print("Point P_reflected_x_axis:", P_reflected_x_axis)
# Transformation IV: Reflection through the line y = -x
reflection_line_matrix = np.array([[0, -1], [-1, 0]])
P_reflected_line = np.dot(reflection_line_matrix, P)
print("After Reflection through the line y = -x:")
print("Point P_reflected_line:", P_reflected_line)
```

OUTPUT:

```
Line segment after applying the sequence of transformations:
After Scaling in y direction by 4 units:
Point P_scaled_y: [ 3 -8]
After Reflection through y axis:
Point P_reflected_y_axis: [-3 -2]
After Rotation about origin by angle 45 degrees:
Point P_rotated: [3.53553391 0.70710678]
After Reflection through the line y = x:
Point P_reflected_line: [-2  3]
PS E:\Python 2nd Sem Practical> python -u "e:\Python 2nd Sem Practical\tempCodeRunnerFile.py"
After Shearing in x direction by -2 units:
Point P_sheared_x: [ 7 -2]
After Scaling in x and y direction by -2 and 2 units respectively:
Point P_scaled_xy: [-6 -4]
After Reflection through x axis:
Point P_reflected_x_axis: [3 2]
After Reflection through the line y = -x:
Point P_reflected_line: [ 2 -3]
```