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:- 04** **Date:-** / /2023

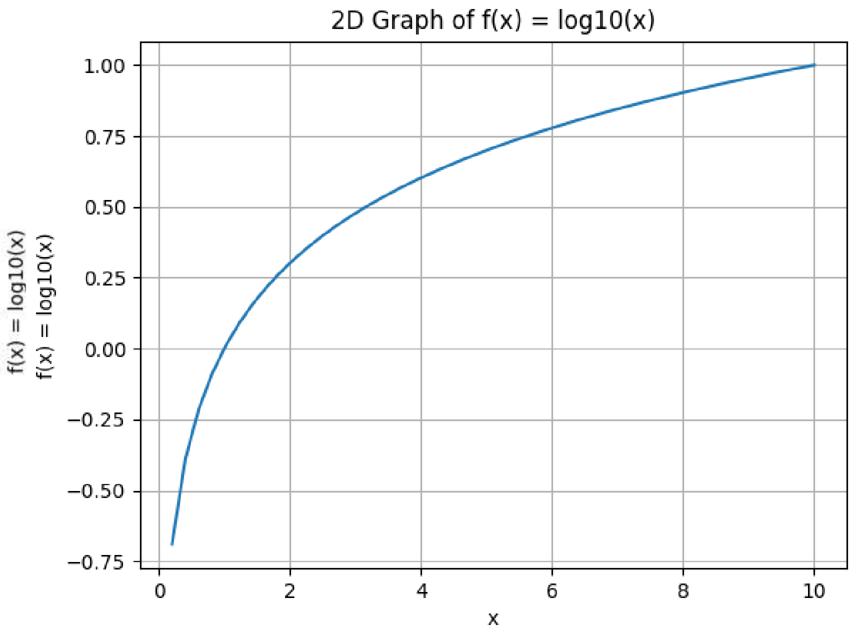
Title of the:- Practical 4 **Expt. No .** 4 **Class :-** S.Y.BCS

Q.1) Write a Python program to plot graph of the functions f(x) = log10(x) in [0,10]

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,10)
y = np.log10(x)
# Plot the graph
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x) = log10(x)')
plt.title('2D Graph of f(x) = log10(x)')
plt.grid(True)
plt.show()
```

OUTPUT:

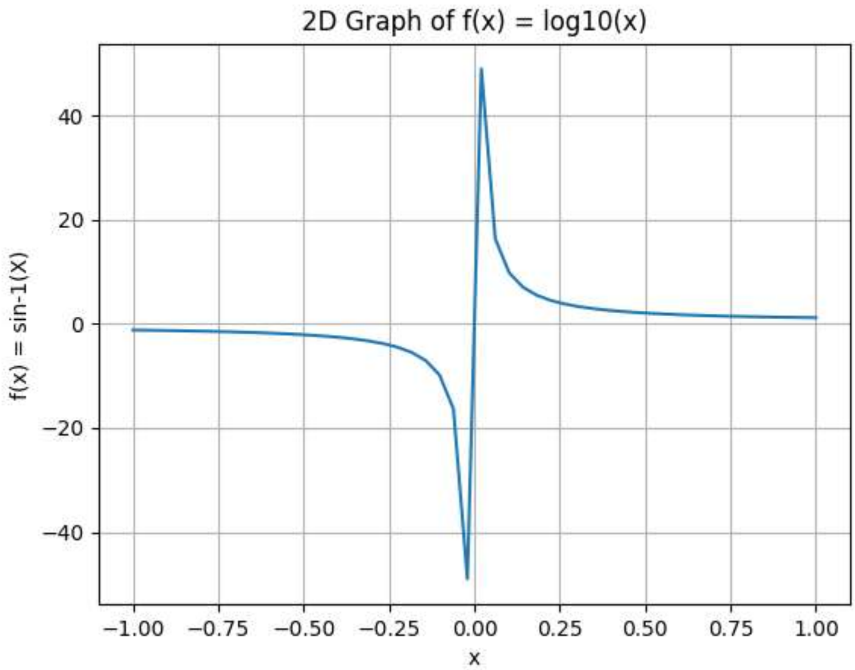


Q.2) Write a Python program to plot graph of the functions f(x) = $\sin^{-1}(x)$ in [-1,1]

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-1,1)
y = 1/np.sin(x)
# Plot the graph
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x) = sin-1(X)')
plt.title('2D Graph of f(x) = log10(x)')
plt.grid(True)
plt.show()
```

OUTPUT:

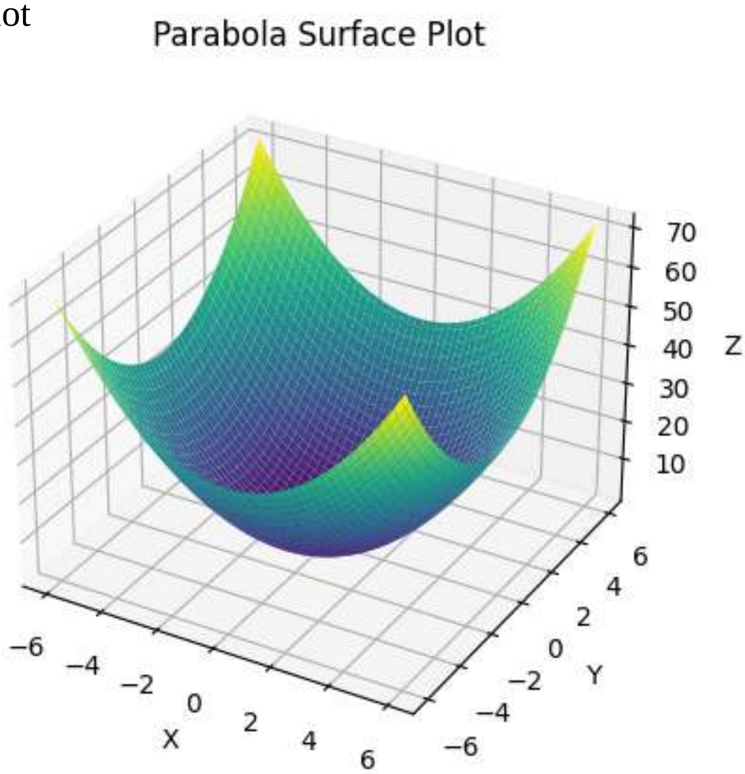


Q.3) Using Python plot the surface plot of parabola z = x**2 + y**2 in -6 <x,y<6

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate data for x, y, and z
x = np.linspace(-6, 6, 100)  # x values from -6 to 6
y = np.linspace(-6, 6, 100)  # y values from -6 to 6
X, Y = np.meshgrid(x, y)     # Create a meshgrid of x and y values
Z = X**2 + Y**2              # Calculate z values using the parabola equation
# Create a 3D 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')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Parabola Surface Plot')
# Show the plot
plt.show()
```

OUTPUT:



Q.4) If the line with points A[3, 1], B[5, -1] is transformed by the transformation matrix [T] = $\begin{matrix} 3 & -2 \\ 2 & 1 \end{matrix}$ then using python, find the equation of transformed line.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define original points A and B
A = np.array([3, 1])
B = np.array([5, -1])
# Define transformation matrix [T]
T = np.array([[3, -2],
              [2, 1]])
# Apply transformation matrix to points A and B
A_transformed = np.dot(T, A)
B_transformed = np.dot(T, B)
# Extract transformed coordinates for points A and B
A_transformed_x = A_transformed[0]
A_transformed_y = A_transformed[1]
B_transformed_x = B_transformed[0]
B_transformed_y = B_transformed[1]
# Calculate slope and y-intercept of the transformed line
m = (B_transformed_y - A_transformed_y) / (B_transformed_x - A_transformed_x)
b = A_transformed_y - m * A_transformed_x
# Print equation of the transformed line
print(f"The equation of the transformed line is: y = {m:.2f}x + {b:.2f}")
```

Output:

The equation of the transformed line is: y = 0.20x + 5.60

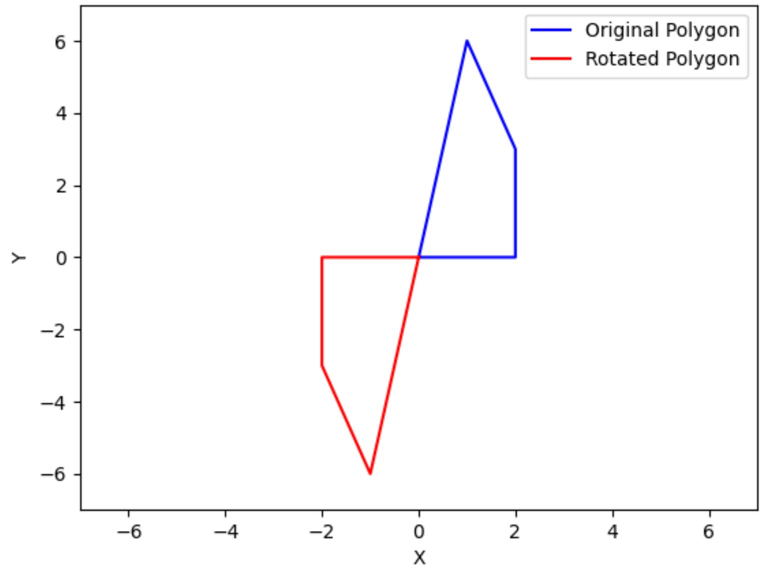
Q.5) Write a Python program to draw a polygon with vertices (0,0), (2,0), (2,3) and (1,6) and rotate by 180^0

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define vertices of the polygon
```

```
vertices = np.array([[0, 0],
                     [2, 0],
                     [2, 3],
                     [1, 6],
                     [0, 0]])  # Closing the polygon by repeating the first vertex
# Plot the original polygon
plt.plot(vertices[:, 0], vertices[:, 1], 'b-', label='Original Polygon')
# Define rotation matrix for 180 degrees (in radians)
angle_rad = np.deg2rad(180)
rotation_matrix = np.array([[np.cos(angle_rad), -np.sin(angle_rad)],
                            [np.sin(angle_rad), np.cos(angle_rad)]])
# Apply rotation matrix to vertices
vertices_rotated = np.dot(rotation_matrix, vertices.T).T
# Plot the rotated polygon
plt.plot(vertices_rotated[:, 0], vertices_rotated[:, 1], 'r-', label='Rotated Polygon')
# Set axis limits and labels
plt.xlim(-7, 7)
plt.ylim(-7, 7)
plt.xlabel('X')
plt.ylabel('Y')
# Add a legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:

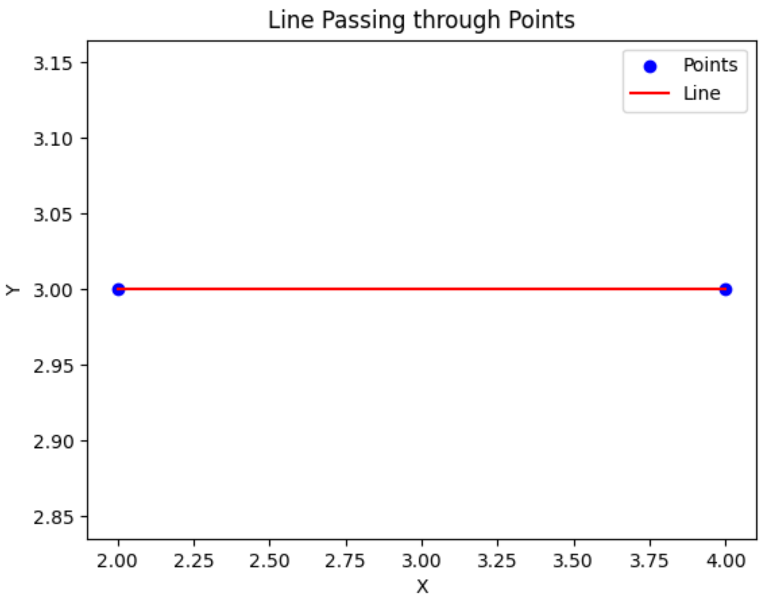


Q.6) Using python, generate line passing through points (2,3) and (4,3) and equation of the line

Synatx:

```
import matplotlib.pyplot as plt

import numpy as np

# Define the points

x = np.array([2, 4])

y = np.array([3, 3])

# Calculate the slope (m) and y-intercept (b) of the line

m = (y[1] - y[0]) / (x[1] - x[0])

b = y[0] - m * x[0]

# Print the equation of the line

print(f"The equation of the line is: y = {m:.2f}x + {b:.2f}")

# Plot the points and the line

plt.scatter(x, y, c='blue', label='Points')

plt.plot(x, m * x + b, c='red', label='Line')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Line Passing through Points')

plt.legend()

plt.show()
```

OUTPUT:



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

Max Z = 150x + 75y

Subjected to

4x + 6y <= 24

5x + 3y <= 15

x > 0 , y > 0

Syntax:

```
from pulp import *
# Create the LP problem as a maximization problem
problem = LpProblem("LPP", LpMaximize)
# Define the decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
y = LpVariable('y', lowBound=0, cat='Continuous')
# Define the objective function
problem += 150 * x + 75 * y, "Z"
# Define the constraints
problem += 4 * x + 6 * y <= 24, "Constraint1"
problem += 5 * x + 3 * y <= 15, "Constraint2"
# Solve the LP problem
problem.solve()
# Print the status of the solution
print("Status:", LpStatus[problem.status])
# 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

Optimal x = 3.0

Optimal y = 0.0

Optimal Z = 450.0

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 = 4x+y+3z+5w
subject to
4x+-6y-4w >= -20
-8x-3y+3z+2w <= 20
x + y <= 11
x >= 0,y>= 0,z>= 0,w>= 0

Syntax:

```
#By using Pulp Method
from pulp import LpMinimize, LpProblem, LpStatus, lpSum, LpVariable, PULP_CBC_CMD
# Create LP problem
problem = LpProblem("LPP", LpMinimize)
# Define decision variables
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
z = LpVariable('z', lowBound=0)
w = LpVariable('w', lowBound=0)
# Objective function
problem += 4 * x + y + 3 * z + 5 * w, "Z"
# Constraints
problem += 4 * x - 6 * y - 4 * w >= -20, "constraint1"
problem += -8 * x - 3 * y + 3 * z + 2 * w <= 20, "constraint2"
problem += x + y <= 11, "constraint3"
# Solve LP problem using simplex method
problem.solve(PULP_CBC_CMD())
# Print status of the solution
print("Status: ", LpStatus[problem.status])
# Print optimal solution if exists
if problem.status == 1:  # LpStatusOptimal
    print("Optimal Solution:")
```

```
    print("x = ", x.value())
    print("y = ", y.value())
    print("z = ", z.value())
    print("w = ", w.value())
    print("Z = ", problem.objective.value())
else:
    print("No optimal solution exists.")
```

OUTPUT:

```
Status:  Optimal
Optimal Solution:
x =  0.0
y =  0.0
z =  0.0
w =  0.0
Z =  0.0
```

```
#by using Simplex Method
import numpy as np
from scipy.optimize import linprog
# Define the coefficients of the objective function
c = np.array([4, 1, 3, 5])
# Define the coefficients of the inequality constraints
A = np.array([[4, -6, 0, -4],
              [-8, -3, 3, 2],
              [1, 1, 0, 0]])
# Define the right-hand side of the inequality constraints
b = np.array([-20, 20, 11])
# Define the bounds on the decision variables
bounds = [(0, None), (0, None), (0, None), (0, None)]
# Solve the LP problem using the simplex method
result = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='simplex')
# Print the optimal solution if exists
if result.success:
    print("Optimal Solution:")
    print("x = ", result.x[0])
    print("y = ", result.x[1])
    print("z = ", result.x[2])
    print("w = ", result.x[3])
    print("Z = ", result.fun)
```

```
else:
    print("No optimal solution exists.")
```

OUTPUT:

```
x =  0.0
y =  3.3333333333333334
z =  0.0
w =  0.0
Z =  3.3333333333333334
```

Q.9) Plot 3D axes with labels X - axis and z –axis and also plot following points with given coordinate in one graph

(I) (70, -25, 15) as a diamond in black color,

(II) (50, 72, -45) as a* in green color,

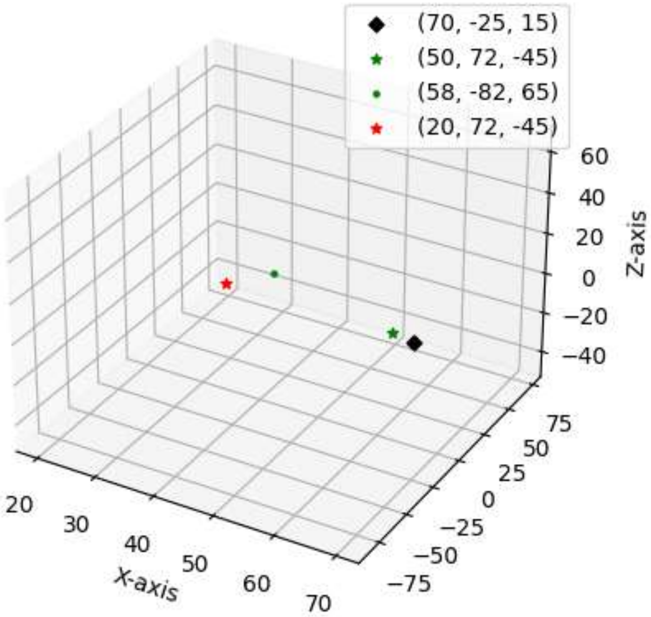
(III) (58, -82, 65) as a dot in green color,

(IV) (20, 72, -45) as a * in Red color.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Create a 3D figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the 3D axes with labels
ax.set_xlabel('X-axis')
ax.set_zlabel('Z-axis')
# Define the points and their coordinates
points = {'(70, -25, 15)': (70, -25, 15),
          '(50, 72, -45)': (50, 72, -45),
          '(58, -82, 65)': (58, -82, 65),
          '(20, 72, -45)': (20, 72, -45)}
# Plot each point with the specified marker and color
for label, (x, y, z) in points.items():
    if label == '(70, -25, 15)':
        ax.scatter(x, y, z, marker='D', color='black', label=label)
    elif label == '(50, 72, -45)':
        ax.scatter(x, y, z, marker='*', color='green', label=label)
    elif label == '(58, -82, 65)':
        ax.scatter(x, y, z, marker='.', color='green', label=label)
    elif label == '(20, 72, -45)':
        ax.scatter(x, y, z, marker='*', color='red', label=label)
```

# Add a legend to the graph
ax.legend()
# Show the 3D graph
plt.show()
OUTPUT:



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

(I) Shearing in X – Direction by 9 unit
(II) Rotation about origin through an angle pi.
(III) Scaling in X-Coordinate by 2 units.
(IV) Reflection trough he line y = x

Syntax:

```
import numpy as np
# Input points A and B
A = np.array([4, -1])
B = np.array([3, 0])
# Transformation 1: Shearing in X-Direction by 9 units
shear_matrix = np.array([[1, 9],
                         [0, 1]])
A_sheared = np.dot(shear_matrix, A)
B_sheared = np.dot(shear_matrix, B)
print("Transformed Point A after Shearing:", A_sheared)
```

```
print("Transformed Point B after Shearing:", B_sheared)
# Transformation 2: Rotation about origin through an angle of pi (180 degrees)
rotation_matrix = np.array([[np.cos(np.pi), -np.sin(np.pi)],
                            [np.sin(np.pi), np.cos(np.pi)]])
A_rotated = np.dot(rotation_matrix, A_sheared)
B_rotated = np.dot(rotation_matrix, B_sheared)
print("Transformed Point A after Rotation:", A_rotated)
print("Transformed Point B after Rotation:", B_rotated)
# Transformation 3: Scaling in X-Coordinate by 2 units
scaling_matrix = np.array([[2, 0],
                           [0, 1]])
A_scaled = np.dot(scaling_matrix, A_rotated)
B_scaled = np.dot(scaling_matrix, B_rotated)
print("Transformed Point A after Scaling:", A_scaled)
print("Transformed Point B after Scaling:", B_scaled)
# Transformation 4: Reflection through the line y = x
reflection_matrix = np.array([[0, 1],
                              [1, 0]])
A_reflected = np.dot(reflection_matrix, A_scaled)
B_reflected = np.dot(reflection_matrix, B_scaled)
print("Transformed Point A after Reflection:", A_reflected)
print("Transformed Point B after Reflection:", B_reflected)
```

OUTPUT:

```
Transformed Point A after Shearing: [-5 -1]
Transformed Point B after Shearing: [3 0]
Transformed Point A after Rotation: [5. 1.]
Transformed Point B after Rotation: [-3.0000000e+00  3.6739404e-16]
Transformed Point A after Scaling: [10.  1.]
Transformed Point B after Scaling: [-6.0000000e+00  3.6739404e-16]
Transformed Point A after Reflection: [ 1. 10.]
Transformed Point B after Reflection: [ 3.6739404e-16 -6.0000000e+00]
```