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 10 **Expt. No .** 10 **Class :-** S.Y.BCS

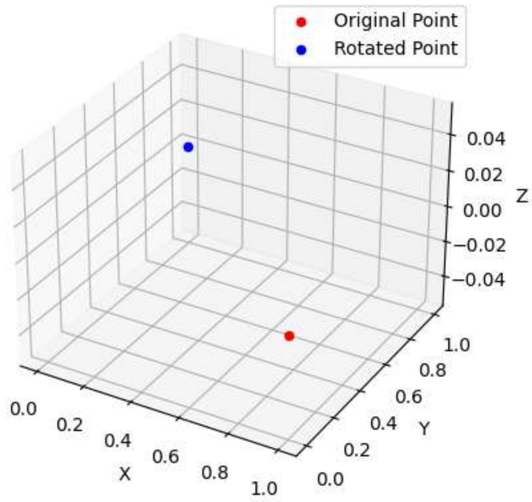
Q.1) Write a python in 3D to rotate the point (1,0,0) through XY plane in Clockwise direction (Rotation Through Z – Axis by an angle of 90^0)

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Define the original point
point = np.array([1, 0, 0])
# Define the rotation matrix for rotation through Z-axis by 90 degrees (clockwise)
angle = np.radians(90)
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle), 0],
                            [np.sin(angle), np.cos(angle), 0],
                            [0, 0, 1]])
# Apply the rotation to the point
point_rotated = np.dot(rotation_matrix, point)
# Create a 3D plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the original point
ax.scatter(point[0], point[1], point[2], color='red', label='Original Point')
# Plot the rotated point
ax.scatter(point_rotated[0], point_rotated[1], point_rotated[2], color='blue',
label='Rotated Point')
# Set plot labels and legend
```

```
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.legend()
# Show the plot
plt.show()
```

OUTPUT:



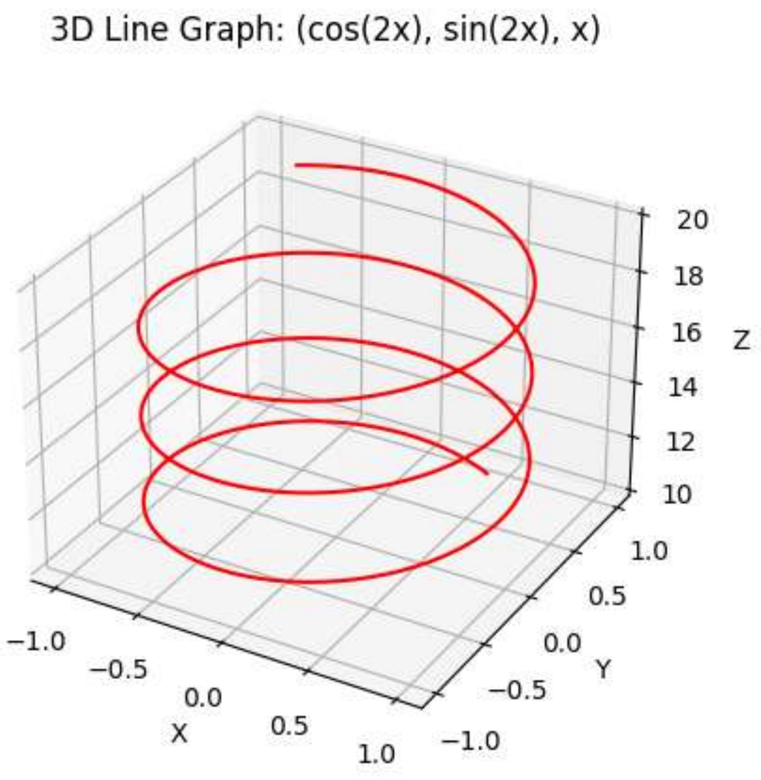
Q.2) Write a Python program to plot 3D line graph Whose parametric equation is (cos(2x),sin(2x),x) for 10 <= x <= 20 (in red color), with title of the graph

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate values for x
x = np.linspace(10, 20, 500)
# Calculate parametric equations for x, y, z
y = np.sin(2 * x)
z = x
x = np.cos(2 * x)
# Create a 3D figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot the 3D line graph
```

```
ax.plot(x, y, z, color='red')
# Set title for the graph
ax.set_title("3D Line Graph: (cos(2x), sin(2x), x)")
# Set labels for x, y, z axes
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
# Show the plot
plt.show()
```

OUTPUT:



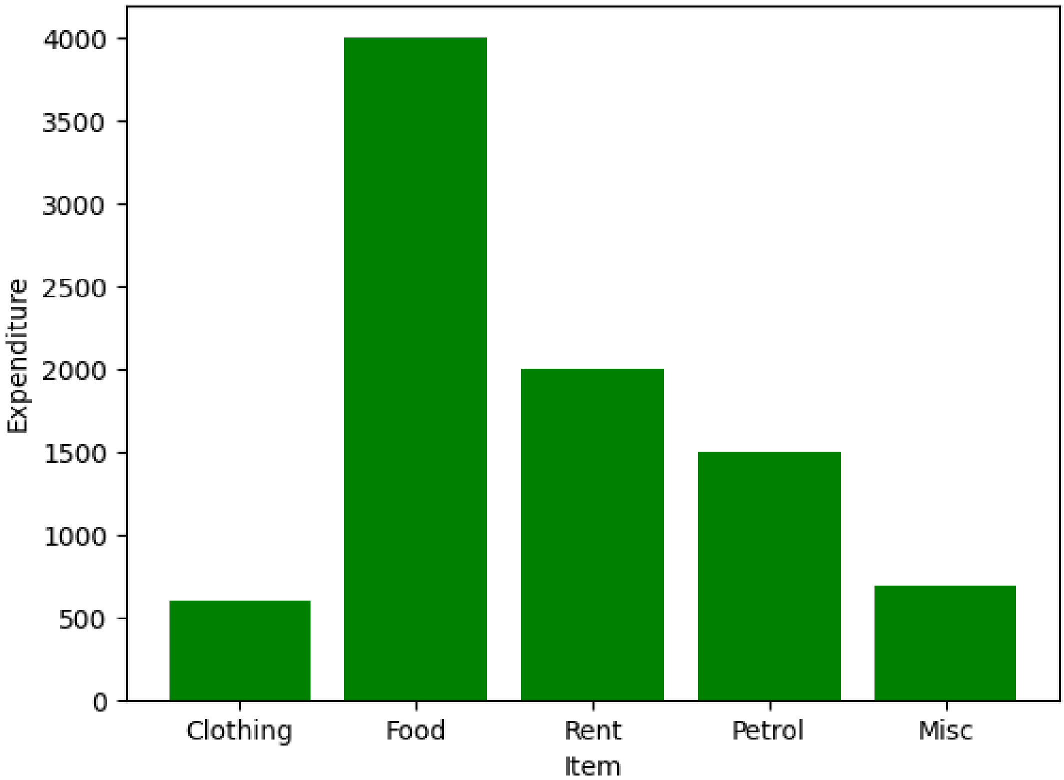
Q.3) Using python, represent the following information using a bar graph (in green color)

Item	Clothing	Food	Rent	Petrol	Misc
Expenditure in Rs	60	4000	2000	1500	700

Syntax:

```
import matplotlib.pyplot as plt
left = [1,2,3,4,5]
height = [600,4000,200,1500,]
tick_label=[‘clothing’,’food’,’rent’,’petrol’,’Misc’]
plt.bar (left,height,tick_label = tick_label,width = 0.8 ,color = ['green','green'])
plt.xlabel('Item')
plt.ylabel('Expenditure')
plt. show()
```

OUTPUT:



Q.4) Write a python program to rotate the ABC by 90° where A(1, 1), B(2, -2), C(1, 2).

Syntax:

```
import numpy as np
# Define the original points
A = np.array([1, 1])
B = np.array([2, -2])
C = np.array([1, 2])

# Define the rotation matrix for rotation by 90 degrees counterclockwise
angle = np.radians(90)
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                            [np.sin(angle), np.cos(angle)]])
# Apply the rotation to the points
A_rotated = np.dot(rotation_matrix, A)
B_rotated = np.dot(rotation_matrix, B)
C_rotated = np.dot(rotation_matrix, C)
# Print the rotated points
print("Rotated Point A: ", A_rotated)
print("Rotated Point B: ", B_rotated)
print("Rotated Point C: ", C_rotated)
```

Output:

```
Rotated Point A:  [-1.  1.]
Rotated Point B:  [2. 2.]
Rotated Point C:  [-2.  1.]
```

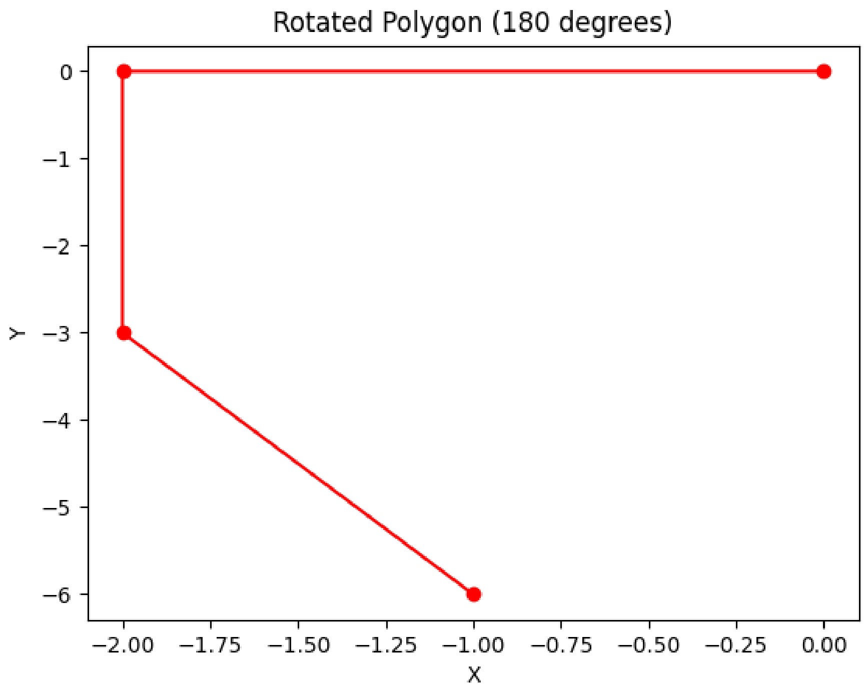
Q.5) Write a Python program to draw a polygon with vertices (0, 0), (2, 0), (2, 3) and (1, 6) and rotate it by 180° .

Syntax:

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

```python
plt.xlabel('X')
plt.ylabel('Y')
# Define the rotation matrix for 180 degrees
theta = np.pi  # 180 degrees
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                            [np.sin(theta), np.cos(theta)]])
# Apply rotation to the vertices
vertices_rotated = np.dot(vertices, rotation_matrix)
# Plot the rotated polygon
plt.figure()
plt.plot(vertices_rotated[:, 0], vertices_rotated[:, 1], 'ro-')
plt.title('Rotated Polygon (180 degrees)')
plt.xlabel('X')
plt.ylabel('Y')
# Show the plots
plt.show()
```

OUTPUT:



Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[5, 0], C[3,3].

Synatx:

```
import numpy as np
# Define the vertices of the triangle
A = np.array([0, 0])
B = np.array([5, 0])
C = np.array([3, 3])
# Calculate the side lengths of the triangle
AB = np.linalg.norm(B - A)
BC = np.linalg.norm(C - B)
CA = np.linalg.norm(A - C)
# Calculate the semiperimeter
s = (AB + BC + CA) / 2
# Calculate the area using Heron's formula
area = np.sqrt(s * (s - AB) * (s - BC) * (s - CA))
# Calculate the perimeter
perimeter = AB + BC + CA
# Print the results
print("Triangle ABC:")
print("Side AB:", AB)
print("Side BC:", BC)
print("Side CA:", CA)
print("Area:", area)
print("Perimeter:", perimeter)
```

OUTPUT:

```
Triangle ABC:
Side AB: 5.0
Side BC: 3.605551275463989
Side CA: 4.242640687119285
```

Area: 7.5000000000000036

Perimeter: 12.848191962583275

Transformed Point A: [38. 10.]

Transformed Point B: [35. 8.]

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

Max Z = x + y

Subjected to

x - y >= 1

x + y >= 2

x > 0 , y > 0

Syntax:

```
from pulp import *
# Create a maximization problem
prob = LpProblem("Maximization Problem", LpMaximize)
# Define the decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
y = LpVariable('y', lowBound=0, cat='Continuous')
# Define the objective function
prob += x + y, "Z"
# Define the constraints
prob += x - y >= 1
prob += x + y >= 2
# Solve the problem
prob.solve()
# Print the status of the solution
print("Status: ", LpStatus[prob.status])
```

```
# If the problem is solved successfully, print the optimal solution
if prob.status == LpStatusOptimal:
    print("Optimal Solution:")
    print("x = ", value(x))
    print("y = ", value(y))
    print("Z = ", value(prob.objective))OUTPUT:
Status: Optimal
```

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 = 3x+2y + 5z
subject to
x+ 2y + z <= 430
3x + 4z <460
X + 4y <= 120
x>=0,y>=0,z>=0

Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("Minimization Problem", LpMinimize)
# 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')
# Define the objective function
prob += 3*x + 2*y + 5*z, "Z"
# Define the constraints
prob += x + 2*y + z <= 430
prob += 3*x + 4*z <= 460
prob += x + 4*y <= 120
# Solve the problem
prob.solve()
# Print the status of the solution
print("Status: ", LpStatus[prob.status])
# If the problem is solved successfully, print the optimal solution
```

```
if prob.status == LpStatusOptimal:
    print("Optimal Solution:")
    print("x = ", value(x))
    print("y = ", value(y))
    print("z = ", value(z))
    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) Shearing in Y direction by 7 unit

(II) Scaling in X and Y direction by 3/2 and 4 unit respectively.

(III) Shearing in X and Y direction by 2 and 4 unit respectively.

(IV) Rotation About origin by an angle 45º

Syntax:

```
import numpy as np
# Initial point
P = np.array([-2, 4])
# Transformation 1: Shearing in Y direction by 7 units
shearing_matrix_1 = np.array([[1, 0],
                              [0, 1]])
shearing_matrix_1[0, 1] = 7
P_sheared_1 = np.dot(shearing_matrix_1, P)
# Transformation 2: Scaling in X and Y direction by 3/2 and 4 units respectively
scaling_matrix = np.array([[3/2, 0],
                           [0, 4]])
P_scaled = np.dot(scaling_matrix, P)
# Transformation 3: Shearing in X and Y direction by 2 and 4 units respectively
shearing_matrix_2 = np.array([[1, 0],
                              [0, 1]])
shearing_matrix_2[0, 1] = 4
shearing_matrix_2[1, 0] = 2
P_sheared_2 = np.dot(shearing_matrix_2, P)
# Transformation 4: Rotation about origin by an angle of 45 degrees
```

```
angle = np.radians(45)
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                            [np.sin(angle), np.cos(angle)]])
P_rotated = np.dot(rotation_matrix, P)
# Print the transformed points
print("Original Point: ", P)
print("Sheared in Y direction by 7 units: ", P_sheared_1)
print("Scaled in X and Y direction by 3/2 and 4 units respectively: ", P_scaled)
print("Sheared in X and Y direction by 2 and 4 units respectively: ", P_sheared_2)
print("Rotated about origin by an angle of 45 degrees: ", P_rotated)
```

OUTPUT:

```
Original Point:  [-2  4]
Sheared in Y direction by 7 units:  [26  4]
Scaled in X and Y direction by 3/2 and 4 units respectively:  [-3. 16.]
Sheared in X and Y direction by 2 and 4 units respectively:  [14  0]
Rotated about origin by an angle of 45 degrees:  [-4.24264069  1.41421356]
```

Q.10) Find the combined transformation of the line segment between the point A[3, 2] & B[2,-3] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi/6.
(II) Scaling in y-Coordinate by -4 units.
(III) Uniform scaling by -6.4units
(IV) Shearing in y – Direction by 5 unit

Syntax:

```
import numpy as np
# Define the initial points A and B
A = np.array([3, 2])
B = np.array([2, -3])
# Transformation 1: Rotation about origin through an angle of pi/6
angle_1 = np.pi/6
rotation_matrix_1 = np.array([[np.cos(angle_1), -np.sin(angle_1)],
                              [np.sin(angle_1), np.cos(angle_1)]])
A_rotated_1 = np.dot(rotation_matrix_1, A)
B_rotated_1 = np.dot(rotation_matrix_1, B)
# Transformation 2: Scaling in y-Coordinate by -4 units
scaling_matrix_2 = np.array([[1, 0],
                             [0, -4]])
A_scaled_2 = np.dot(scaling_matrix_2, A_rotated_1)
B_scaled_2 = np.dot(scaling_matrix_2, B_rotated_1)
```

```
# Transformation 3: Uniform scaling by -6.4 units
scaling_matrix_3 = np.array([[-6.4, 0],
                             [0, -6.4]])
A_scaled_3 = np.dot(scaling_matrix_3, A_scaled_2)
B_scaled_3 = np.dot(scaling_matrix_3, B_scaled_2)
# Transformation 4: Shearing in y-Direction by 5 units
shearing_matrix_4 = np.array([[1, 0],
                             [0, 1]])
shearing_matrix_4[0, 1] = 5
A_sheared_4 = np.dot(shearing_matrix_4, A_scaled_3)
B_sheared_4 = np.dot(shearing_matrix_4, B_scaled_3)
# Print the input and output points for each transformation
print("Input Point A: ", A)
print("Input Point B: ", B)
print("Transformation 1 - Rotation: ")
print("  - Rotated Point A: ", A_rotated_1)
print("  - Rotated Point B: ", B_rotated_1)
print("Transformation 2 - Scaling in y-Coordinate: ")
print("  - Scaled Point A: ", A_scaled_2)
print("  - Scaled Point B: ", B_scaled_2)
print("Transformation 3 - Uniform Scaling: ")
print("  - Scaled Point A: ", A_scaled_3)
print("  - Scaled Point B: ", B_scaled_3)
print("Transformation 4 - Shearing in y-Direction: ")
print("  - Sheared Point A: ", A_sheared_4)
print("  - Sheared Point B: ", B_sheared_4)
```

OUTPUT:

```
Input Point A:  [3 2]
Input Point B:  [ 2 -3]
Transformation 1 - Rotation:
  - Rotated Point A:  [1.59807621 3.23205081]
  - Rotated Point B:  [ 3.23205081 -1.59807621]
Transformation 2 - Scaling in y-Coordinate:
  - Scaled Point A:  [  1.59807621 -12.92820323]
  - Scaled Point B:  [3.23205081 6.39230485]
Transformation 3 - Uniform Scaling:
  - Scaled Point A:  [-10.22768775  82.74050067]
  - Scaled Point B:  [-20.68512517 -40.91075101]
Transformation 4 - Shearing in y-Direction:
  - Sheared Point A:  [403.47481562  82.74050067]
  - Sheared Point B:  [-225.23888022  -40.91075101]
Combined Transformation of A:  [403.47481562  82.74050067]
```