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

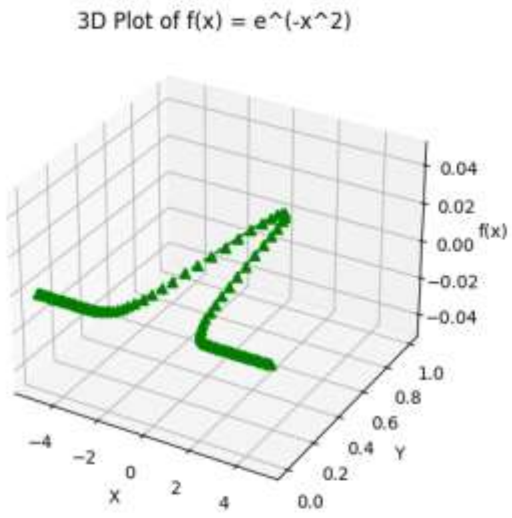
Title of the:- Practical 24 **Expt. No .** 24 **Class :-** S.Y.BCS

Q.1) Write the python program to plot 3D graph of the function f(x) = e(-x^2) in [-5,5] with green dashed points line with upward pointing triangle.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the function
def f(x):
    return np.exp(-x**2)
# Generate x values in the range [-5, 5]
x = np.linspace(-5, 5, 100)
# Calculate y values using the function
y = f(x)
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the points with green dashed lines and upward pointing triangles as markers
ax.plot(x, y, 'g--', marker='^', markersize=6)
# Set labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('f(x)')
ax.set_title('3D Plot of f(x) = e^(-x^2)')
# Show the plot
plt.show()
```

OUTPUT:

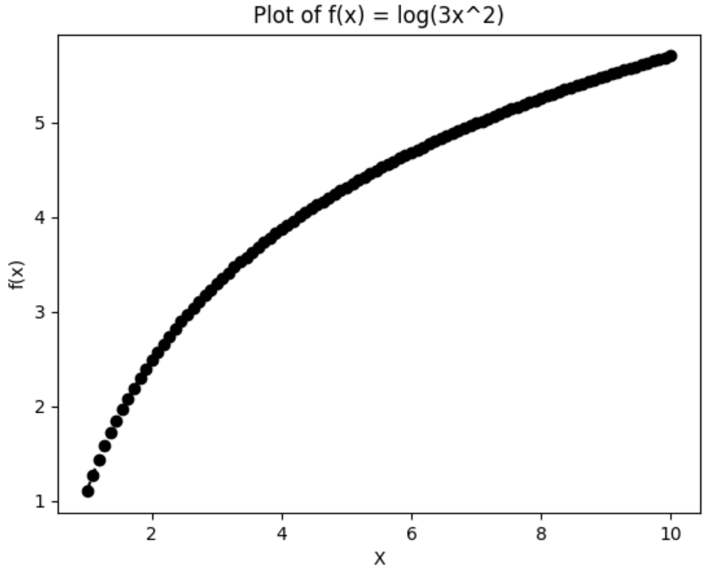


Q.2) Write the python program to plot graph of the function f(x) = log(3x^2) in [1,10] with black dashed points

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the function
def f(x):
    return np.log(3 * x**2)
# Generate x values in the range [1, 10]
x = np.linspace(1, 10, 100)
# Calculate y values using the function
y = f(x)
# Create a plot
plt.plot(x, y, 'k--', marker='o', markersize=6)
# Set labels and title
plt.xlabel('X')
plt.ylabel('f(x)')
plt.title('Plot of f(x) = log(3x^2)')
# Show the plot
plt.show()
```

OUTPUT:



Q.3) Write the python program to plot the graph of the function using def ()

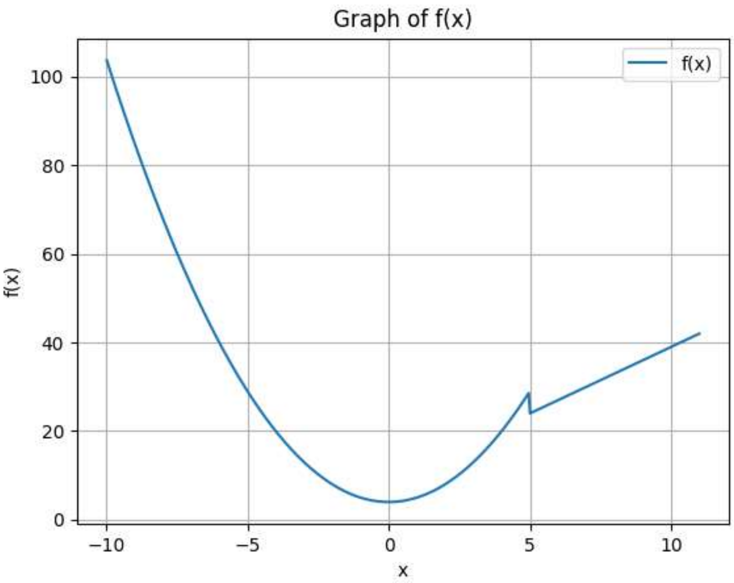
$$f(x) = \begin{cases} x^2 + 4, if & -10 < x < 5 \\ 3x + 9, if & 5 < x \geq 0 \end{cases}$$

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
def f(x):
    """Function to define f(x)."""
    if -10 < x < 5:
        return x**2 + 4
    elif 5 <= x:
        return 3*x + 9
    else:
        return None
# Generate x values
x = np.linspace(-11, 11, 500)  # Generate 500 points between -11 and 11
# Calculate y values using f(x)
y = np.array([f(xi) for xi in x])
```

```
# Create the plot
plt.plot(x, y, label='f(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graph of f(x)')
plt.legend()
plt.grid(True)
plt.show()
```

OUTPUT:



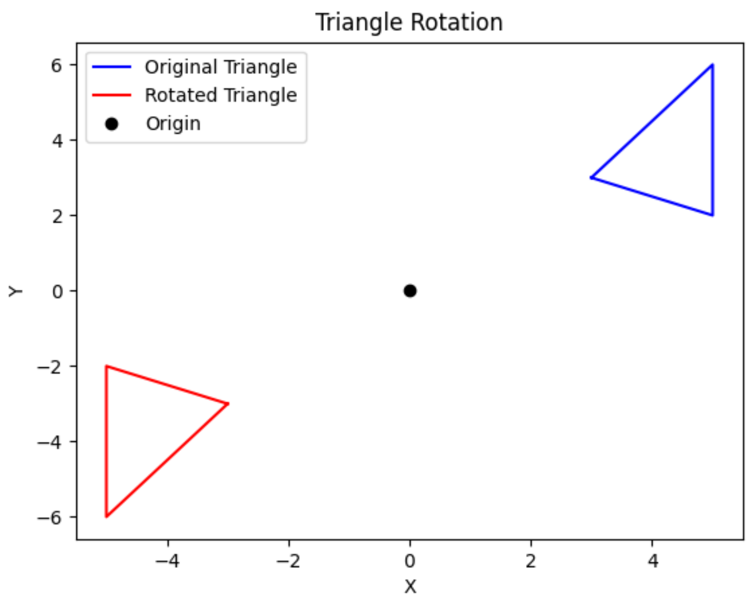
Q.4) Write the python program to plot triangle with vertices [3,3],[5,6],[5,2] and its rotation about the origin by angle –pi radians

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the vertices of the original triangle
v1 = np.array([3, 3])
v2 = np.array([5, 6])
v3 = np.array([5, 2])
# Calculate the rotation matrix
theta = -np.pi  # Angle of rotation in radians
R = np.array([[np.cos(theta), -np.sin(theta)],
```

```
              [np.sin(theta), np.cos(theta)]])
# Apply the rotation matrix to each vertex
v1_rotated = np.dot(R, v1)
v2_rotated = np.dot(R, v2)
v3_rotated = np.dot(R, v3)
# Create a plot
plt.figure()
plt.plot([v1[0], v2[0], v3[0], v1[0]], [v1[1], v2[1], v3[1], v1[1]], 'b-', label='Original Triangle')
plt.plot([v1_rotated[0], v2_rotated[0], v3_rotated[0], v1_rotated[0]], [v1_rotated[1], v2_rotated[1], v3_rotated[1], v1_rotated[1]], 'r-', label='Rotated Triangle')
plt.plot(0, 0, 'ko', label='Origin')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Triangle Rotation')
plt.legend()
# Show the plot
plt.show()
```

Output:



Q.5) Write a python to generate vector x in the interval [-22,22] using numpy package with 80 subinterval

Syntax:

```
import numpy as np
# Generate vector x with 80 subintervals
n_subintervals = 80
lower_bound = -22
upper_bound = 22
x = np.linspace(lower_bound, upper_bound, n_subintervals+1)
# Print the generated vector x
print("Vector x:", x)
```

OUTPUT:

```
Vector x: [-22.   -21.45 -20.9  -20.35 -19.8  -19.25 -18.7  -18.15 -17.6  -17.05
 -16.5  -15.95 -15.4  -14.85 -14.3  -13.75 -13.2  -12.65 -12.1  -11.55
 -11.   -10.45  -9.9   -9.35  -8.8   -8.25  -7.7   -7.15  -6.6   -6.05
  -5.5   -4.95  -4.4   -3.85  -3.3   -2.75  -2.2   -1.65  -1.1   -0.55
   0.     0.55   1.1    1.65   2.2    2.75   3.3    3.85   4.4    4.95
   5.5    6.05   6.6    7.15   7.7    8.25   8.8    9.35   9.9   10.45
  11.    11.55  12.1   12.65  13.2   13.75  14.3   14.85  15.4   15.95
  16.5   17.05  17.6   18.15  18.7   19.25  19.8   20.35  20.9   21.45
  22.  ]
```

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.

Syntax:

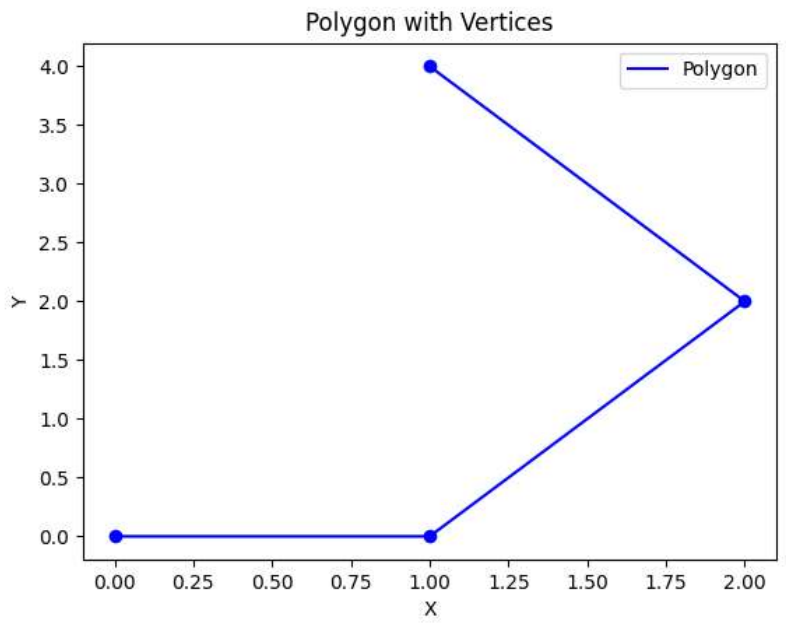
```
import numpy as np

import matplotlib.pyplot as plt

# Define the vertices of the polygon

vertices = np.array([[0, 0], [1, 0], [2, 2], [1, 4]])

# Extract x and y coordinates of the vertices

x = vertices[:, 0]

y = vertices[:, 1]

# Plot the polygon

plt.plot(x, y, 'b-', label='Polygon')

plt.plot(x, y, 'bo')
```

```
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polygon with Vertices')
plt.legend()
# Calculate the area of the polygon using shoelace formula
def calculate_area(vertices):
    x = vertices[:, 0]
    y = vertices[:, 1]
    return 0.5 * np.abs(np.dot(x, np.roll(y, 1)) - np.dot(y, np.roll(x, 1)))
area = calculate_area(vertices)
# Calculate the perimeter of the polygon
perimeter = np.sum(np.sqrt(np.sum(np.diff(vertices, axis=0)**2, axis=1)))
# Print the calculated area and perimeter
print("Area of the polygon:", area)
print("Perimeter of the polygon:", perimeter)
# Show the plot
plt.show()
```

OUTPUT:

Area of the polygon: 4.0

Perimeter of the polygon: 5.47213595499958



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

Max Z = 3.5x + 2y

Subjected to

x + y >= 5

x >=4

y<=5

x >= 0,y>= 0.

Syntax:

```
from pulp import *
# Create the LP problem
problem = LpProblem("Maximize Z", LpMaximize)
# Define the decision variables
x = LpVariable('x', lowBound=0)  # x >= 0
y = LpVariable('y', lowBound=0)  # y >= 0
# Define the objective function
problem += 3.5 * x + 2 * y
# Define the constraints
problem += x + y >= 5
problem += x >= 4
problem += y <= 5
# Solve the LP problem
status = problem.solve()
# Check the solution status
if status == 1:
    # Print the optimal solution
    print("Optimal solution:")
    print(f"x = {value(x)}")
```

```
    print(f"y = {value(y)}")
    print(f"Z = {value(problem.objective)}")
else:
    print("No feasible solution found.")
```

OUTPUT:

No feasible solution 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 => 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: Optimal

Status: Infeasible

Q.9) Write a python program lo apply the following transformation on the point = (3, -1)

(I) Reflection through X axis
(II) Reflection through the line y = x.
(III) Scaling in X Coordinate by factor 2
(IV) Scaling in Y Coordinate by factor 1.5

Sy import numpy as np

```
# Define the point
point = np.array([3, -1])
# Transformation 1: Reflection through X axis
reflection_x = np.array([[1, 0], [0, -1]])
point_reflection_x = np.dot(reflection_x, point)
print("After reflection through X axis:", point_reflection_x)
# Transformation 2: Reflection through the line y = x
reflection_yx = np.array([[0, 1], [1, 0]])
point_reflection_yx = np.dot(reflection_yx, point)
print("After reflection through the line y = x:", point_reflection_yx)
```

Transformation 3: Scaling in X Coordinate by factor 2

scaling_x = np.array([[2, 0], [0, 1]])

point_scaling_x = np.dot(scaling_x, point)

print("After scaling in X Coordinate by factor 2:", point_scaling_x)

Transformation 4: Scaling in Y Coordinate by factor 1.5

scaling_y = np.array([[1, 0], [0, 1.5]])

point_scaling_y = np.dot(scaling_y, point)

print("After scaling in Y Coordinate by factor 1.5:", point_scaling_y)ntax:

OUTPUT:
After reflection through X axis: [3 1]
After reflection through the line y = x: [-1 3]
After scaling in X Coordinate by factor 2: [6 -1]
After scaling in Y Coordinate by factor 1.5: [3. -1.5]

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

(I) Reflection Through the line y = x
(II) Scaling in X-Coordinate by factor 3
(III) Shearing in Y – Direction by 4.5 unit
(IV) Rotation about origin by an angle pi.

Syntax:
import numpy as np
# Define the points A and B
A = np.array([4, -1])
B = np.array([3, 0])
# Transformation 1: Reflection through the line y = x
reflection_yx = np.array([[0, 1], [1, 0]])
A_reflection_yx = np.dot(reflection_yx, A)
B_reflection_yx = np.dot(reflection_yx, B)
# Transformation 2: Scaling in X-Coordinate by factor 3
scaling_x = np.array([[3, 0], [0, 1]])
A_scaling_x = np.dot(scaling_x, A_reflection_yx)
B_scaling_x = np.dot(scaling_x, B_reflection_yx)
# Transformation 3: Shearing in Y-Direction by 4.5 units

```
shearing_y = np.array([[1, 0], [0, 1]])
shearing_y[0, 1] = 4.5
A_shearing_y = np.dot(shearing_y, A_scaling_x)
B_shearing_y = np.dot(shearing_y, B_scaling_x)
# Transformation 4: Rotation about origin by an angle pi
rotation_pi = np.array([[-1, 0], [0, -1]])
A_rotation_pi = np.dot(rotation_pi, A_shearing_y)
B_rotation_pi = np.dot(rotation_pi, B_shearing_y)
# Print the transformed points
print("After Reflection through the line y = x:")
print("A:", A_reflection_yx)
print("B:", B_reflection_yx)
print("\nAfter Scaling in X-Coordinate by factor 3:")
print("A:", A_scaling_x)
print("B:", B_scaling_x)
print("\nAfter Shearing in Y-Direction by 4.5 units:")
print("A:", A_shearing_y)
print("B:", B_shearing_y)
print("\nAfter Rotation about origin by an angle pi:")
print("A:", A_rotation_pi)
print("B:", B_rotation_pi)
```

OUTPUT:

```
After Reflection through the line y = x:
A: [-1  4]
B: [0 3]
After Scaling in X-Coordinate by factor 3:
A: [-3  4]
B: [0 3]
After Shearing in Y-Direction by 4.5 units:
A: [13  4]
B: [12  3]
After Rotation about origin by an angle pi:
A: [-13  -4]
B: [-12  -3]
```