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 18 **Expt. No .** 18 **Class :-** S.Y.BCS

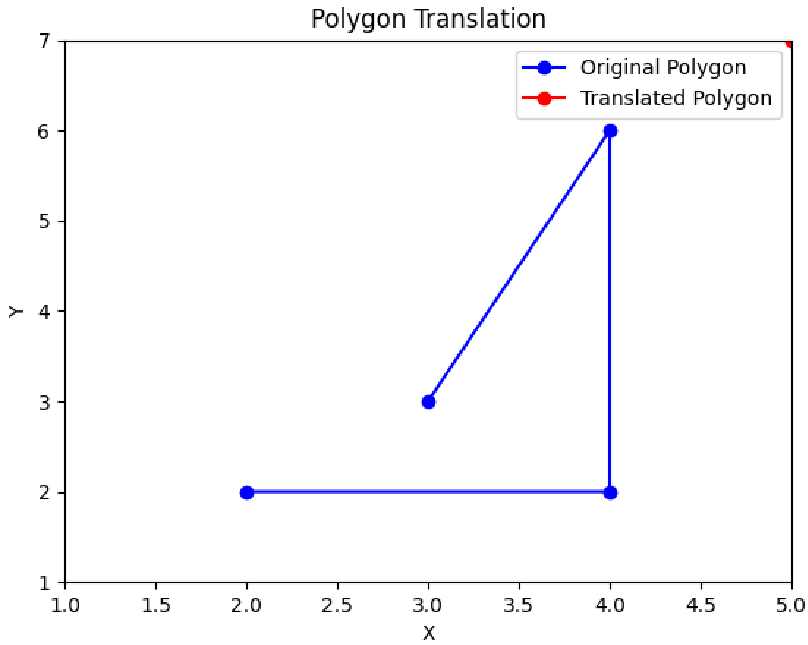
Q.1) Write a python program to draw polygon with vertices [3,3],[4,6],[4,2] and [2,2] and its translation in x and y direction by factor 3 and 5 respectively.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Given vertices of the polygon
vertices = np.array([[3, 3], [4, 6], [4, 2], [2, 2]])
# Plot the original polygon
plt.plot(vertices[:, 0], vertices[:, 1], 'bo-', label='Original Polygon')
# Translation factors
tx = 3  # Translation in x-direction
ty = 5  # Translation in y-direction
# Translated vertices
translated_vertices = vertices + np.array([tx, ty])
# Plot the translated polygon
plt.plot(translated_vertices[:, 0], translated_vertices[:, 1], 'ro-', label='Translated
Polygon')
# Set x and y axis limits
plt.xlim(vertices[:, 0].min() - 1, vertices[:, 0].max() + 1)
plt.ylim(vertices[:, 1].min() - 1, vertices[:, 1].max() + 1)
# Add legend, title and axis labels
plt.legend()
plt.title('Polygon Translation')
plt.xlabel('X')
```

```
plt.ylabel('Y')
# Show the plot
plt.show()
```

OUTPUT:



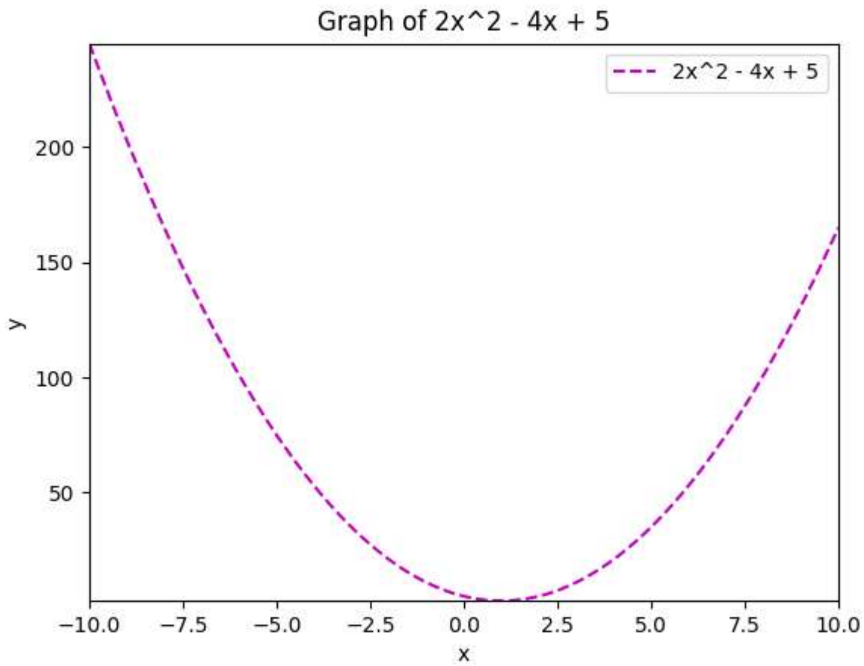
Q.2) Write a python program to plot the graph $2x^2 - 4x + 5$ in [-10,10] in magenta colored dashed pattern.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the function
def func(x):
    return 2 * x**2 - 4 * x + 5
# Generate x values in the range [-10,10]
x = np.linspace(-10, 10, 500)
# Generate y values using the function
y = func(x)
# Plot the graph with magenta colored dashed pattern
```

```
plt.plot(x, y, 'm--', label='2x^2 - 4x + 5')
# Set x and y axis limits
plt.xlim(-10, 10)
plt.ylim(y.min(), y.max())
# Add legend, title and axis labels
plt.legend()
plt.title('Graph of 2x^2 - 4x + 5')
plt.xlabel('x')
plt.ylabel('y')
# Show the plot
plt.show()
```

OUTPUT:



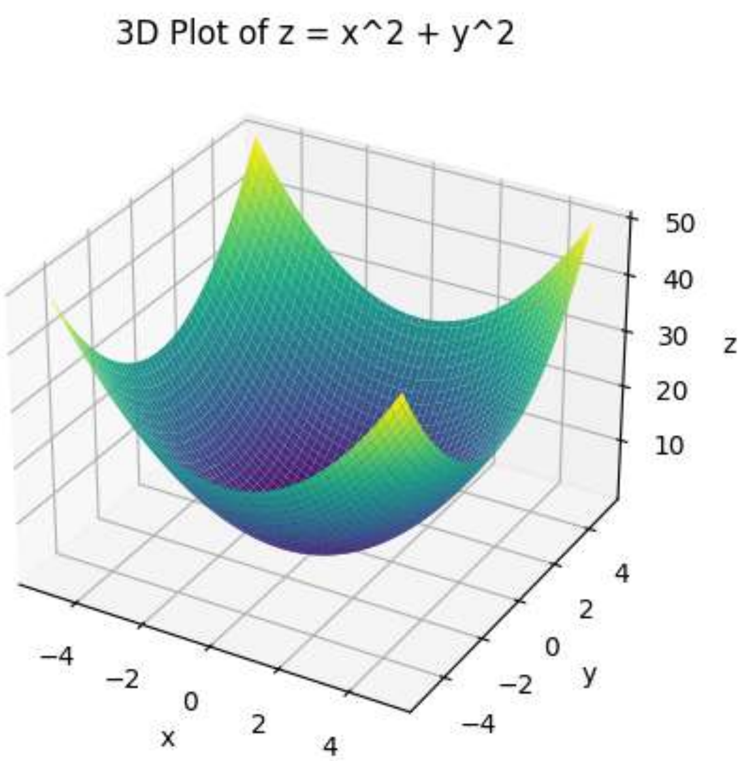
Q.3) Write a Python program to generate 3D plot of the function $z = x^2 + y^2$ in $-5 < x,y < 5$.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Generate x, y values in the range [-5, 5]
```

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
# Create a grid of x, y values
X, Y = np.meshgrid(x, y)
# Compute the corresponding z values using the function z = x^2 + y^2
Z = X**2 + Y**2
# 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')
# Set labels for x, y, and z axes
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
# Set title
ax.set_title('3D Plot of z = x^2 + y^2')
# Show the plot
plt.show()
```

OUTPUT:



Q.4) Write a Python program to generate vector x in the interval [-22, 22] using numpy package 80 subintervals

Syntax:

```
import numpy as np
# Define the interval and the number of subintervals
start = -22
end = 22
num_subintervals = 80
# Calculate the step size
step = (end - start) / num_subintervals
# Generate the vector x using numpy's arange() function
x = np.arange(start, end + step, step)
# Print the generated vector x
print(x)
```

Output:

```
[-2.20000000e+01 -2.14500000e+01 -2.09000000e+01 -2.03500000e+01
 -1.98000000e+01 -1.92500000e+01 -1.87000000e+01 -1.81500000e+01
 -1.76000000e+01 -1.70500000e+01 -1.65000000e+01 -1.59500000e+01
 -1.54000000e+01 -1.48500000e+01 -1.43000000e+01 -1.37500000e+01
 -1.32000000e+01 -1.26500000e+01 -1.21000000e+01 -1.15500000e+01
 -1.10000000e+01 -1.04500000e+01 -9.90000000e+00 -9.35000000e+00
 -8.80000000e+00 -8.25000000e+00 -7.70000000e+00 -7.15000000e+00
 -6.60000000e+00 -6.05000000e+00 -5.50000000e+00 -4.95000000e+00
 -4.40000000e+00 -3.85000000e+00 -3.30000000e+00 -2.75000000e+00
 -2.20000000e+00 -1.65000000e+00 -1.10000000e+00 -5.50000000e-01
  2.84217094e-14  5.50000000e-01  1.10000000e+00  1.65000000e+00
  2.20000000e+00  2.75000000e+00  3.30000000e+00  3.85000000e+00
  4.40000000e+00  4.95000000e+00  5.50000000e+00  6.05000000e+00
  6.60000000e+00  7.15000000e+00  7.70000000e+00  8.25000000e+00
  8.80000000e+00  9.35000000e+00  9.90000000e+00  1.04500000e+01
  1.10000000e+01  1.15500000e+01  1.21000000e+01  1.26500000e+01
  1.32000000e+01  1.37500000e+01  1.43000000e+01  1.48500000e+01
  1.54000000e+01  1.59500000e+01  1.65000000e+01  1.70500000e+01
  1.76000000e+01  1.81500000e+01  1.87000000e+01  1.92500000e+01
  1.98000000e+01  2.03500000e+01  2.09000000e+01  2.14500000e+01
  2.20000000e+01]
```

Q.5) Write a Python program to rotate the triangle ABC by 90 degree, where A[1,2], B[2, -2]and C[-1, 2].

Syntax:

```
import numpy as np
# Define the coordinates of the triangle ABC
A = np.array([1, 2])
B = np.array([2, -2])
C = np.array([-1, 2])
# Define the rotation matrix for 90 degrees counterclockwise
theta = np.deg2rad(90)
rotation_matrix=np.array([[np.cos(theta),-np.sin(theta)],[np.sin(theta),
np.cos(theta)]])
# Rotate the triangle ABC using the rotation matrix
A_rotated = np.dot(rotation_matrix, A)
B_rotated = np.dot(rotation_matrix, B)
C_rotated = np.dot(rotation_matrix, C)
# Print the coordinates of the rotated triangle
print("Original Triangle ABC:")
print("A:", A)
print("B:", B)
print("C:", C)
print("\nRotated Triangle ABC (90 degrees counterclockwise):")
print("A_rotated:", A_rotated)
print("B_rotated:", B_rotated)
print("C_rotated:", C_rotated)
```

OUTPUT:

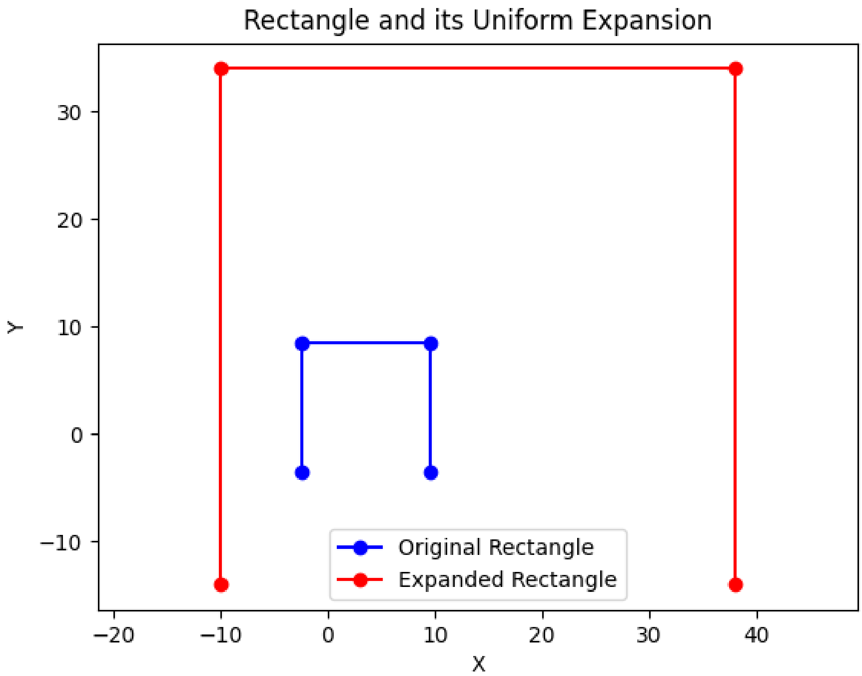
```
Original Triangle ABC:
A: [1 2]
B: [ 2 -2]
C: [-1  2]
Rotated Triangle ABC (90 degrees counterclockwise):
A_rotated: [-2.  1.]
B_rotated: [2. 2.]
C_rotated: [-2. -1.]
```

Q.6) Write a Python program to plot the rectangle with vertices at [2, 1], [2, 4], [5, 4], [5, 1] and its uniform expansion by factor 4.

Synatx:

```
import matplotlib.pyplot as plt
import numpy as np
# Define the vertices of the rectangle
vertices = np.array([[2, 1], [2, 4], [5, 4], [5, 1]], dtype=float)
# Define the uniform expansion factor
expansion_factor = 4
# Calcuate the center of the rectangle
center = np.mean(vertices, axis=0)
# Translate the rectangle to the origin
vertices -= center
# Perform uniform expansion
vertices *= expansion_factor
# Translate the rectangle back to its original position
vertices += center
# Extract the x and y coordinates of the vertices
x = vertices[:, 0]
y = vertices[:, 1]
# Plot the original rectangle
plt.plot(x, y, 'bo-', label='Original Rectangle')
# Plot the expanded rectangle
plt.plot(x * expansion_factor, y * expansion_factor, 'ro-', label='Expanded
Rectangle')
# Set the aspect ratio to 'equal'
plt.axis('equal')
# Set the title and labels
plt.title('Rectangle and its Uniform Expansion')
plt.xlabel('X')
plt.ylabel('Y')
# Add a legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:



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

Max Z = 2x + 3y

Subjected to

5x – y >= 0

x + y >= 6

x > 0 , y > 0

Syntax:

```
from pulp import LpMaximize, LpProblem, LpVariable, lpSum, value
# Create a linear programming problem
prob = LpProblem("Linear Programming Problem", LpMaximize)
# Define decision variables
x = LpVariable('x', lowBound=0, cat='Continuous')
y = LpVariable('y', lowBound=0, cat='Continuous')
# Define the objective function
prob += 2*x + 3*y, "Z"
# Add inequality constraints
prob += 5*x - y >= 0, "Constraint1"
prob += x + y >= 6, "Constraint2"
# Solve the linear programming problem
prob.solve()
# Check if the opimization was successful
```

```
if prob.status == 1:
    # Extract the optimal values of x and y
    x_opt = value(x)
    y_opt = value(y)
    # Extract the optimal value of Z (objective function)
    z_opt = value(prob.objective)
    # Print the results
    print("Optimal value of x: {:.2f}".format(x_opt))
    print("Optimal value of y: {:.2f}".format(y_opt))
    print("Optimal value of Z: {:.2f}".format(z_opt))
else:
    print("Linear programming problem failed to converge.")
```

OUTPUT:

Linear programming problem failed to converge.

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: Infeasible

Q.9) Write a python program to find the combined transformation of the line segment between the points. A[3,2] and B[2,-3] for the following sequence of transformation.

(I)First rotation about origin through an angle pi/

(II) Followed by scaling in Y – coordinate by 5 units respectively

(III) Followed by reflection through the origin

Syntax:

```
import numpy as np
# Define the line segment as a numpy array
A = np.array([3, 2])
B = np.array([2, -3])
# Define the transformations as matrices
# (I) Rotation about origin through an angle pi/2
R = np.array([[0, -1], [1, 0]])
# (II) Scaling in Y-coordinate by 5 units
S = np.array([[1, 0], [0, 5]])
# (III) Reflection through the origin
F = np.array([[-1, 0], [0, -1]])
# Compute the combined transformation
T = F @ S @ R
# Apply the combined transformation to the line segment
A_new = T @ A
B_new = T @ B
# Print the results
print("Line segment before transformation:")
print("A:", A)
print("B:", B)
print("\nCombined transformation matrix:")
print(T)
print("\nLine segment after transformation:")
print("A':", A_new)
```

print("B':", B_new)

OUTPUT:
Line segment before transformation:
A: [3 2]
B: [2 -3]
Combined transformation matrix:
[[0 1]
 [-5 0]]
Line segment after transformation:
A': [2 -15]
B': [-3 -10]

Q.10) Apply each of the following transformation of the line segment on the point P[3, -1]
I. Reflection through Y-axis.
11. Scaling in X and Y direction by 1 /2 and 3 units respectively
111. Shearing in both X and Y direction by -2 and 4 units respectively.
IV. Rotation about origin by an angle 60 degrees.
Syntax:

```
import numpy as np
# Define the point P
P = np.array([3, -1])
# I. Reflection through Y-axis
T1 = np.array([[-1, 0], [0, 1]])
P1 = T1 @ P
# II. Scaling in X and Y direction by 1/2 and 3 units respectively
T2 = np.array([[1/2, 0], [0, 3]])
P2 = T2 @ P
# III. Shearing in both X and Y direction by -2 and 4 units respectively
T3 = np.array([[1, -2], [4, 1]])
P3 = T3 @ P
# IV. Rotation about origin by an angle 60 degrees
angle = np.deg2rad(60)
T4 = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
P4 = T4 @ P
# Print the results
print("Original point P:", P)
print("\nTransformation I - Reflection through Y-axis:")
print("P1:", P1)
```

```
print("\nTransformation II - Scaling in X and Y direction:")
print("P2:", P2)
print("\nTransformation III - Shearing in X and Y direction:")
print("P3:", P3)
print("\nTransformation IV - Rotation about origin by 60 degrees:")
print("P4:", P4)
```

OUTPUT:

```
Original point P: [ 3 -1]
Transformation I - Reflection through Y-axis:
P1: [-3 -1]
Transformation II - Scaling in X and Y direction:
P2: [ 1.5 -3. ]
Transformation III - Shearing in X and Y direction:
P3: [ 5 11]
Transformation IV - Rotation about origin by 60 degrees:
P4: [2.3660254  2.09807621]
```