**Sahakar Maharshi Bhausaheb Santuji Thorat College Sangamner**

**Remark**

**Demonstrator’s Signature**

**Date:-**

**/**

**/20**

**DEPARTMENT OF COMPUTER SCIENCE**

**Sub : Mathematics**

**Name:- Gorde Yash Somnath Roll.No:-\_21 Date:- Title of the expt:- Slip no 10 Page.no:- Class:- BCS**

**Q1 ) Attempt any TWO of the following**

1. **Write a python program in 3D to rotate the point(1,0,0) through XY plane in clockwise direction(Rotation through Z-axis by an angle of 90⸰)**

**-**

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

p = [1, 0, 0]

theta = math.radians(90) cos\_theta = math.cos(theta) sin\_theta = math.sin(theta

)

rotation\_matrix = np.array([ [cos\_theta, -sin\_theta, 0],

[sin\_theta, cos\_theta, 0],

[0, 0, 1]

])

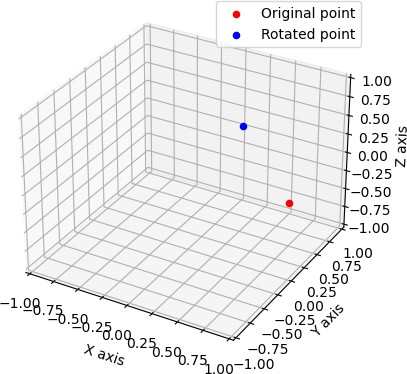
rotated\_p = rotation\_matrix.dot(p) fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d') ax.scatter(p[0], p[1], p[2], c='r', label='Original point')

ax.scatter(rotated\_p[0], rotated\_p[1], rotated\_p[2], c='b', label='Rotated point') ax.set\_xlim([-1, 1])

ax.set\_ylim([-1, 1])

ax.set\_zlim([-1, 1]) ax.set\_xlabel('X axis') ax.set\_ylabel('Y axis') ax.set\_zlabel('Z axis') ax.legend() plt.show()



1. **Represent the following information using bar graph(in green color)**

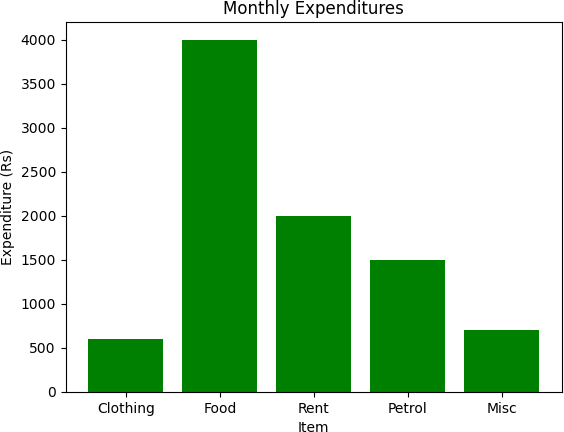
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Clothing** | **Food** | **rent** | **Petrol** | **Misc** |
| **Expenditure in RS** | **600** | **4000** | **2000** | **1500** | **700** |

**-**

import matplotlib.pyplot as plt

items = ['Clothing', 'Food', 'Rent', 'Petrol', 'Misc'] expenditures = [600, 4000, 2000, 1500, 700] plt.bar(items, expenditures, color='green') plt.title('Monthly Expenditures') plt.xlabel('Item')

plt.ylabel('Expenditure (Rs)') plt.show()



1. **Write a python program to plot a 3D line graph whose parametric equation is ( cos(2x),sin(2x),x) for 10≤x≤20 (in red color) , with titile to the graph**

**-**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D def parametric\_eq(x):

return (np.cos(2\*x), np.sin(2\*x), x)

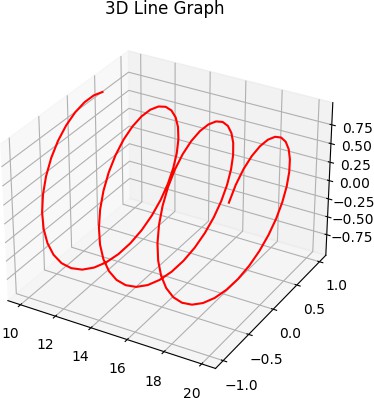
x\_vals = np.linspace(10, 20, 100)

y\_vals, z\_vals, \_ = parametric\_eq(x\_vals)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d') ax.plot(x\_vals, y\_vals, z\_vals, color='red') ax.set\_title('3D Line Graph')

plt.show()



**Q2) Attempt any TWO of the following**

**A ) Write a python program to rotate the ∆ABC,where A(1,1),B(2,-2),C(1,2)**

**-**

import numpy as np

import matplotlib.pyplot as plt

A = [1, 1]

B = [2, -2]

C = [1, 2]

theta = np.pi/4

R = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]]) A\_rot = R.dot(A)

B\_rot = R.dot(B) C\_rot = R.dot(C)

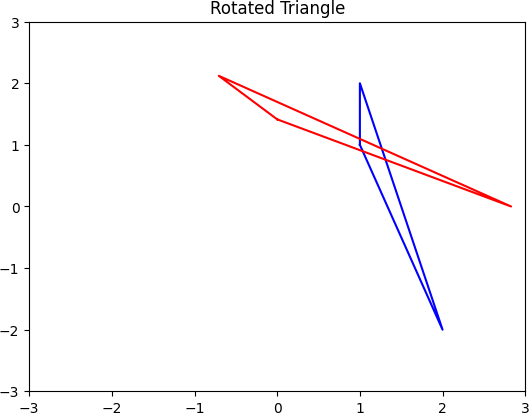
fig, ax = plt.subplots()

ax.plot([A[0], B[0], C[0], A[0]], [A[1], B[1], C[1], A[1]], 'b')

ax.plot([A\_rot[0], B\_rot[0], C\_rot[0], A\_rot[0]], [A\_rot[1], B\_rot[1], C\_rot[1], A\_rot[1]], 'r')

ax.set\_xlim([-3, 3])

ax.set\_ylim([-3, 3]) ax.set\_title('Rotated Triangle') plt.show()



**B ) Draw a polygon with vertices (0,0),(2,0),(2,3),(1,6) , write a python program t rotate the polygon by 180⸰**

**-**

import numpy as np

import matplotlib.pyplot as plt

vertices = np.array([(0, 0), (2, 0), (2, 3), (1, 6)]) theta = np.pi

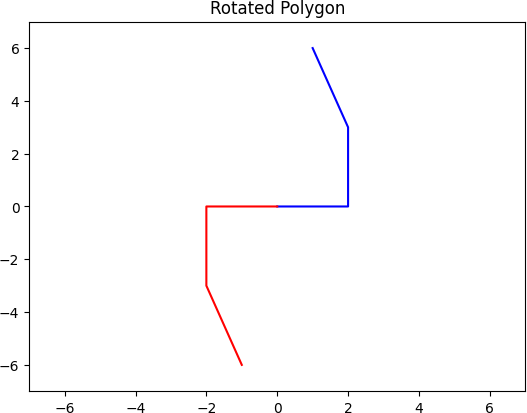
R = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]]) vertices\_rot = np.dot(vertices, R)

fig, ax = plt.subplots() ax.plot(vertices[:, 0], vertices[:, 1], 'b')

ax.plot(vertices\_rot[:, 0], vertices\_rot[:, 1], 'r')

ax.set\_xlim([-7, 7])

ax.set\_ylim([-7, 7]) ax.set\_title('Rotated Polygon') plt.show()



**C ) Find the area and perimeter of the ∆ABC where A[0,0],B[5,0],C[3,3]**

**-**

import numpy as np A = np.array([0, 0])

B = np.array([5, 0])

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

AB = np.linalg.norm(B - A) BC = np.linalg.norm(C - B) AC = np.linalg.norm(C - A)

perimeter = AB + BC + AC s = perimeter / 2

area = np.sqrt(s \* (s - AB) \* (s - BC) \* (s - AC))

print("Area:", area) print("Perimeter:", perimeter)

output :

Area: 7.5000000000000036

Perimeter: 12.848191962583275

**Q3) Attempt the following**

1. **Attempt any ONE of the following**

**I ) Solve the following LPP :**

**Max Z=x+y Subject to x-y≥1**

**X+y≥2 X,y≥0**

**-**

from scipy.optimize import linprog c = [1, 1]

A = [[-1, 1], [1, 1]]

b = [-1, 2]

bounds = [(0, None), (0, None)]

result = linprog(c, A\_ub=A, b\_ub=b, bounds=bounds, method='simplex')

if result.success: optimal\_value = result.fun optimal\_point = result.x

print("Optimal value:", optimal\_value) print("Optimal point:", optimal\_point)

else:

print("No solution found.")

output ;

Optimal value: 1.0

Optimal point: [1. 0.]

**II) Write a python program to display the following LPP by using pulp module and sim plex**

**Method .Find its optimal solution if exist Max z=3x+2y+5z**

**Subject to x+2y+z≤430**

**3x+4z≤460 X+4y≤120**

**X,y,z≥0**

**-**

from pulp import \*

prob = LpProblem("LP Problem", LpMaximize)

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

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

z = LpVariable('z', lowBound=0, cat='Continuous')

prob += 3\*x + 2\*y + 5\*z prob += x + 2\*y + z <= 430 prob += 3\*x + 4\*z <= 460 prob += x + 4\*y <= 120

prob.solve(solvers.PULP\_CBC\_CMD(msg=False))

print("Status:", LpStatus[prob.status]) if LpStatus[prob.status] == 'Optimal':

print("Optimal value:", value(prob.objective))

print("Optimal point: x = %s, y = %s, z = %s" % (value(x), value(y), value(z)))

1. **Attempt any ONE of the following**
   1. **) Write a python program to apply the following transformation on the point (-2,4) :**

**A ) Shering in Y direction by 7 units**

**B ) Scaling in X and Y direction by** 𝟑 **and 4 unit respectively**

𝟐

**C ) Shering in X and Y direction by 2 and 4 units respectively**

**D ) Rotation about origin an angle 45⸰**

**-**

import math

point = (-2, 4)

print("Original point:", point)

shear\_y = lambda x, y: (x, y+7) point = shear\_y(\*point)

print("After shearing in Y direction:", point)

scale = lambda x, y: (3/2\*x, 4\*y) point = scale(\*point) print("After scaling:", point)

shear\_x = lambda x, y: (x+2\*y, y+4\*x) point = shear\_x(\*point)

print("After shearing in X and Y direction:", point)

rotate = lambda x, y: (x\*math.cos(math.pi/4) - y\*math.sin(math.pi/4), x\*math.sin(math.pi/4) + y\*math.cos(math.pi/4))

point = rotate(\*point) print("After rotation:", point)

Output :

Original point: (-2, 4)

After shearing in Y direction: (-2, 11) After scaling: (-3.0, 44)

After shearing in X and Y direction: (85.0, 32.0)

After rotation: (37.47665940288702, 82.73149339882606)

* 1. **)Write a python program to find the combined transformation of the line segment between the points A[3,2] & B[2,-3] for the following sequence of transformation**

**A ) Rotation about origin through an angle** 𝝅

𝟔

**B ) Scaling in Y-coordinate by -4 units**

**C ) Uniform scaling by -6.4 units**

**D ) Shering in Y-direction by 5 units**

**-**

import math

import numpy as np

A = np.array([3, 2])

B = np.array([2, -3])

print("Original points:\nA =", A, "\nB =", B)

rotate = lambda point, angle: np.dot(np.array([[math.cos(angle), -math.sin(angle)],

[math.sin(angle), math.cos(angle)]]), point)

A = rotate(A, math.pi/6)

B = rotate(B, math.pi/6)

print("After rotation:\nA =", A, "\nB =", B)

scale\_y = lambda point, factor: np.array([point[0], point[1]\*factor])

A = scale\_y(A, -4)

B = scale\_y(B, -4)

print("After scaling in Y-coordinate:\nA =", A, "\nB =", B)

uniform\_scale = lambda point, factor: np.array([point[0]\*factor, point[1]\*factor])

A = uniform\_scale(A, -6.4)

B = uniform\_scale(B, -6.4)

print("After uniform scaling:\nA =", A, "\nB =", B)

shear\_y = lambda point, factor: np.dot(np.array([[1, 0],

[factor, 1]]), point)

A = shear\_y(A, 5)

B = shear\_y(B, 5)

print("After shearing in Y-direction:\nA =", A, "\nB =", B)

Output :

Original points:

A = [3 2]

B = [ 2 -3]

After rotation:

A = [1.59807621 3.23205081]

B = [ 3.23205081 -1.59807621]

After scaling in Y-coordinate:

A = [ 1.59807621 -12.92820323]

B = [3.23205081 6.39230485]

After uniform scaling:

A = [-10.22768775 82.74050067]

B = [-20.68512517 -40.91075101]

After shearing in Y-direction:

A = [-10.22768775 31.60206191]

B = [ -20.68512517 -144.33637685]