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

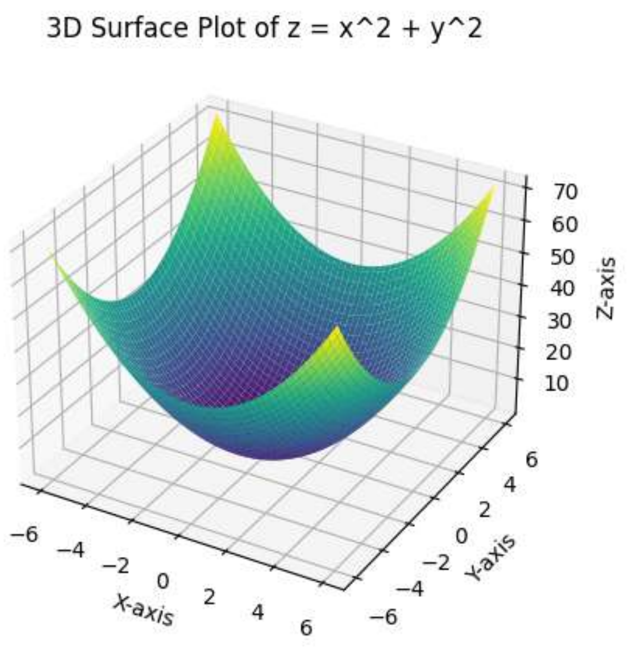
Title of the:- Practical 17 **Expt. No .** 17 **Class :-** S.Y.BCS

Q.1) Write a python program to plot the 3D graph of the function z = x^2 + y^2 in -6 < x, y < 6 using surface plot.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Create a meshgrid for x and y values
x = np.linspace(-6, 6, 100)
y = np.linspace(-6, 6, 100)
X, Y = np.meshgrid(x, y)
# Compute the values of z
Z = X**2 + Y**2
# Create a 3D surface plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
# Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_title('3D Surface Plot of z = x^2 + y^2')
# Show the plot
plt.show()
```

OUTPUT:

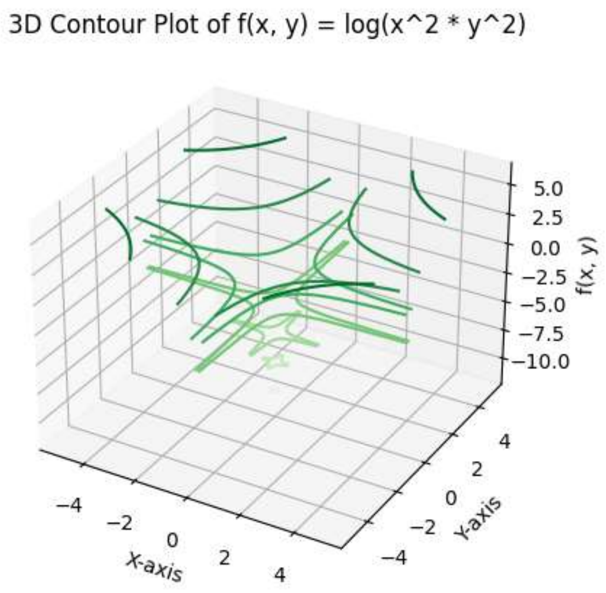


Q.2) Write a python program to plot 3D contours for the function f(x,y) = log(x^2y2) when -5<=x,y<=5 with green color map

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Create a meshgrid for x and y values
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
# Compute the values of f(x, y)
Z = np.log(X**2 * Y**2)
# Create a 3D contour plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.contour(X, Y, Z, cmap='Greens')
# Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('f(x, y)')
ax.set_title('3D Contour Plot of f(x, y) = log(x^2 * y^2)')
# Show the plot
plt.show()
```

OUTPUT:



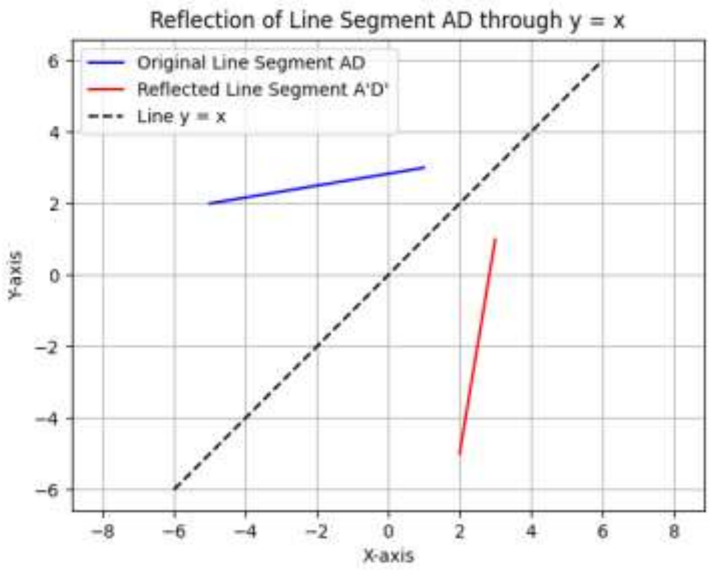
Q.3) Write a Python program to reflect the line segment joining the points A[-5, 2] and D[l, 3] through the line y = x.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the points A and D
A = np.array([-5, 2])
D = np.array([1, 3])
# Define the line y = x
def reflect_y_equals_x(point):
    return np.array([point[1], point[0]])
# Reflect points A and D through the line y = x
A_reflected = reflect_y_equals_x(A)
D_reflected = reflect_y_equals_x(D)
# Plot the original line segment and its reflection
```

```
fig, ax = plt.subplots()
ax.plot([A[0], D[0]], [A[1], D[1]], 'b', label='Original Line Segment AD')
ax.plot([A_reflected[0], D_reflected[0]], [A_reflected[1], D_reflected[1]], 'r',
label='Reflected Line Segment A\'D\'')
ax.plot([-6, 6], [-6, 6], 'k--', label='Line y = x')  # Plot the line y = x
ax.legend()
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Reflection of Line Segment AD through y = x')
plt.axis('equal')
plt.grid(True)
plt.show()
```

OUTPUT:



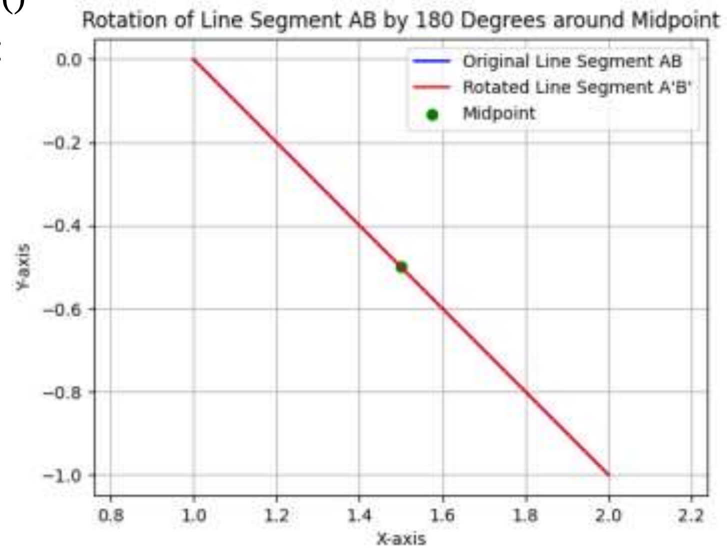
Q.4) write a python program to rotate line line segment by 180 degrees having end points (1, 0) and (2,-1).

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the end points of the line segment
A = np.array([1, 0])
```

```
B = np.array([2, -1])
# Find the midpoint of the line segment
midpoint = (A + B) / 2
# Define the rotation matrix for 180 degrees
rotation_matrix = np.array([[-1, 0],[0, -1]])
# Rotate the end points of the line segment around the midpoint
A_rotated = np.dot(rotation_matrix, A - midpoint) + midpoint
B_rotated = np.dot(rotation_matrix, B - midpoint) + midpoint
# Plot the original line segment and its rotated version
fig, ax = plt.subplots()
ax.plot([A[0], B[0]], [A[1], B[1]], 'b', label='Original Line Segment AB')
ax.plot([A_rotated[0], B_rotated[0]], [A_rotated[1], B_rotated[1]], 'r',
label='Rotated Line Segment A\'B\'')
ax.scatter(midpoint[0], midpoint[1], color='g', marker='o', label='Midpoint') #
Plot the midpoint
ax.legend()
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Rotation of Line Segment AB by 180 Degrees around Midpoint')
plt.axis('equal')
plt.grid(True)
plt.show()
```

Output:

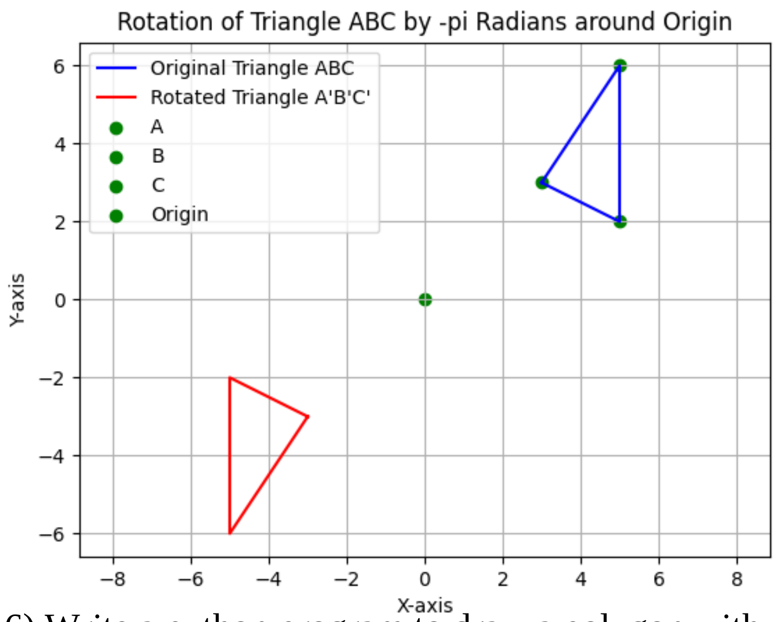


Q.5) Write a python program to plot triangle with vertices [3, 3], [5, 6], [5, 2], and its rotation about the origin by angle –pi radians.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the vertices of the triangle
A = np.array([3, 3])
B = np.array([5, 6])
C = np.array([5, 2])
# Define the rotation angle in radians
theta = -np.pi
# Define the rotation matrix for the given angle
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],[np.sin(theta), np.cos(theta)]])
# Rotate the vertices of the triangle around the origin
A_rotated = np.dot(rotation_matrix, A)
B_rotated = np.dot(rotation_matrix, B)
C_rotated = np.dot(rotation_matrix, C)
# Plot the original triangle and its rotated version
fig, ax = plt.subplots()
ax.plot([A[0], B[0], C[0], A[0]], [A[1], B[1], C[1], A[1]], 'b', label='Original Triangle ABC')
ax.plot([A_rotated[0], B_rotated[0], C_rotated[0], A_rotated[0]], [A_rotated[1], B_rotated[1], C_rotated[1], A_rotated[1]], 'r', label='Rotated Triangle A\'B\'C\'')
ax.scatter(A[0], A[1], color='g', marker='o', label='A')  # Plot vertex A
ax.scatter(B[0], B[1], color='g', marker='o', label='B')  # Plot vertex B
ax.scatter(C[0], C[1], color='g', marker='o', label='C')  # Plot vertex C
ax.scatter(0, 0, color='g', marker='o', label='Origin')  # Plot origin
ax.legend()
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Rotation of Triangle ABC by -pi Radians around Origin')
plt.axis('equal')
plt.grid(True)
plt.show()
```

OUTPUT:



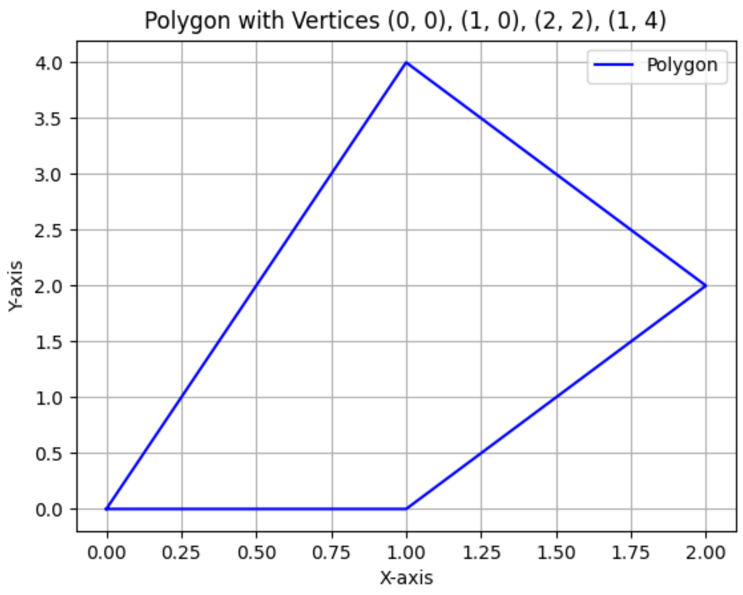
Q.6) Write a python program to draw a polygon with vertices (0, 0), (1, 0), (2, 2), (1, 4) and find its area and perimeter.

Synatx:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the vertices of the polygon
vertices = np.array([[0, 0], [1, 0], [2, 2], [1, 4], [0, 0]])
# Extract x and y coordinates of the vertices
x = vertices[:, 0]
y = vertices[:, 1]
# Plot the polygon
fig, ax = plt.subplots()
ax.plot(x, y, 'b', label='Polygon')
# Calculate the area of the polygon
area = 0.5 * np.abs(np.dot(x, np.roll(y, 1)) - np.dot(y, np.roll(x, 1)))
# Calculate the perimeter of the polygon
perimeter = np.sum(np.sqrt(np.diff(x) ** 2 + np.diff(y) ** 2))
# Print the calculated area and perimeter
print("Area of the polygon: ", area)
print("Perimeter of the polygon: ", perimeter)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Polygon with Vertices (0, 0), (1, 0), (2, 2), (1, 4)')
ax.legend()
```

```
plt.grid(True)
plt.show()
```

OUTPUT:



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

Max Z = 4x + y + 3z + 5w

Subjected to

4x + 6y - 5z - 4w >= -20

-8x - 3y + 3z + 2w <= 5 20

x > 0 , y > 0

Syntax:

```
from pulp import *
# Create the LP problem
lp_problem = LpProblem("Linear_Programming_Problem",
LpMaximize)
# Define the decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
y = LpVariable('y', lowBound=0, cat='Continuous')
z = LpVariable('z', lowBound=0, cat='Continuous')
w = LpVariable('w', lowBound=0, cat='Continuous')
# Set the objective function
lp_problem += 4*x + y + 3*z + 5*w
# Add the constraints
lp_problem += 4*x + 6*y - 5*z - 4*w >= -20
lp_problem += -8*x - 3*y + 3*z + 2*w <= 5
lp_problem += x >= 0
```

```
        lp_problem += y >= 0
        # Solve the LP problem
        lp_problem.solve()
        # Print the status of the LP problem
        print("Status: ", LpStatus[lp_problem.status])
        # Print the optimal values of the decision variables
        print("Optimal Values:")
        print("x = ", x.varValue)
        print("y = ", y.varValue)
        print("z = ", z.varValue)
        print("w = ", w.varValue)
        # Print the optimal value of the objective function
        print("Optimal Objective Function Value = ", lpSum([4*x, y, 3*z,
5*w]).getValue())
```

```
        OUTPUT:
        Status:  Unbounded
        Optimal Values:
        x =  0.83333333
        y =  0.0
        z =  0.0
        w =  5.8333333
```

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 >= 6
y >= 6
x + y <= 11
x>=0, y>=0

Syntax:

```
from pulp import *
# Create the LP problem as a minimization problem
problem = LpProblem("LPP", LpMinimize)
# Define the decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
y = LpVariable('y', lowBound=0, cat='Continuous')
# Define the objective function
problem += x + y, "Z"
```

```
# Define the constraints
problem += x >= 6, "Constraint1"
problem += y >= 6, "Constraint2"
problem += x + y <= 11, "Constraint3"
# Solve the LP problem using the simplex method
problem.solve(PULP_CBC_CMD(msg=False))
# Print the status of the solution
print("Status:", LpStatus[problem.status])
# If the problem has an optimal solution
if problem.status == LpStatusOptimal:
    # Print the optimal values of x and y
    print("Optimal x =", value(x))
    print("Optimal y =", value(y))
    # Print the optimal value of the objective function
    print("Optimal Z =", value(problem.objective))
```

OUTPUT:

```
Status: Infeasible
```

Q.9) Apply each of the following Transformation on the point P[2, -3].
(I)Refection through X-axis.
(II)Scaling in Y-coordinate by factor 1.5.
(III) Shearing in both X and Y direction by -2 and 4 units respectively.
(IV) Rotation about origin by an angle 30 degrees.

Syntax:

```
import numpy as np
# Define the original point P
P = np.array([2, -3])
# (I) Reflection through X-axis
reflection_X = np.array([[1, 0],[0, -1]])
P_reflection_X = np.dot(reflection_X, P)
# (II) Scaling in Y-coordinate by factor 1.5
scaling_Y = np.array([[1, 0],[0, 1.5]])
P_scaling_Y = np.dot(scaling_Y, P)
# (III) Shearing in both X and Y direction by -2 and 4 units respectively
shearing_XY = np.array([[1, -2],[4, 1]])
P_shearing_XY = np.dot(shearing_XY, P)
# (IV) Rotation about origin by an angle of 30 degrees
angle = np.deg2rad(30)
rotation = np.array([[np.cos(angle), -np.sin(angle)],[np.sin(angle), np.cos(angle)]])
```

```
P_rotation = np.dot(rotation, P)
# Print the results
print("Original Point P:", P)
print("Result after reflection through X-axis:", P_reflection_X)
print("Result after scaling in Y-coordinate by factor 1.5:", P_scaling_Y)
print("Result after shearing in both X and Y direction by -2 and 4 units
respectively:", P_shearing_XY)
print("Result after rotation about origin by an angle of 30 degrees:", P_rotation)
```

OUTPUT:

```
Original Point P: [ 2 -3]
Result after reflection through X-axis: [2 3]
Result after scaling in Y-coordinate by factor 1.5: [ 2.  -4.5]
Result after shearing in both X and Y direction by -2 and 4 units respectively: [8
5]
Result after rotation about origin by an angle of 30 degrees: [ 3.23205081 -
1.59807621]
```

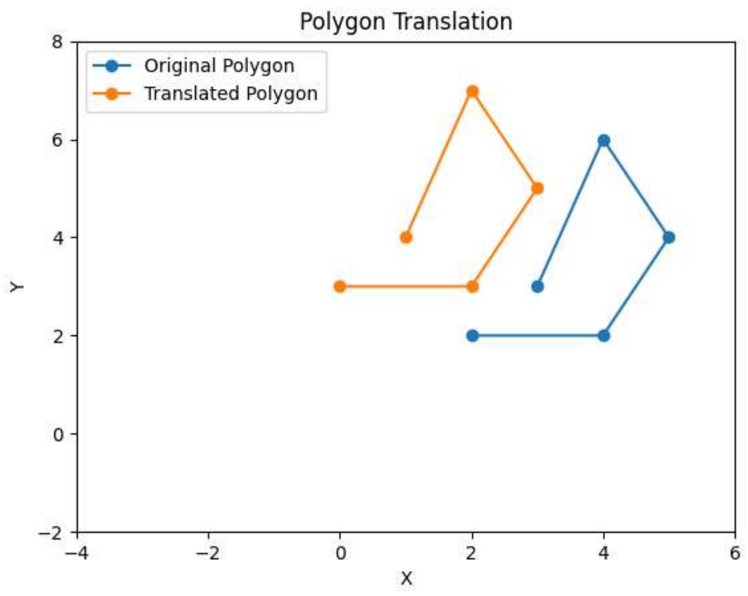
Q.10) Write a python program to draw polygon with vertices [3,3],[4,6],[5,4],[4,2] and [2,2] and its translation in x and y direction by factor -2 and 1 respectively

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the original vertices of the polygon
vertices = np.array([[3, 3],[4, 6],[5, 4],[4, 2],[2, 2]])
# Plot the original polygon
plt.plot(vertices[:, 0], vertices[:, 1], '-o', label='Original Polygon')
# Define the translation matrix
translation_matrix = np.array([[-2, 1]])
# Perform the translation on the vertices
vertices_translated = vertices + translation_matrix
# Plot the translated polygon
plt.plot(vertices_translated[:, 0], vertices_translated[:, 1], '-o', label='Translated
Polygon')
# Set plot title and labels
plt.title('Polygon Translation')
plt.xlabel('X')
plt.ylabel('Y')
# Add legend
```

```
plt.legend()
# Set plot limits
plt.xlim(-4, 6)
plt.ylim(-2, 8)
# Show the plot
plt.show()
```

OUTPUT:


Polygon Translation
Original Polygon
Translated Polygon
X
Y