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 6 **Expt. No .** 6 **Class :-** S.Y.BCS

Q.1) Using python, generate 3D surface Plot for the function f (x) = sin (x**2 + y**2) in the interval [0, 10].

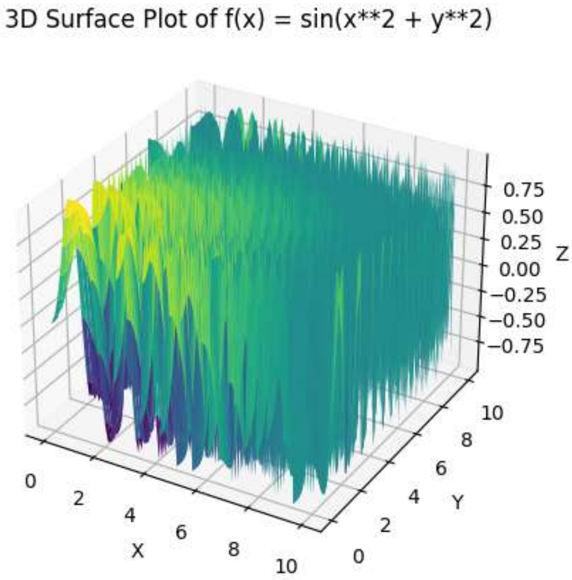
Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the function
def f(x, y):
    return np.sin(x**2 + y**2)
# Generate x, y values
x = np.linspace(0, 10, 100)
y = np.linspace(0, 10, 100)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
# Create a 3D figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the surface
ax.plot_surface(X, Y, Z, cmap='viridis')
# Add labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('3D Surface Plot of f(x) = sin(x**2 + y**2)')
```

Show the plot

plt.show()

OUTPUT:



Q.2) Using Python, plot the graph of function f(x) = sin(x) – e**x + 3*x**2 - log 10(x) on the Interval [0, 2*pi]

Syntax:

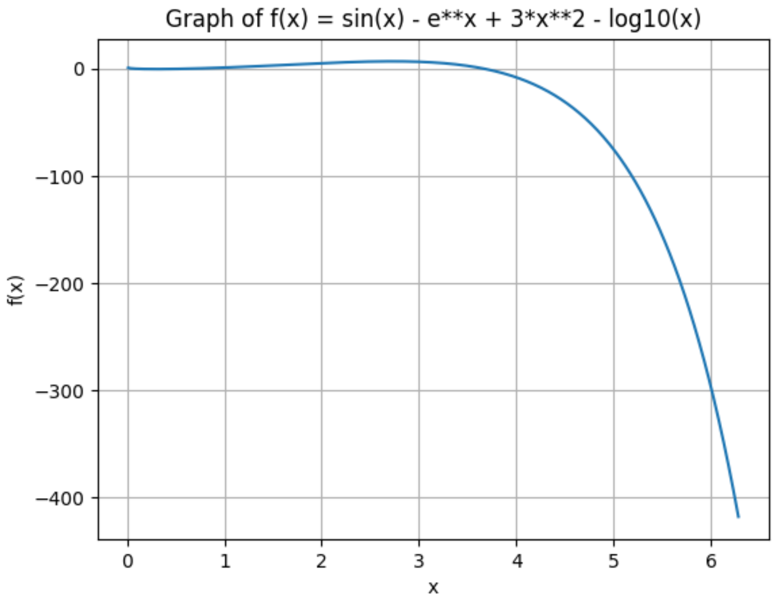
```
import numpy as np
import matplotlib.pyplot as plt
# Define the function
def f(x):
    return np.sin(x) - np.exp(x) + 3 * x**2 - np.log10(x)
# Generate x values
x = np.linspace(0, 2*np.pi, 500)  # 500 points between 0 and 2*pi
y = f(x)  # Evaluate f(x) for each x value
# Create a plot
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graph of f(x) = sin(x) - e**x + 3*x**2 - log10(x)')
```

plt.grid(True)

Show the plot

plt.show()

OUTPUT:



Q.3) Draw the horizontal bar graph for the following data in Maroorn.

City	Pune	Mumbai	Nasik	Nagpur	Thane
Air Quality Index	168	190	170	178	195

Syntax:

import matplotlib.pyplot as plt

Data

left = [1, 2, 3, 4, 5]

height = [168, 190, 170, 178, 195]

tick_label = ['Pune', 'Mumbai', 'Nasik', 'Nagpur', 'Thane']

Create a horizontal bar graph

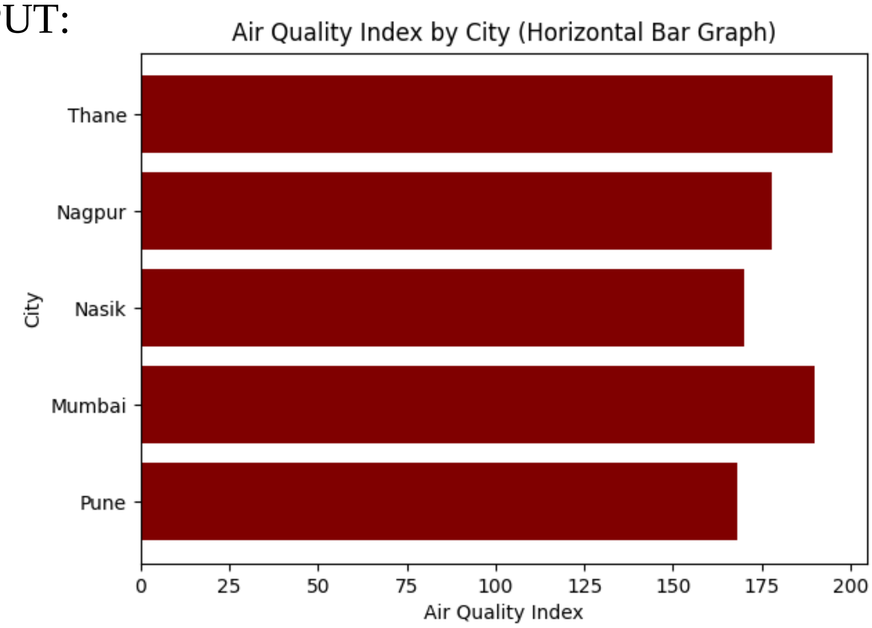
plt.barh(left, height, tick_label=tick_label, color='maroon')

Set labels and title

plt.xlabel('Air Quality Index')

```
plt.ylabel('City')
plt.title('Air Quality Index by City (Horizontal Bar Graph)')
# Show the plot
plt.show()
```

OUTPUT:



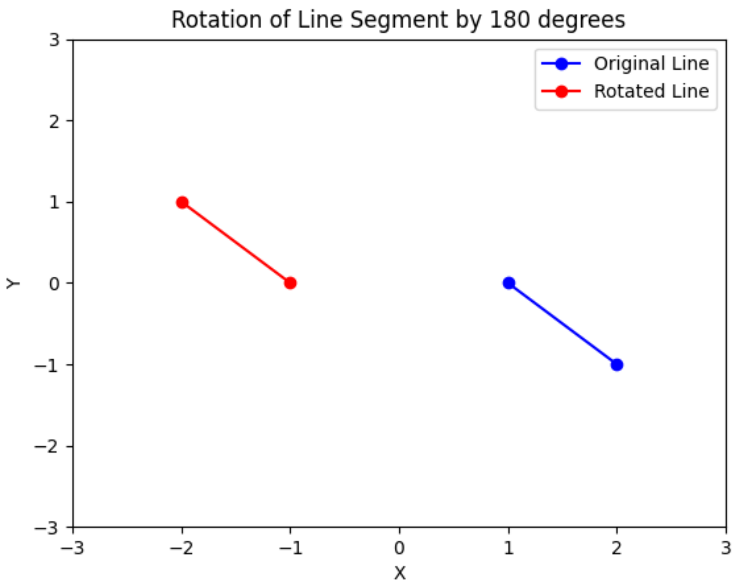
Q.4) Using python, rotate the line segment by 180° having end points (1, 0) and (2, -1)

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the original line segment
x1, y1 = 1, 0
x2, y2 = 2, -1
# Plot the original line segment
plt.plot([x1, x2], [y1, y2], 'bo-', label='Original Line')
# Compute the rotated coordinates
x1_rot, y1_rot = -x1, -y1
x2_rot, y2_rot = -x2, -y2
# Plot the rotated line segment
plt.plot([x1_rot, x2_rot], [y1_rot, y2_rot], 'ro-', label='Rotated Line')
```

```
# Set axis limits
plt.xlim(-3, 3)
plt.ylim(-3, 3)
# Add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Rotation of Line Segment by 180 degrees')
# Add legend
plt.legend()
# Show the plot
plt.show()
```

Output:



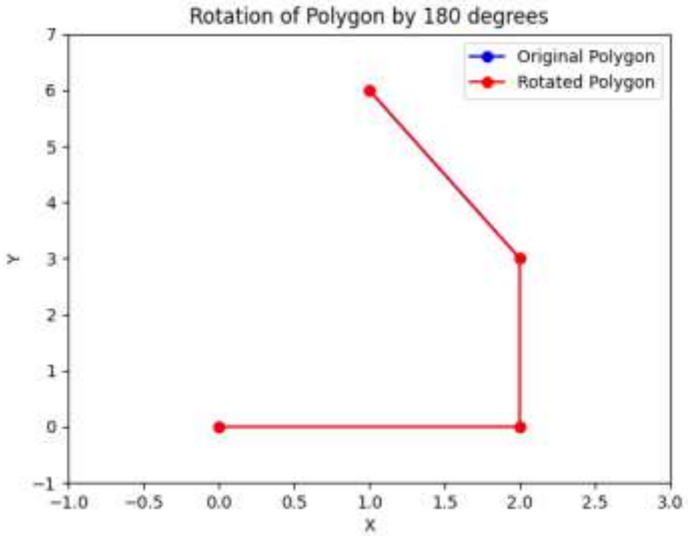
Q.5) Write a Python program to draw a polygon with vertices (0, 0), (2, 0), (2, 3) and (1, 6) and rotate it by 180°.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the original polygon vertices
vertices = np.array([[0, 0], [2, 0], [2, 3], [1, 6]])
# Plot the original polygon
plt.plot(vertices[:, 0], vertices[:, 1], 'bo-', label='Original Polygon')
# Compute the rotated coordinates
```

```
vertices_rot = np.flip(vertices, axis=0)
# Plot the rotated polygon
plt.plot(vertices_rot[:, 0], vertices_rot[:, 1], 'ro-', label='Rotated Polygon')
# Set axis limits
plt.xlim(-1, 3)
plt.ylim(-1, 7)
# Add labels and title
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Rotation of Polygon by 180 degrees')
# Add legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:



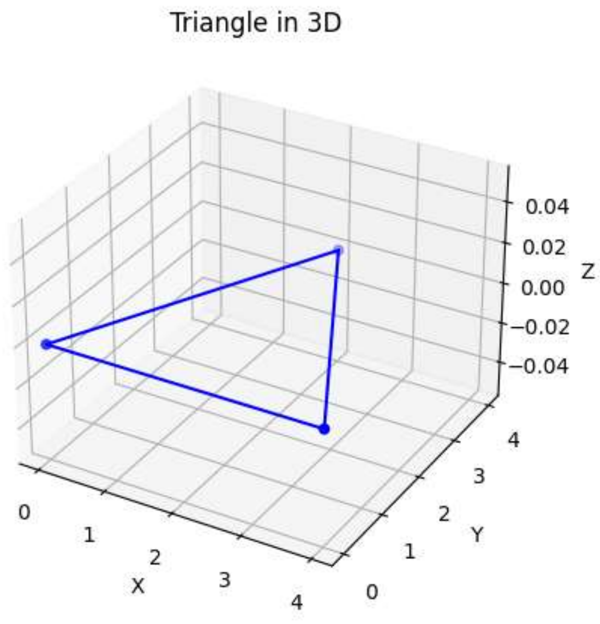
Q.6) Using python, generate triangle with vertices (0,0),(4,0),(2,4), check whether the triangle is isosceles triangle..

Synatx:

```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

# Define the triangle vertices

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

```
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the triangle vertices
ax.scatter(vertices[:, 0], vertices[:, 1], vertices[:, 2], c='blue', marker='o')
# Plot the triangle edges
for i in range(3):
    ax.plot([vertices[i, 0], vertices[(i+1)%3, 0]],
            [vertices[i, 1], vertices[(i+1)%3, 1]],
            [vertices[i, 2], vertices[(i+1)%3, 2]], c='blue')
# Check if any two sides are equal
d1 = np.linalg.norm(vertices[0, :2] - vertices[1, :2])
d2 = np.linalg.norm(vertices[0, :2] - vertices[2, :2])
d3 = np.linalg.norm(vertices[1, :2] - vertices[2, :2])
if d1 == d2 or d1 == d3 or d2 == d3:
    print("The triangle is an isosceles triangle.")
else:
    print("The triangle is not an isosceles triangle.")
# Set plot labels and title
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Triangle in 3D')
# Show the plot
plt.show()
```

OUTPUT:



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

Max Z = x + y

Subjected to

2x - 2y => 1

x + y >= 2

x > 0 , y > 0

Syntax:

from pulp import LpProblem, LpMaximize, LpVariable, LpStatus, lpSum, value

Create a maximization problem

prob = LpProblem("MaximizationProblem", LpMaximize)

Define variables

x = LpVariable('x', lowBound=0)

y = LpVariable('y', lowBound=0)

Define objective function

prob += x + y

Define constraints

```
prob += 2*x - 2*y >= 1
prob += x + y >= 2
# Solve the problem
prob.solve()
# Get the status of the solution
status = LpStatus[prob.status]
# Print the status of the solution
print("Status: {}".format(status))
# If the problem is solved successfully, print the optimal values of x and y
if status == 'Optimal':
    print("Optimal Solution:")
    print("x = {}".format(value(x)))
    print("y = {}".format(value(y)))
    print("Objective Function (Z) = {}".format(value(prob.objective)))
```

OUTPUT:

Status: Unbounded

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:

```
#By using Pulp Method
from pulp import LpProblem, LpMinimize, LpVariable, LpStatus, lpSum, value
# Create a minimization problem
prob = LpProblem("MinimizationProblem", LpMinimize)
```

```
# Define variables
x = LpVariable('x', lowBound=0)
y = LpVariable('y', lowBound=0)
# Define objective function
prob += x + y
# Define constraints
prob += x >= 6
prob += y >= 6
prob += x + y <= 11
# Solve the problem
prob.solve()
# Get the status of the solution
status = LpStatus[prob.status]
# Print the status of the solution
print("Status: {}".format(status))
# If the problem is solved successfully, print the optimal values of x and y
if status == 'Optimal':
    print("Optimal Solution:")
    print("x = {}".format(value(x)))
    print("y = {}".format(value(y)))
    print("Objective Function (Z) = {}".format(value(prob.objective)))
```

OUTPUT:

```
Status: Infeasible
```

```
from scipy.optimize import linprog
# Define the coefficients of the objective function
c = [1, 1]
# Define the coefficient matrix of the inequality constraints
A_ub = [[-1, 0], [0, -1], [-1, -1]]
# Define the right-hand side of the inequality constraints
b_ub = [-6, -6, -11]
# Define the bounds on the variables
bounds = [(6, None), (6, None)]
# Solve the linear programming problem using the simplex method
result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='simplex')
# Print the result
print("Status: {}".format(result.message))
if result.success:
    print("Optimal Solution:")
```

```
    print("x = {}".format(result.x[0]))
    print("y = {}".format(result.x[1]))
    print("Objective Function (Z) = {}".format(result.fun))
```
OUTPUT:
Status: Optimization terminated successfully.
Optimal Solution:
x = 6.0
y = 6.0
Objective Function (Z) = 12.0

Q.9) Apply Python. Program in each of the following transformation on the point P[4,-2]
(I)Refection through y-axis.
(II)Scaling in X-co-ordinate by factor 7.
(III) Shearing in Y Direction by 3 unit.
(IV) Reflection through the line y = - x
Syntax:
```
# Point P
P = [4, -2]
print("Original Point P: {}".format(P))
# Reflection through y-axis
P_reflect_y_axis = [-P[0], P[1]]
print("Reflection through y-axis: {}".format(P_reflect_y_axis))
# Scaling in X-coordinates by factor 7
scaling_factor_x = 7
P_scaling_x = [scaling_factor_x * P[0], P[1]]
print("Scaling in X-coordinates by factor 7: {}".format(P_scaling_x))
# Shearing in Y-direction by 3 units
shearing_factor_y = 3
P_shearing_y = [P[0], P[0] + shearing_factor_y * P[1]]
print("Shearing in Y-direction by 3 units: {}".format(P_shearing_y))
# Reflection through the line y = -x
P_reflect_line = [-P[1], -P[0]]
print("Reflection through the line y = -x: {}".format(P_reflect_line))
```
OUTPUT:
Original Point P: [4, -2]
Reflection through y-axis: [-4, -2]
Scaling in X-coordinates by factor 7: [28, -2]
Shearing in Y-direction by 3 units: [4, -10]
Reflection through the line y = -x: [2, -4]

Q.10) Find the combined transformation by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle 60^o.

(II) Scaling in X-Coordinate by 7 units.

(III) Uniform Scaling by 4 unit

(IV) Reflection through the line y = x

Syntax:

```
# Point P
P = [2, 3]
print("Original Point P: {}".format(P))
# Transformation 1: Rotation about origin through an angle of 60 degrees
import math
angle = 60
angle_rad = math.radians(angle)
P_rotation = [round(P[0] * math.cos(angle_rad) - P[1] * math.sin(angle_rad)), round(P[0] * math.sin(angle_rad) + P[1] * math.cos(angle_rad))]
print("Transformation 1: Rotation about origin through an angle of 60 degrees: {}".format(P_rotation))
# Transformation 2: Scaling in X-Coordinate by 7 units
scaling_factor_x = 7
P_scaling_x = [scaling_factor_x * P_rotation[0], P_rotation[1]]
print("Transformation 2: Scaling in X-Coordinate by 7 units: {}".format(P_scaling_x))
# Transformation 3: Uniform Scaling by 4 units
scaling_factor_uniform = 4
P_scaling_uniform = [scaling_factor_uniform * P_scaling_x[0], scaling_factor_uniform * P_scaling_x[1]]
print("Transformation 3: Uniform Scaling by 4 units: {}".format(P_scaling_uniform))
# Transformation 4: Reflection through the line y = x
P_reflect_line = [P_scaling_uniform[1], P_scaling_uniform[0]]
print("Transformation 4: Reflection through the line y = x: {}".format(P_reflect_line))
```

OUTPUT:

```
Original Point P: [2, 3]
Transformation 1: Rotation about origin through an angle of 60 degrees: [1, 3]
Transformation 2: Scaling in X-Coordinate by 7 units: [7, 3]
Transformation 3: Uniform Scaling by 4 units: [28, 12]
Transformation 4: Reflection through the line y = x: [12, 28]
```