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 19 **Expt. No .** 19 **Class :-** S.Y.BCS

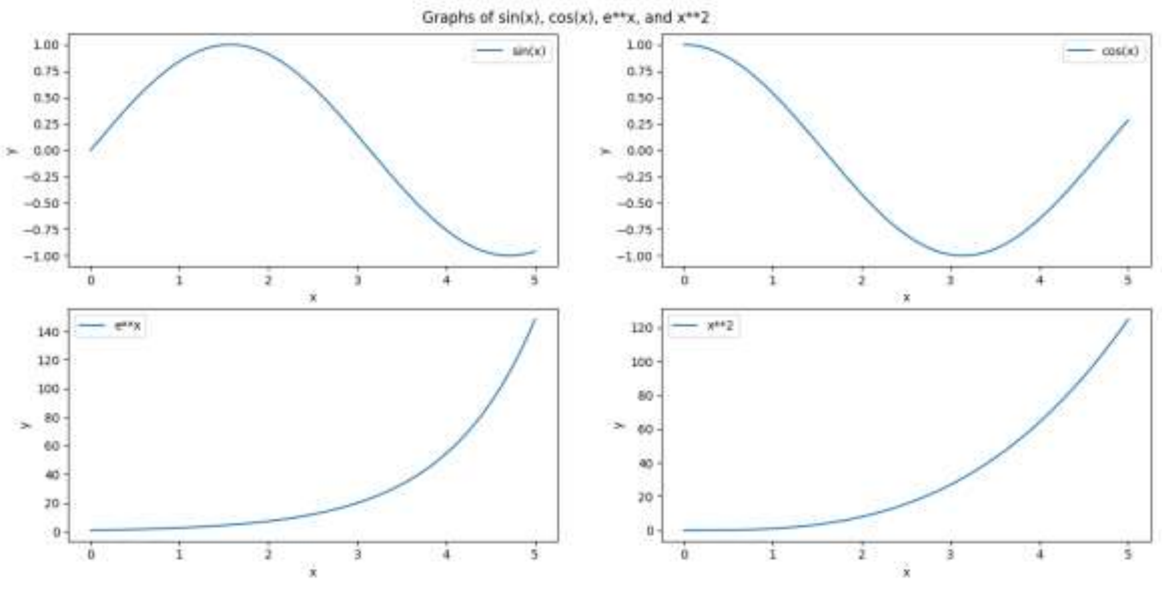
Q.1) Plot the graphs of sin x, cos x, e**x and x**3 in [O, 5] in one figure with (2 x 2) subplot

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**3

# 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:



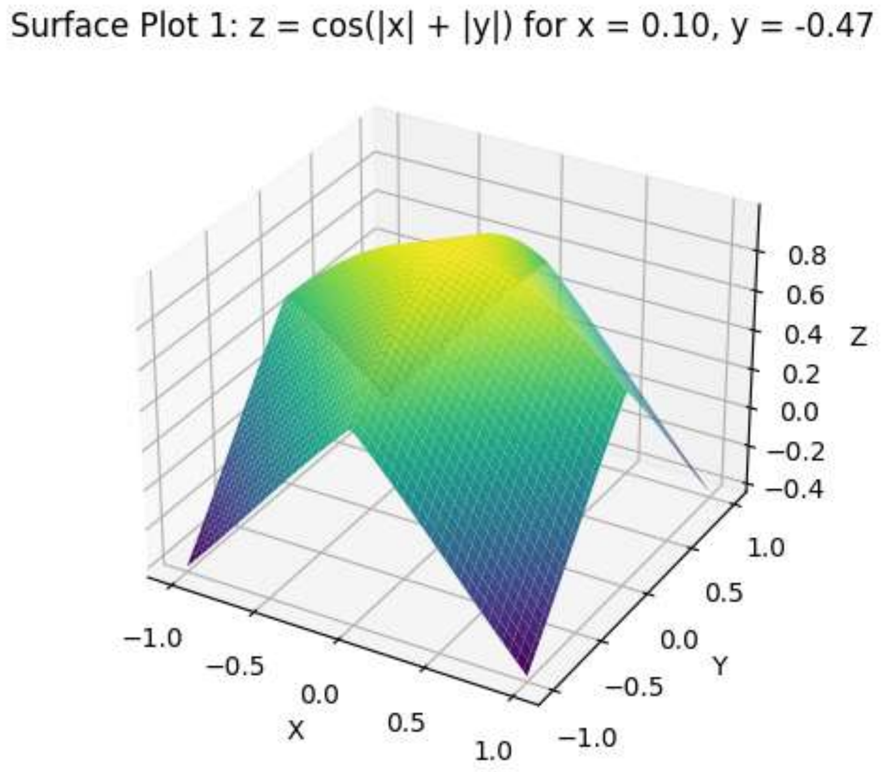
Q.2) Write a python program to plot 30 Surface Plot of the function z = cos(|x| +|y|) in -1 < x,y1 < 1.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Generate 30 random x, y pairs within the range -1 < x, y < 1
```

```
np.random.seed(0)
x_vals = np.random.uniform(-1, 1, size=30)
y_vals = np.random.uniform(-1, 1, size=30)
# Create a 2D grid of x, y values
x, y = np.meshgrid(np.linspace(-1, 1, 100), np.linspace(-1, 1, 100))
# Compute the z values for each x, y pair
z = np.cos(np.abs(x) + np.abs(y))
# Plot the surface plots
for i in range(30):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(x, y, z, cmap='viridis')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title(f'Surface Plot {i+1}: z = cos(|x| + |y|) for x = {x_vals[i]:.2f}, y =
{y_vals[i]:.2f}')
    plt.show()
```

OUTPUT:

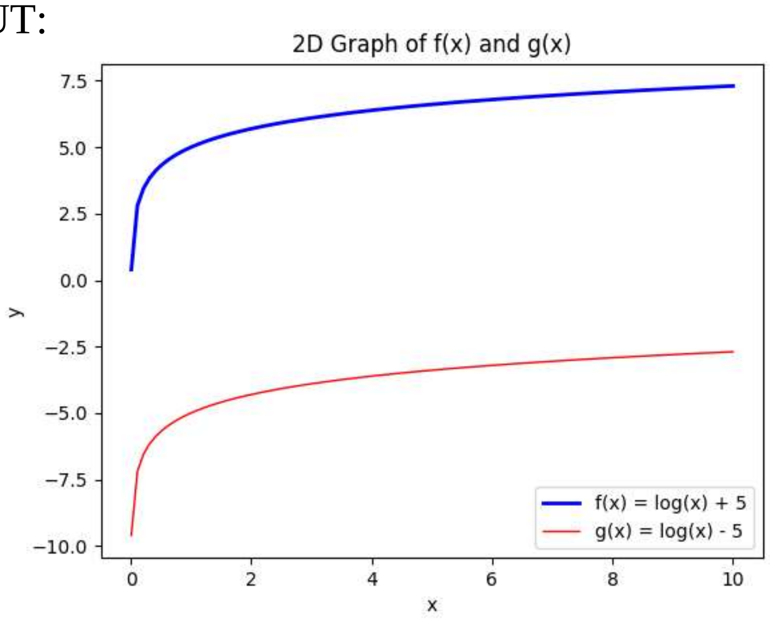


Q.3) Write a python program to plot 2D graph of the functions f(x) = log(x) + 5 and g(x) = log(x) – 5 in [0, 10] by setting different line width and different colors to the curve.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the functions
def f(x):
    return np.log(x) + 5
def g(x):
    return np.log(x) - 5
# Generate x values in the range [0, 10]
x = np.linspace(0.01, 10, 100)
# Compute y values for f(x) and g(x)
y_f = f(x)
y_g = g(x)
# Create the plot
plt.plot(x, y_f, label='f(x) = log(x) + 5', linewidth=2, color='blue')
plt.plot(x, y_g, label='g(x) = log(x) - 5', linewidth=1, color='red')
# Add labels and legend
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
# Set the title and show the plot
plt.title('2D Graph of f(x) and g(x)')
plt.show()
```

OUTPUT:



Q.4) Write a python program to rotate the segment by 90° having endpoints (0,0) and (4,4)

Syntax:

```
import math
# Define the endpoints of the line segment
x1, y1 = 0, 0
x2, y2 = 4, 4
# 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:

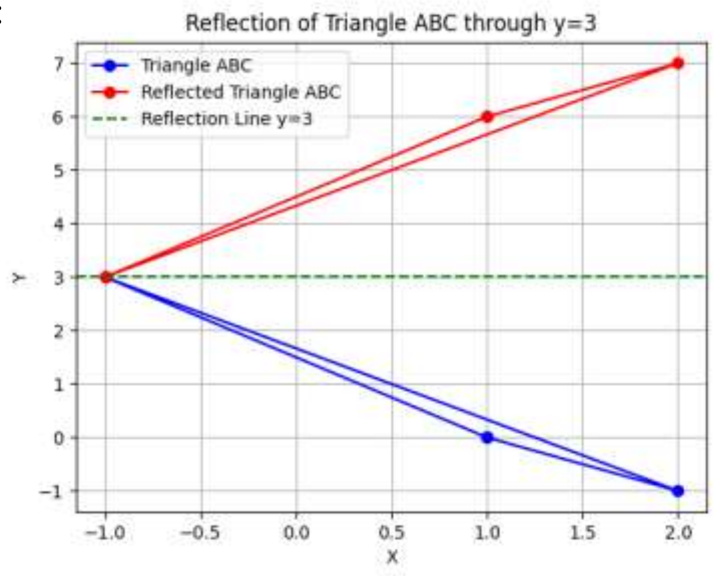
```
Original Endpoint 1: (0, 0)
Original Endpoint 2: (4, 4)
Rotated Endpoint 1: (0, 0)
Rotated Endpoint 2: (-4, -4)
```

Q.5) Write a Python program to Ref1ect the triangle ABC through the line y=3, where A[1, 0], B[2, -1] and C[-1, 3]

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the coordinates of the triangle ABC
A = np.array([1, 0])
B = np.array([2, -1])
C = np.array([-1, 3])
# Define the equation of the reflection line y = 3
reflection_line = 3
# Reflect the triangle ABC through the reflection line
A_reflected = np.array([A[0], 2*reflection_line - A[1]])
B_reflected = np.array([B[0], 2*reflection_line - B[1]])
C_reflected = np.array([C[0], 2*reflection_line - C[1]])
# Plot the original and reflected triangles
plt.plot([A[0], B[0], C[0], A[0]], [A[1], B[1], C[1], A[1]], 'bo-', label='Triangle ABC')
plt.plot([A_reflected[0], B_reflected[0], C_reflected[0], A_reflected[0]],
         [A_reflected[1], B_reflected[1], C_reflected[1], A_reflected[1]], 'ro-', label='Reflected Triangle ABC')
plt.axhline(y=reflection_line, color='g', linestyle='--', label='Reflection Line y=3')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Reflection of Triangle ABC through y=3')
plt.grid(True)
plt.show()
```

Output:



Q.6) Write a Python program to draw a polygon with vertices (0,0),(1,0),(2,2),(1,4). Also find area and perimeter of the polygon.

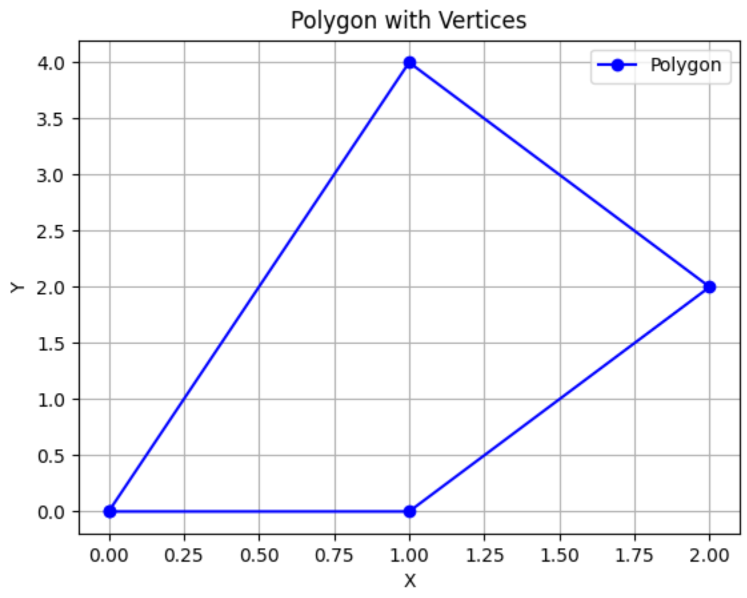
Synatx:

```
import matplotlib.pyplot as plt

# Define the coordinates of the vertices of the polygon

vertices = [(0, 0), (1, 0), (2, 2), (1, 4)]

# Extract the x and y coordinates of the vertices

x = [vertex[0] for vertex in vertices]

y = [vertex[1] for vertex in vertices]

# Plot the polygon

plt.plot(x + [x[0]], y + [y[0]], 'bo-', label='Polygon')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.title('Polygon with Vertices')

plt.grid(True)

plt.show()

# Calculate the area of the polygon using shoelace formula
```

```
area = 0
for i in range(len(vertices)):
    area += (x[i] * y[(i + 1) % len(vertices)]) - (x[(i + 1) % len(vertices)] * y[i])
area /= 2
area = abs(area)
# Calculate the perimeter of the polygon
perimeter = 0
for i in range(len(vertices)):
    dx = x[(i + 1) % len(vertices)] - x[i]
    dy = y[(i + 1) % len(vertices)] - y[i]
    perimeter += ((dx ** 2) + (dy ** 2)) ** 0.5
# Print the area and perimeter of the polygon
print("Area of the Polygon:", area)
print("Perimeter of the Polygon:", perimeter)
```

OUTPUT:

```
Area of the Polygon: 4.0
Perimeter of the Polygon: 9.595241580617241
```



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

Max Z = x + 2y + z
Subjected to
X + 0.5y + 0.5z <= 1
1.5x + 2y + z >= 8
x > 0 , y > 0
Syntax:

```
from scipy.optimize import linprog
# Coefficients of the objective function
c = [1, 2, 1]
# Coefficients of the inequality constraints (LHS matrix)
A = [[1, 0.5, 0.5],
    [-1.5, -2, -1]]
# RHS values of the inequality constraints
b = [1, -8]
# Bounds on the variables (x, y, z)
bounds = [(0, None), (0, None), (0, None)]
# Specify the inequality constraint directions (<=, >=)
# and the corresponding bound values (1, -1)
ineq_ops = ['<=', '>=']
# Solve the linear programming problem
res = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='simplex')
# Print the results
print("Optimization Result:")
print("Objective Value (Z):", res.fun)
print("Optimal Solution (x, y, z):", res.x)
```

OUTPUT:

```
Optimization Result:
Objective Value (Z): 4.0
Optimal Solution (x, y, z): [0. 2. 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 + 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) Rotation about origin through an angle 48 degree
(II) Scaling in X – coordinate by 2 factor
(III) Reflection through the line y = 2x - 3
(IV) Shearing in X Direction by 7 units

Syntax:

```
import numpy as np
# Define the initial point
point = np.array([-2, 4])
# Transformation 1: Rotation about origin through an angle of 48 degrees
angle = np.deg2rad(48)
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                           [np.sin(angle), np.cos(angle)]])
rotated_point = np.dot(rotation_matrix, point)
# Transformation 2: Scaling in X-coordinate by a factor of 2
scaling_factor = np.array([[2, 0],
                          [0, 1]])
scaled_point = np.dot(scaling_factor, rotated_point)
# Transformation 3: Reflection through the line y = 2x - 3
reflection_matrix = np.array([[1, -4],
                             [-4, 1]])
reflected_point = np.dot(reflection_matrix, scaled_point)
# Transformation 4: Shearing in X-direction by 7 units
shearing_factor = np.array([[1, 7],
                           [0, 1]])
sheared_point = np.dot(shearing_factor, reflected_point)
# Print the results
print("Initial Point:", point)
print("Rotated Point:", rotated_point)
print("Scaled Point:", scaled_point)
print("Reflected Point:", reflected_point)
print("Sheared Point:", sheared_point)
```

OUTPUT:

```
Initial Point: [-2  4]
Rotated Point: [-4.31084051  1.19023277]
Scaled Point: [-8.62168103  1.19023277]
Reflected Point: [-13.38261213  35.67695689]
Sheared Point: [236.35608611  35.67695689]
```

Q.10) Find Combined transformation of the line segment between the points A[4,-1] and B[3,0] for the following sequence :
First rotation about origin through an angle pi; followed by scaling in x coordinate by 3 units. Followed by reflection through the line y = x;

Syntax:

```
import numpy as np
```

```
# Define the initial points A and B
A = np.array([4, -1])
B = np.array([3, 0])
# Transformation 1: Rotation about origin through an angle pi (180 degrees)
angle = np.pi
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                            [np.sin(angle), np.cos(angle)]])
rotated_A = np.dot(rotation_matrix, A)
rotated_B = np.dot(rotation_matrix, B)
# Transformation 2: Scaling in x-coordinate by 3 units
scaling_factor = np.array([[3, 0],
                          [0, 1]])
scaled_A = np.dot(scaling_factor, rotated_A)
scaled_B = np.dot(scaling_factor, rotated_B)
# Transformation 3: Reflection through the line y = x
reflection_matrix = np.array([[0, 1],
                              [1, 0]])
reflected_A = np.dot(reflection_matrix, scaled_A)
reflected_B = np.dot(reflection_matrix, scaled_B)
# Print the results
print("Initial Points A and B:")
print("A =", A)
print("B =", B)
print("Combined Transformed Points A and B:")
print("A' =", reflected_A)
print("B' =", reflected_B)
```

OUTPUT:

```
Initial Point: [-2  4]
Rotated Point: [-4.31084051  1.19023277]
Scaled Point: [-8.62168103  1.19023277]
Reflected Point: [-13.38261213  35.67695689]
Sheared Point: [236.35608611  35.67695689]
```