MATHS_PRACTICAL_SEM_4_SOLVED SLIPS

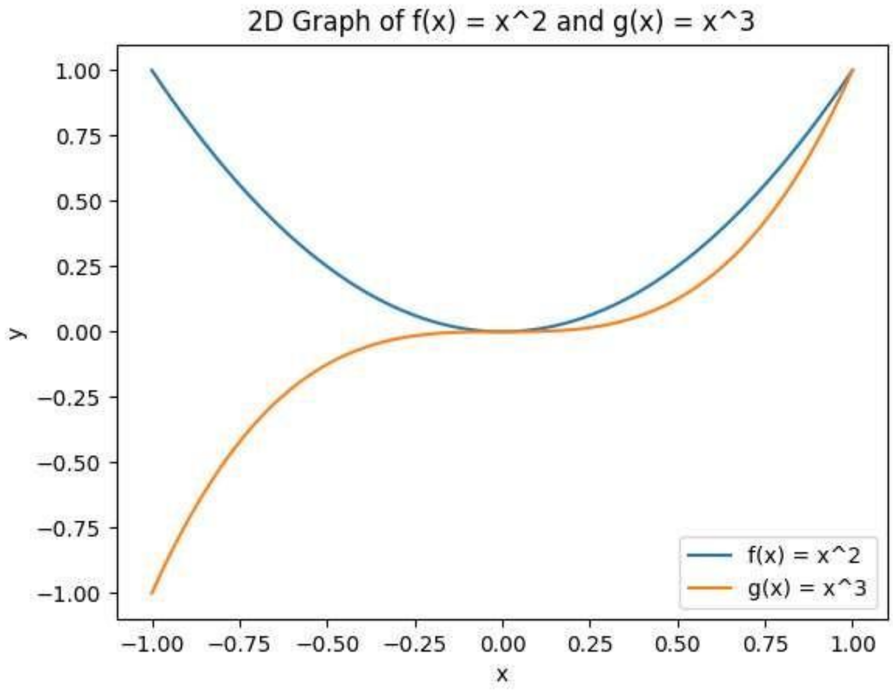
SLIP - 1

Q.1)Write a Python program to plot 2D graph of the functions $f(x) = x^2$ and $g(x) = x^3$ in [-1, 1] Syntax: import matplotlib.pyplot as plt import numpy as np def f(x):

```
   return x**2
def g(x):
   return x**3
# Generate x values in the range [-1, 1] x
= np.linspace(-1, 1, 100)
# Calculate y values for f(x) and
g(x) y_f = f(x) y_g = g(x)
# Create a figure and axes fig,
ax = plt.subplots()
# Plot f(x) and g(x) on the same
graph ax.plot(x, y_f, label='f(x) =
x^2') ax.plot(x, y_g, label='g(x) =
x^3') # Add labels and legend
ax.set_xlabel('x') ax.set_ylabel('y')
ax.legend()
```

# Set title ax.set_title('2D Graph of f(x) = x^2 and
g(x) = x^3')
# Show the plot plt.show()

OUTPUT:



Q.2) Write a Python program to plot 3D graph of the function f(x) = e**x**3 in [-5, 5] with green dashed points line with upward pointing triangle.
Syntax:
import numpy as np import
matplotlib.pyplot as plt
# Generate x values x =
np.linspace(-5, 5, 100)
# Compute y values using the given function
y = np.exp(-x**2)
# Create 3D plot fig = plt.figure() ax =
fig.add_subplot(111, projection='3d')

Plot the points with green dashed line and upward-pointing triangles ax.plot(x, y, np.zeros_like(x), linestyle='dashed', color='green', marker='^')

Set labels for axes

ax.set_xlabel('x')

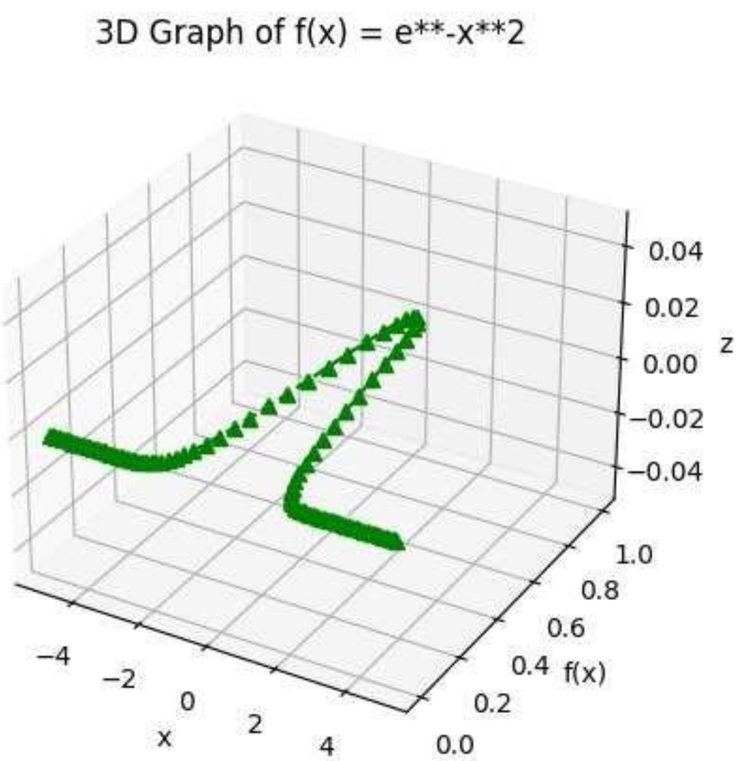
ax.set_ylabel('f(x)')

ax.set_zlabel('z') # Set title for the plot

ax.set_title('3D Graph of f(x) = e**-
x**2')

Show the plot plt.show()

OUTPUT:



Q.3) Using python, represent the foll owing information using a bar gra ph (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 =

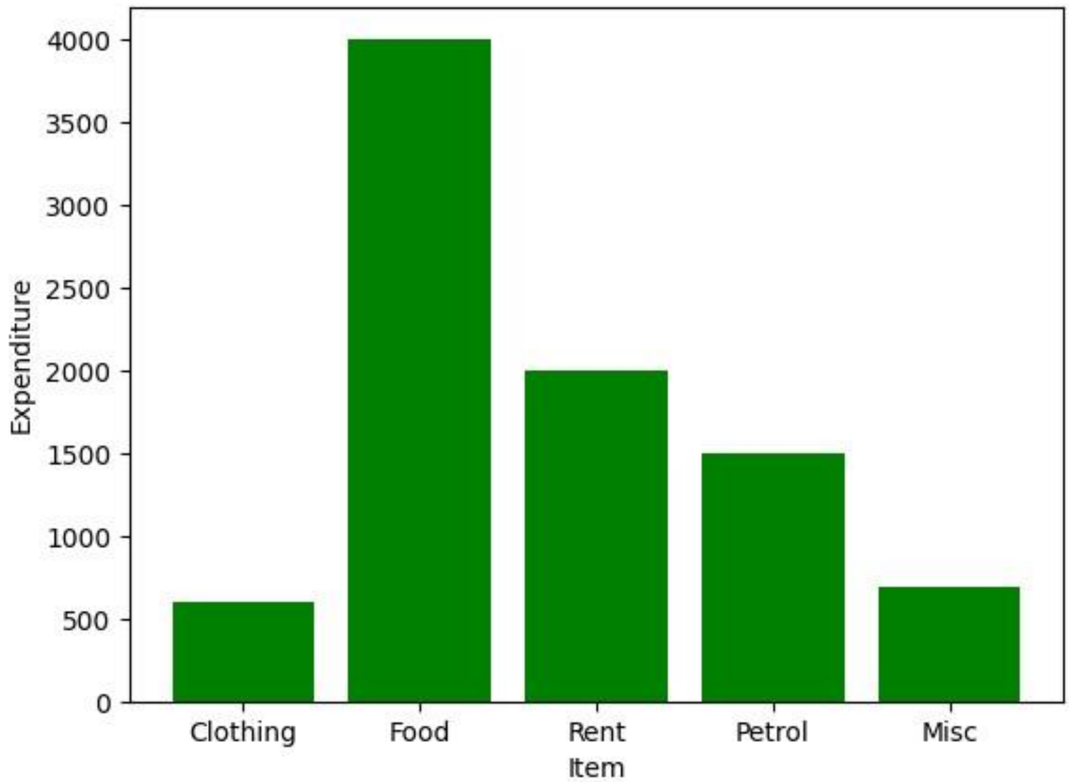
[600,4000,200,1500,]

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 reflect the line segment joining the points A[5, 3] and B[l, 4] through the line y = x + 1.

Syntax:

import numpy as np #
Define the points A and B
A = np.array([5, 3])

```
B = np.array([1, 4])
# Define the equation of the reflecting line def reflect(line,
point):     m = line[0]     c = line[1]     x, y = point     x_reflect =
(2 * m * (y - c) + x * (m ** 2 - 1)) / (m ** 2 + 1)     y_reflect =
(2 * m * x + y * (1 - m ** 2) + 2 * c) / (m ** 2 + 1)     return
np.array([x_reflect, y_reflect])
# Define the equation of the reflecting line y = x + 1
line = np.array([1, -1])
# Reflect points A and B through the reflecting line
A_reflected = reflect(line, A)
B_reflected = reflect(line, B) # Print the
reflected points print("Reflected Point
A':", A_reflected) print("Reflected Point
B':", B_reflected)
```

Output:

Reflected Point A': [4. 4.]

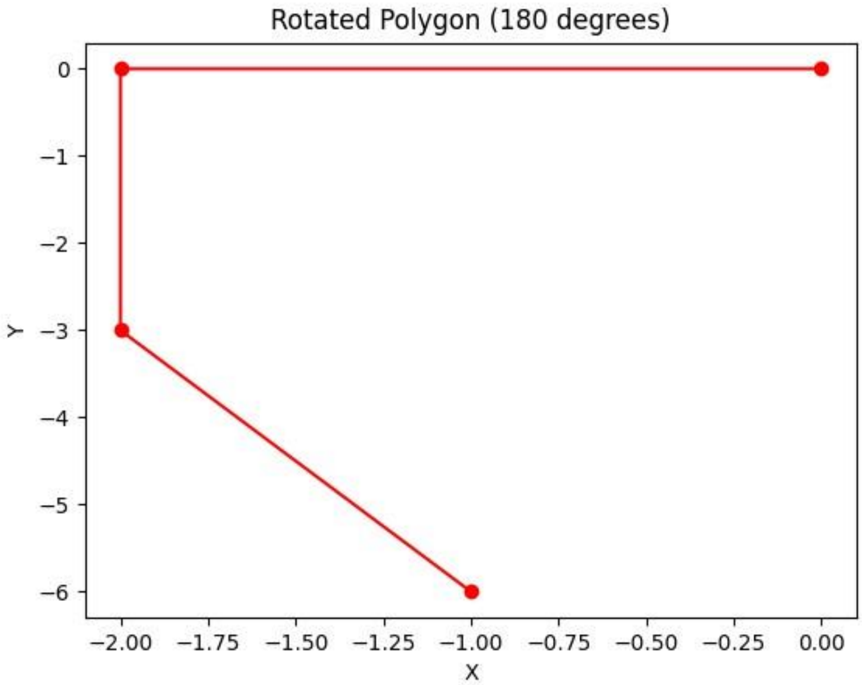
Reflected Point B': [5. 0.]

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')
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 = 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 = 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: Optimal
Status: Infeasible
```

Q.9) Apply Python. Program in each of the following transformation on the point P[3,-1]
(I)Refection through X-axis.
(II)Scaling in X-co-ordinate by factor 2.

(III) Scaling in Y-co-ordinate by factor 1.5.
(IV) Reflection through the line y = x Syntax:
Original point x = 3 y = -1 print("Original
point: ({}, {})".format(x, y)) #
Transformation 1: Reflection through X-axis
x_reflected = x y_reflected = -y
print("After reflection through X-axis: ({}, {})".format(x_reflected, y_reflected))
# Transformation 2: Scaling in X-coordinate by factor 2
x_scaled = x * 2 y_scaled = y
print("After scaling in X-coordinate by factor 2: ({}, {})".format(x_scaled,
y_scaled))
# Transformation 3: Scaling in Y-coordinate by factor 1.5
x_scaled = x y_scaled = y * 1.5
print("After scaling in Y-coordinate by factor 1.5: ({}, {})".format(x_scaled,
y_scaled))
Transformation 4: Reflection through the line y = x x_reflected
= y
y_reflected = x
print("After reflection through the line y = x: ({}, {})".format(x_reflected,
y_reflected))

OUTPUT:
Original point: (3, -1)
After reflection through X-axis: (3, 1)
After scaling in X-coordinate by factor 2: (6, -1)
After scaling in Y-coordinate by factor 1.5: (3, -1.5)
After reflection through the line y = x: (-1, 3)

Q.10) Find the combined transformation of the line segment between the point A[5, -2] & B[4, 3] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi.
(II) Scaling in X-Coordinate by 2 units.
(III) Reflection trough he line y = x (IV) Shearing in X – Direction by 4 unit Syntax: import numpy as np # Input points A and B

A = np.array([5, -2])
B = np.array([4, 3])

```
# Transformation 1: Rotation about origin through an angle pi def
rotation(pi, point):
    rotation_matrix = np.array([[np.cos(pi), -np.sin(pi)],
                               [np.sin(pi), np.cos(pi)]])
return np.dot(rotation_matrix, point)
A = rotation(np.pi, A)
B = rotation(np.pi, B)
# Transformation 2: Scaling in X-coordinate by 2 units
def scaling_x(sx, point):     scaling_matrix =
np.array([[sx, 0],
                               [0, 1]])
    return np.dot(scaling_matrix, point)
A = scaling_x(2, A)
B = scaling_x(2, B)
# Transformation 3: Reflection through the line y = -x def
reflection(line, point):
    reflection_matrix = np.array([[-line[0]**2 + line[1]**2, 2*line[0]*line[1]],
                               [2*line[0]*line[1], -line[0]**2 + line[1]**2]]) / (line[0]**2 +
line[1]**2)     return np.dot(reflection_matrix, point)
A = reflection(np.array([1, -1]), A)
B = reflection(np.array([1, -1]), B)
# Transformation 4: Shearing in X direction by 4 units
def shearing_x(shx, point):     shearing_matrix =
np.array([[1, shx],
                               [0, 1]])
    return np.dot(shearing_matrix, point)
A = shearing_x(4, A)
B = shearing_x(4, B)
# Print the transformed points A and B print("Transformed
Point A:", A)
print("Transformed Point B:", B)
```

OUTPUT:

```
Status: Infeasible
Transformed Point A: [38. 10.]
Transformed Point B: [35.  8.]
```

SLIP - 2

Q. 1)Write a Python program to plot 2D graph of the functions $f(x) = x^2$ and in [0, 10] Syntax:

import numpy as np import

matplotlib.pyplot as plt x =

np.linspace(0,10)

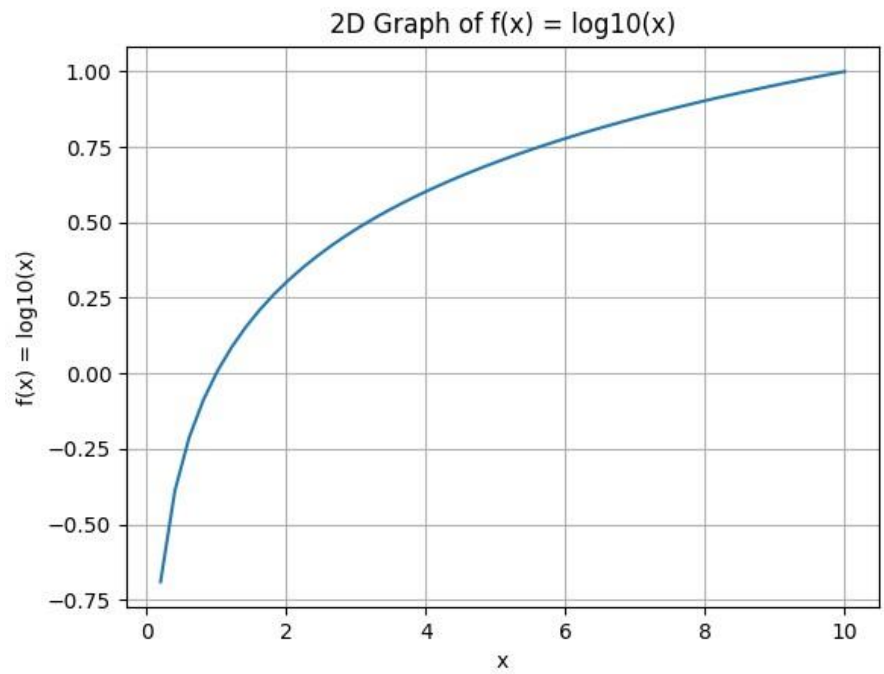
Compute y values using the function f(x) =

log10(x) 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) 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 # Generate x and y values in the

interval [0,10] x = np.linspace(0, 10, 100) y =

np.linspace(0, 10, 100) # Create a grid of x

and y values

X, Y = np.meshgrid(x, y)

Compute z values using the function f(x, y) = sin(x^2 + y^2)

Z = np.sin(X**2 + Y**2) # Create a 3D plot fig =

plt.figure() ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, Z) # Set labels and title

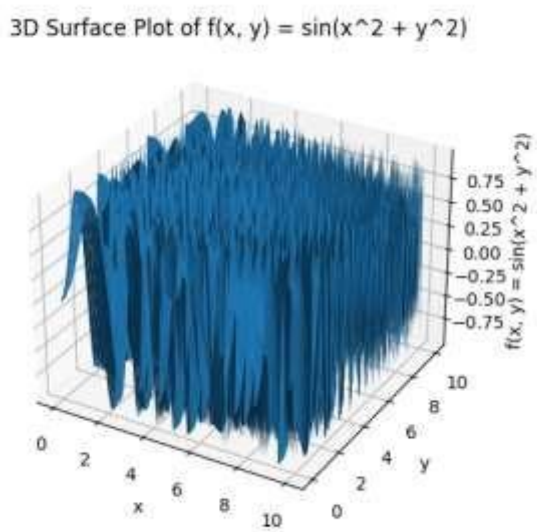
ax.set_xlabel('x') ax.set_ylabel('y') ax.set_zlabel('f(x, y)

= sin(x^2 + y^2)') ax.set_title('3D Surface Plot of f(x,

y) = sin(x^2 + y^2)')

Show the plot plt.show()

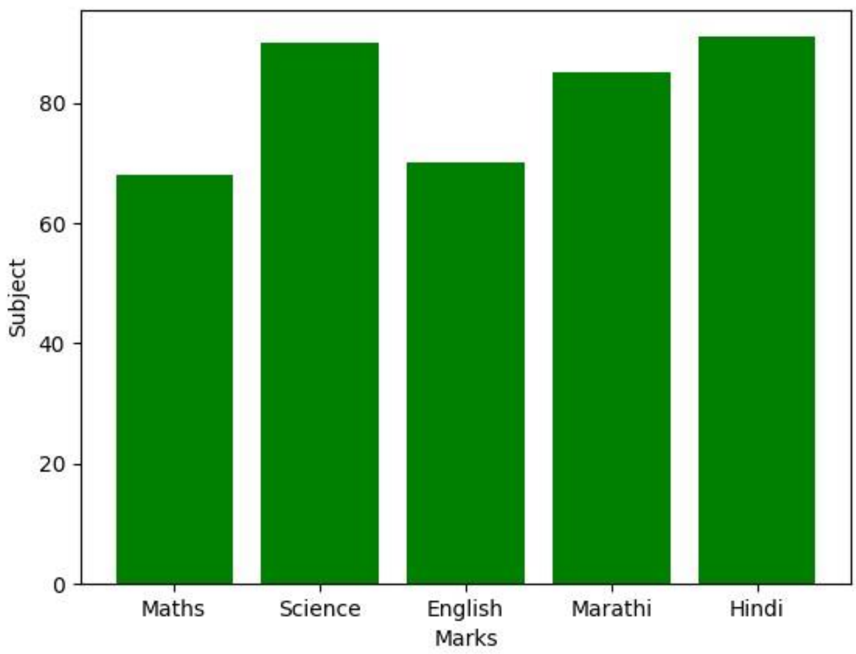
OUTPUT:



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

Subject	Maths	Science	English	Marathi	Hindi
Percentage of passing	68	90	70	85	91

Syntax: import
matplotlib.pyplot as plt left =
[1,2,3,4,5] height =
[68,90,70,85,91]
tick_label=['Maths','Science','English','Marathi',’Hindi'] 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) Using sympy declare the points A(0, 2), B(5, 2), C(3, 0) check whether these points arc collinear. Declare the line passing through the points A and B, find the distance of this line from point C.

Syntax:

```
from sympy import Point, Line
# Declare the points A, B, and C
A = Point(0, 2)
B = Point(5, 2)
C = Point(3, 0)
# Check if points A, B, and C are collinear
collinear = Point.is_collinear(A, B, C) if
collinear:
   print("Points A, B, and C are collinear.") else:
print("Points A, B, and C are not collinear.") #
Declare the line passing through points A and B
AB_line = Line(A, B)
# Find the distance of the line AB from point C distance =
AB_line.distance(C) print("Distance of the line passing through A and B
from point C: ", distance)
```

Output:

Points A, B, and C are not collinear.

Distance of the line passing through A and B from point C: 2

Q.5) Using python, drawn a regular polygon with 6 sides and radius 1 centered at (1, 2) and find its area and perimeter. Syntax: import numpy as np import matplotlib.pyplot as plt # Define the center of the hexagon center = np.array([1, 2])

Define the radius of the hexagon radius

= 1

Calculate the coordinates of the vertices of the

hexagon angles = np.linspace(0, 2*np.pi, 7)[:-1] x =

center[0] + radius * np.cos(angles) y = center[1] + radius

* np.sin(angles)

Plot the hexagon plt.plot(x, y, '-o', color='b',

label='Hexagon') plt.xlabel('x') plt.ylabel('y')

plt.title('Regular Hexagon with Radius 1 Centered at (1,

2)') plt.grid(True) plt.axis('equal') plt.legend() plt.show()

Calculate the area of the hexagon side_length = np.sqrt(3) * radius

Length of each side of the hexagon area = 3 * np.sqrt(3) / 2 *

side_length**2 # Area of the hexagon

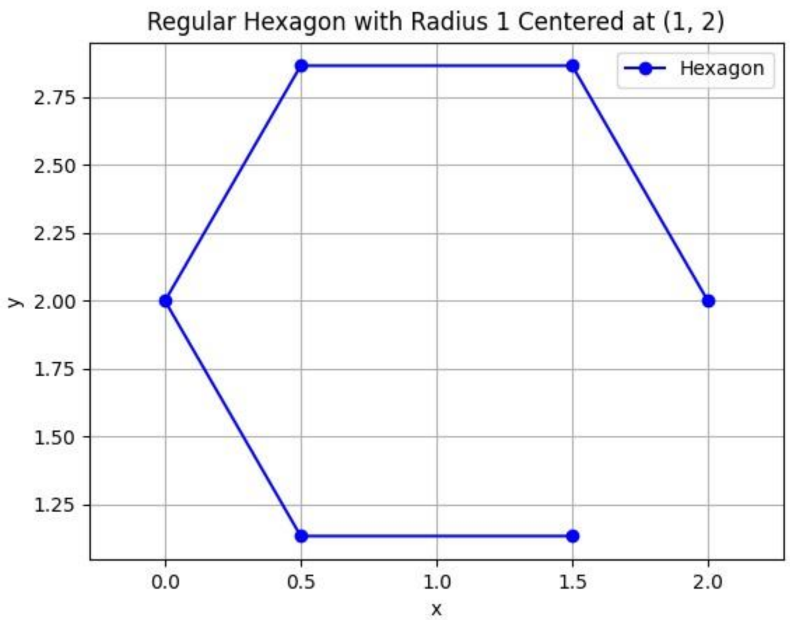
Calculate the perimeter of the hexagon perimeter = 6

* side_length # Perimeter of the hexagon print("Area

of the hexagon: ", area)

print("Perimeter of the hexagon: ", perimeter)

OUTPUT:



Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[6, 0], C[4,4].

Synatx:

import numpy as np

Define the vertices of the triangle

A = np.array([0, 0])

B = np.array([6, 0])

C = np.array([4, 4])

```
# 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:

Side AB: 6.0

Side BC: 4.47213595499958

Side CA: 5.656854249492381

Area: 11.999999999999998

Perimeter: 16.12899020449196

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

Max Z = 5x +

3y Subjected to

x + 6 <= 7 2x +

5y <= 7 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 +=
5 * x + 3 * y, "Z" # Define the constraints
problem +=  x + y <= 7, "Constraint1"
problem += 2*x +5* 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 = 7.0

Optimal y = 0.0

Optimal Z = 35.0

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 = 3x +2y +5z
subject to x + 2y + z <=
430 3x + 2z <= 460 x +

4y <= 120 x=>0,
y=>0,z=>0 Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("Linear Programming Problem", LpMinimize)
# Define decision variables x =
LpVariable('x', lowBound=0) y =
LpVariable('y', lowBound=0) z =
LpVariable('z', lowBound=0) #
Define the objective function
prob += 3 * x + 2 * y + 5 * z #
Define the constraints prob += x
+ 2 * y + z <= 430 prob += 3 * x
+ 2 * z <= 460 prob += x + 4 * y
<= 120 # Solve the problem
prob.solve()
# Print the status of the problem print("Status:",
LpStatus[prob.status])
# If the problem is solved, print the optimal solution and its value
if prob.status == LpStatusOptimal:
   print("Optimal Solution:")    print("x =",
value(x))    print("y =", value(y))    print("z =",
value(z))    print("Objective Value =",
value(prob.objective)) else:
   print("No optimal solution found.")
```

OUTPUT:

```
Status: Optimal
Optimal Solution:
x = 0.0
y = 0.0 z =
  0.0
Objective Value = 0.0
```

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 3.

(III) Scaling in Y-co-ordinate by factor 2.5.

(IV) Reflection through the line y =- x

Syntax: import numpy as np # Original point P

```
P = np.array([4, -2])
```

# Reflection through y-axis
P_reflection_y_axis = np.array([-P[0], P[1]])
# Scaling in X-coordinate by factor 3
P_scaling_x = np.array([3 * P[0], P[1]])
# Scaling in Y-coordinate by factor 2.5
P_scaling_y = np.array([P[0], 2.5 * P[1]])
# Reflection through the line y = -x
P_reflection_line = np.array([-P[1], -P[0]])
# Print the transformed points print("Original Point P: ", P)
print("Reflection through y-axis: ", P_reflection_y_axis)
print("Scaling in X-coordinate by factor 3: ", P_scaling_x)
print("Scaling in Y-coordinate by factor 2.5: ", P_scaling_y)
print("Reflection through the line y = -x: ",
P_reflection_line)

OUTPUT:
Original Point P: [4 -2]
Reflection through y-axis: [-4 -2]
Scaling in X-coordinate by factor 3: [12 -2]
Scaling in Y-coordinate by factor 2.5: [4. -5.]
Reflection through the line y = -x: [2 -4]

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) Rotation about origin through an angle pi.
(II) Shearing in y direction by 4.5 units.
(III) Scaling in X – coordinate by 3 unit.
(IV) Reflection trough he line y = x

Syntax: import
numpy as np
# Define the original points A and B
A = np.array([4, -1])
B = np.array([3, 0])
# Define the transformation matrices for each transformation
# (I) Rotation about origin through an angle pi
rotation_matrix = np.array([[np.cos(np.pi), -np.sin(np.pi)],
                    [np.sin(np.pi), np.cos(np.pi)]]) #
(II) Shearing in y direction by 4.5 units

```
shearing_matrix = np.array([[1, 0],
                            [0, 1]])
shearing_matrix[1, 0] = 4.5
# (III) Scaling in X-coordinate by 3 units
scaling_matrix = np.array([[3, 0],
                           [0, 1]])
# (IV) Reflection through the line y = x
reflection_matrix = np.array([[0, 1],
                              [1, 0]])
# Perform the combined transformation
AB_transformed =
A.dot(rotation_matrix).dot(shearing_matrix).dot(scaling_matrix).dot(reflection_
matrix)
BA_transformed =
B.dot(rotation_matrix).dot(shearing_matrix).dot(scaling_matrix).dot(reflection_
matrix)
# Print the transformed points print("Original
Point A: ", A) print("Original Point B: ", B)
print("Transformed Point A': ", AB_transformed)
print("Transformed Point B': ", BA_transformed)
```

OUTPUT:

```
Original Point A:  [ 4 -1]
Original Point B:  [3 0]
Transformed Point A':  [ 1.00000000e+00 -5.32907052e-15]
Transformed Point B':  [-3.6739404e-16 -9.0000000e+00]
```

SLIP-3

Q. 1)Write a Python program to plot graph of the functions f(x) = cos(x) in [0,2*pi] Syntax:

import numpy as np import

matplotlib.pyplot as plt x =

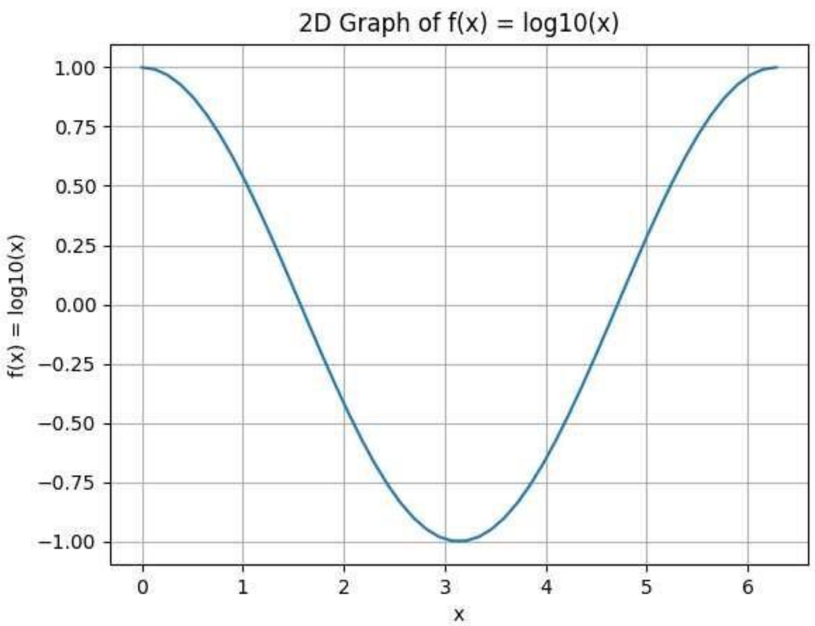
np.linspace(0,2*np.pi)

Compute y values using the function f(x) =

log10(x) y = np.cos(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 generate 3D plot of the functions z = sin x +cos y in -10 < x, y < 10. Syntax:

import numpy as np import

matplotlib.pyplot as plt from

mpl_toolkits.mplot3d import Axes3D

Generate x, y coordinates x

= np.linspace(-10, 10, 100) y

= np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)

Compute z values

Z = np.sin(X) + np.cos(Y) # Create 3D

plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

Plot the surface surf = ax.plot_surface(X, Y,

Z, cmap='viridis')

Add labels and title ax.set_xlabel('X')

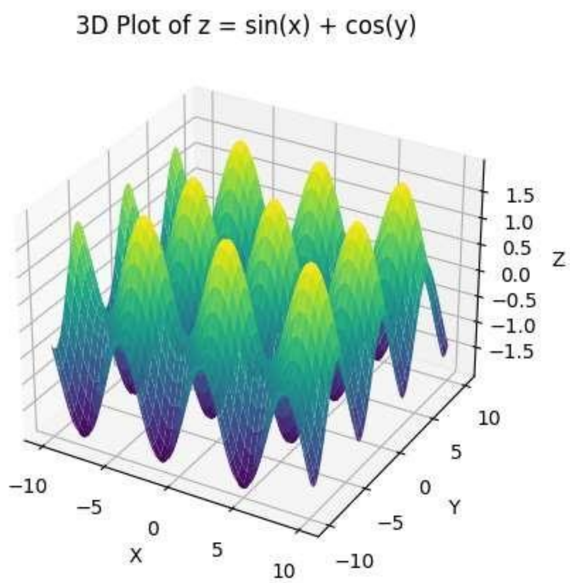
ax.set_ylabel('Y') ax.set_zlabel('Z')

ax.set_title('3D Plot of z = sin(x) +

cos(y)')

Show the plot plt.show()

OUTPUT:



Q.3) Following is the information of student participating in various games in school. Represent it by a Bar graph with bar width 0.7 inches.

Game	Cricket	Football	Hockey	Chess	Tennis
Number of student	65	30	54	10	20

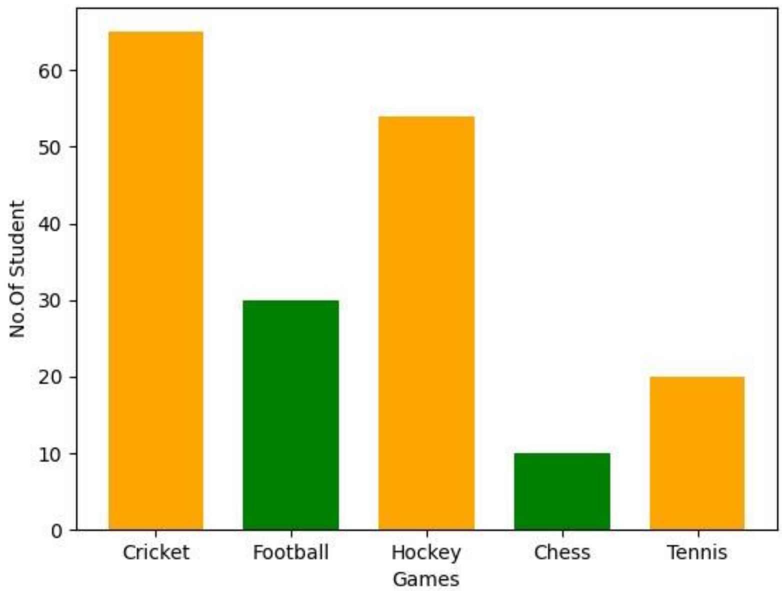
Syntax:

import matplotlib.pyplot as plt left = [1,2,3,4,5] height = [65,30,54,10,20]

tick_label=['Cricket','Football','Hockey','Chess','Tennis'] plt.bar

(left,height,tick_label = tick_label,width = 0.7 ,color = ['orange','green'])

plt.xlabel('Games') plt.ylabel('No.Of Student') plt. show()



OUTPUT:

Q.4) write a Python program to reflect the line segment joining the points A[5, 3] and B[l, 4] through the line y = x + 1.

Syntax:

```
import numpy as np #
Define the points A and B
A = np.array([5, 3])
B = np.array([1, 4])
# Define the equation of the reflecting line
def reflect(line, point):
    m = line[0]     c = line[1]     x, y = point     x_reflect = (2 * m *
(y - c) + x * (m ** 2 - 1)) / (m ** 2 + 1)     y_reflect = (2 * m * x
+ y * (1 - m ** 2) + 2 * c) / (m ** 2 + 1)     return
np.array([x_reflect, y_reflect])
# Define the equation of the reflecting line y = x + 1
line = np.array([1, -1])
# Reflect points A and B through the reflecting line
A_reflected = reflect(line, A)
B_reflected = reflect(line, B) # Print the
reflected points print("Reflected Point
A':", A_reflected) print("Reflected Point
B':", B_reflected)
```

Output:

```
Reflected Point A': [4. 4.]
Reflected Point B': [5. 0.]
```

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

Syntax:

```
import numpy as np #
Define original line points
A = np.array([2, 1])
B = np.array([4, -1])
# Define transformation matrix [T]
T = np.array([[1, 2], [2, 1]])
```

```
# Find transformed points A' and B'
A_transformed = np.dot(T, A)
B_transformed = np.dot(T, B)
# Extract coordinates of transformed points
x1_transformed, y1_transformed = A_transformed
x2_transformed, y2_transformed = B_transformed
# Find equation of transformed line
m_transformed = (y2_transformed - y1_transformed) / (x2_transformed -
x1_transformed)
b_transformed = y1_transformed - m_transformed * x1_transformed
# Format the equation of the transformed line
equation_transformed = f'y = {m_transformed} * x + {b_transformed}'
print("Equation of transformed line: ", equation_transformed)
```

OUTPUT:

Equation of transformed line: y = -1.0 * x + 9.0

Q.6) Generate line segment having endpoints (0, 0) and (10, 10) find midpoint of line segment.

Synatx:

```
# Define endpoints x1, y1
= 0, 0 x2, y2 = 10, 10 #
Calculate midpoint
midpoint_x = (x1 + x2) /
2 midpoint_y = (y1 + y2)
/ 2
# Print midpoint
print("Midpoint: ({}, {})".format(midpoint_x, midpoint_y))
```

OUTPUT:

Midpoint: (5.0, 5.0)

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

Max Z = 3.5x +

2y Subjected to x

+ y => 5 x => 15

y <= 2 x > 0 , y >

0 Syntax:

from pulp import *

Create a maximization problem problem =

LpProblem("Maximize Z", LpMaximize)

Define decision variables x = LpVariable("x",

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

lowBound=0, cat='Continuous') # Define the

objective function Z = 3.5 * x + 2 * y problem += Z

Add constraints problem += x + y >= 5 problem

+= x >= 15 problem += y <= 2 # Solve the problem

problem.solve()

Print the optimal solution and the optimal value of

Z print("Optimal solution:") print("x =", value(x))

print("y =", value(y))

print("Optimal value of Z =", value(problem.objective))

OUTPUT:

Optimal solution:

x = 15.0 y

= 2.0

Optimal value of Z = 56.5

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 = $3x_1+5x_2+4x_3$
subject to $2x_1$
+ $3x_2$ <= 8
$2x_2 + 5x_3$ <= 10

$3x_1 + 2x_2 + 4x_3 <= 15$
$X_1 => 0$, $X_2 => 0$, $X_3 => 0$

Syntax:
#By using Pulp Method from
pulp import *
# Create a minimization problem
problem = LpProblem("Minimize Z", LpMinimize)
# Define 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 Z = 3 *
x + 5 * y + 4 * z problem += Z # Add
constraints problem += 2 * x + 3 * y
<= 8 problem += 2 * y + 5 * z <= 10
problem += 3 * x + 2 * y + 4 * z <=
15
# Solve the problem
problem.solve()
# Check the status of the solution if
problem.status == LpStatusOptimal:
   # Print the optimal solution and the optimal value of Z
print("Optimal solution:")   print("x =", value(x))
print("y =", value(y))   print("z =", value(z))
print("Optimal value of Z =", value(problem.objective))
else: print("No optimal solution found.")

OUTPUT:
Optimal solution:
x = 0.0
y = 0.0
z = 0.0
Optimal value of 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([3, 5, 4])
# Define the coefficients of the inequality constraints (Ax <= b)
A = np.array([[2, 3, 0],
            [0, 2, 5],
            [3, 2, 4]])
b = np.array([8, 10, 15])
# Define the bounds for the decision variables (x >= 0)
bounds = [(0, None), (0, None), (0, None)]
# Solve the linear programming problem using the simplex method result
= linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='simplex')
# Check if an optimal solution was found
if result.success:
   # Print the optimal solution and the optimal value of
Z    print("Optimal solution:")    print("x =",
result.x[0])    print("y =", result.x[1])    print("z =",
result.x[2])    print("Optimal value of Z =", result.fun)
else:
   print("No optimal solution found.")
```

OUTPUT:
Optimal solution:
x = 0.0 y = 0.0 z
= 0.0
Optimal value of Z = 0.0

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 3.
(III) Scaling in Y-co-ordinate by factor 2.5.
(IV) Reflection through the line y = - x

```
Syntax: import
numpy as np #
Point P
```

P = np.array([4, -2]) # Reflection through y-axis
reflection_y_axis = np.array([-1, 1]) * P
# Scaling in X-coordinates by factor 3
scaling_x = np.array([3, 1]) * P # Scaling in Y-coordinates
by factor 2.5 scaling_y = np.array([1, 2.5]) * P #
Reflection through the line y = -x reflection_line =
np.array([1, -1]) * P
print("Original point P:", P) print("Reflection through y-
axis:", reflection_y_axis) print("Scaling in X-coordinates
by factor 3:", scaling_x) print("Scaling in Y-coordinates
by factor 2.5:", scaling_y) print("Reflection through the
line y = -x:", reflection_line)

OUTPUT:
Original point P: [4 -2]
Reflection through y-axis: [-4 -2]
Scaling in X-coordinates by factor 3: [12 -2]
Scaling in Y-coordinates by factor 2.5: [4. -5.]
Reflection through the line y = -x: [4 2]

Q.10) Find the combined transformation of the line segment between the point A[2, -1] & B[5, 4] by using Python program for the following sequence of transformation:-
(I) Rotation about origin through an angle
pi. (II) Scaling in X-Coordinate by 3 units.
(III) Shearing in X – Direction by 6 unit (IV)
Reflection trough he line y = x Syntax:
import numpy as np #
Define the points A and B
A = np.array([2, -1])
B = np.array([5, 4])
# Transformation 1: Rotation about origin through an angle of pi (180 degrees)
rotation_angle = np.pi
rotation_matrix = np.array([[np.cos(rotation_angle), -np.sin(rotation_angle)],
[np.sin(rotation_angle), np.cos(rotation_angle)]])
A_rotation = np.dot(rotation_matrix, A)
B_rotation = np.dot(rotation_matrix, B)
# Transformation 2: Scaling in X-coordinate by 3 units
scaling_x = np.array([3, 1]) A_scaling_x
= scaling_x * A

B_scaling_x = scaling_x * B
# Transformation 3: Shearing in X-direction by 6 units shearing_x
= np.array([1, 0]) + np.array([6, 0])
A_shearing_x = A + shearing_x * A
B_shearing_x = B + shearing_x * B
# Transformation 4: Reflection through the line y = x
reflection_line = np.array([1, -1])
A_reflection_line = reflection_line * A
B_reflection_line = reflection_line * B
print("Original line segment between A and B:")
print("A =", A) print("B =", B)
print("Transformation 1: Rotation about origin through an angle of pi:")
print("A after rotation =", A_rotation) print("B after rotation =", B_rotation) print("Transformation 2: Scaling in X-coordinate by 3 units:") print("A after scaling in X-coordinate =", A_scaling_x)
print("B after scaling in X-coordinate =", B_scaling_x)
print("Transformation 3: Shearing in X-direction by 6 units:") print("A after shearing in X-direction =", A_shearing_x) print("B after shearing in X-direction =", B_shearing_x) print("Transformation 4: Reflection through the line y = x:") print("A after reflection through y = x =", A_reflection_line) print("B after reflection through y = x =", B_reflection_line)

OUTPUT:

Status: Infeasible

Original line segment between A and B:

A = [2 -1]

B = [5 4]

Transformation 1: Rotation about origin through an angle of pi:

A after rotation = [-2. 1.]

B after rotation = [-5. -4.]

Transformation 2: Scaling in X-coordinate by 3 units: A after scaling in X-coordinate = [6 -1]

B after scaling in X-coordinate = [15 4]

Transformation 3: Shearing in X-direction by 6 units:

A after shearing in X-direction = [16 -1]

B after shearing in X-direction = [40 4]

Transformation 4: Reflection through the line y = x:

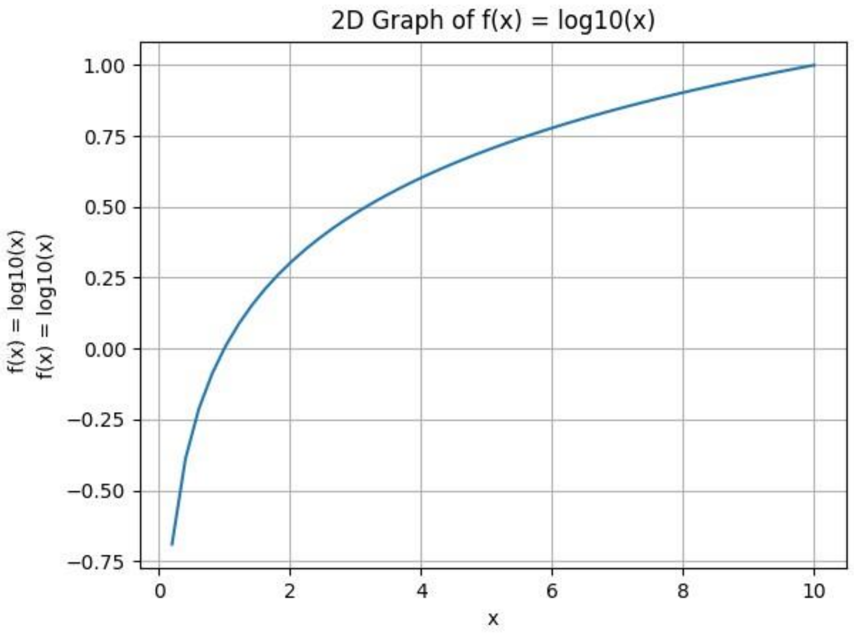
A after reflection through y = x = [2 1]

B after reflection through y = x = [5 -4]

SLIP-4

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) #

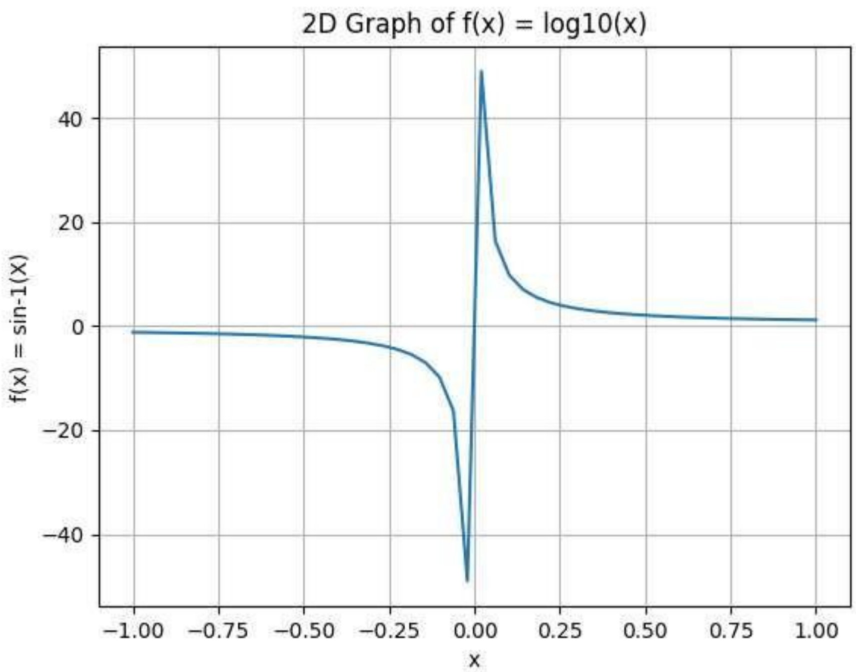
Plot the graph plt.plot(x, y)

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:

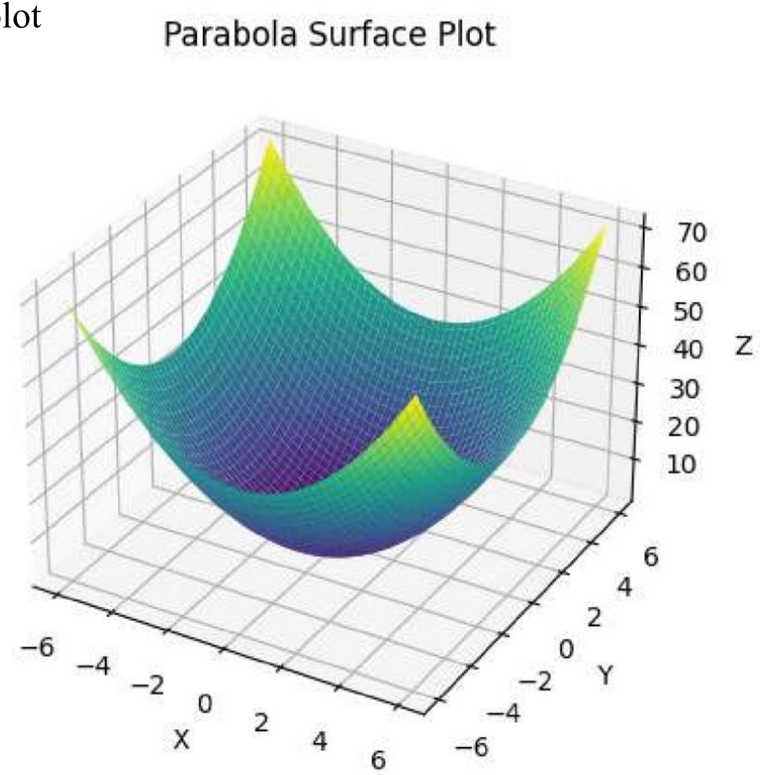


Using Python plot the surface plot of parabola z = x**2 + y**2 in -6 <x,y<6 Syntax:

Q.3)

```
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') If the line with points A[3,
1], B[5, -1] is transformed by the
# Show the plot
plt.show()
```

OUTPUT:



Q.4)

transformation matrix [T] = 3 −2

then using python, find the

equation of transformed line.

2 1

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

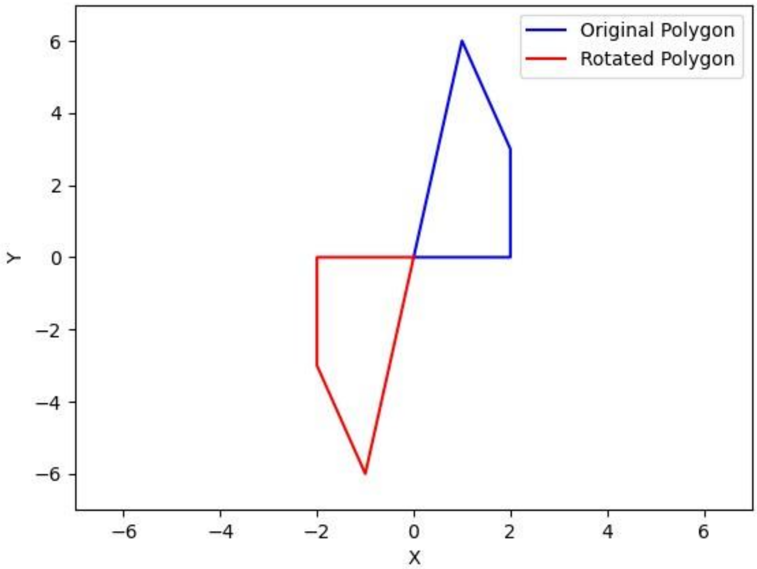
Q.5) Write a Python program to draw a polygon with vertices (0,0), (2,0), (2,3) and (1,6) and rotate by 180^0

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:



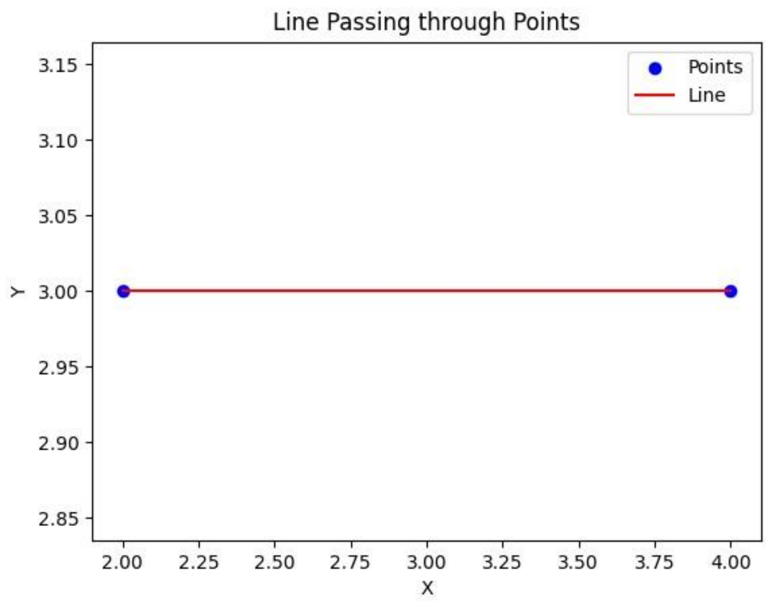
Q.6) Using python, generate line passing through points (2,3) and (4,3) and equation of the line Synatx: import matplotlib.pyplot as plt import numpy as np # Define the points x = np.array([2, 4]) y = np.array([3, 3])

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
```

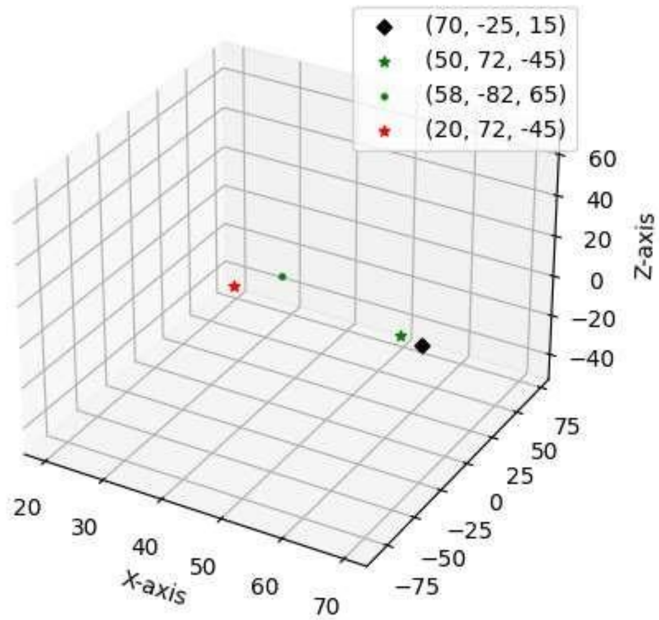
```
#by using Simplex Method
import numpy as np
from scipy.optimize import linprog
# Define the coefficients of the objective function c
= np.array([4, 1, 3, 5])
# Define the coefficients of the inequality constraints
A = np.array([[4, -6, 0, -4],
              [-8, -3, 3, 2],
              [1, 1, 0, 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]
```

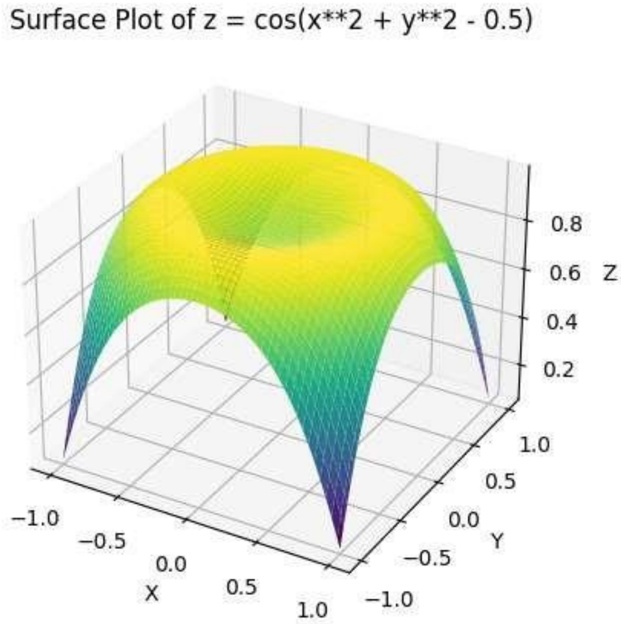
SLIP-5

Q.1) Using Python plot the surface plot of function z = cos (x**2 + y**2 - 0.5) in the interval from -1 < x,y < 1.

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
# Define the function def
func(x, y):
    return np.cos(x**2 + y**2 - 0.5)
# Generate x, y values in the interval from -1 to 1
x = np.linspace(-1, 1, 100) y = np.linspace(-1, 1,
100)
X, Y = np.meshgrid(x, y)  # Create a grid of x, y values
Z = func(X, Y)  # Compute z values using the function
# Create a 3D plot fig = plt.figure() ax =
fig.add_subplot(111, projection='3d') ax.plot_surface(X,
Y, Z, cmap='viridis')  # Plot the surface ax.set_xlabel('X')
ax.set_ylabel('Y') ax.set_zlabel('Z')
ax.set_title('Surface Plot of z = cos(x**2 + y**2 - 0.5)') plt.show()
# Show the plot
```

OUTPUT:



Q.2) 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

func(x, y):

    return np.sin(x**2 + y**2)

Generate x, y values in the interval from 0 to 10

x = np.linspace(0, 10, 100) y = np.linspace(0, 10,

100)

X, Y = np.meshgrid(x, y) # Create a grid of x, y values

Z = func(X, Y) # Compute z values using the function

Create a 3D plot fig = plt.figure() ax =

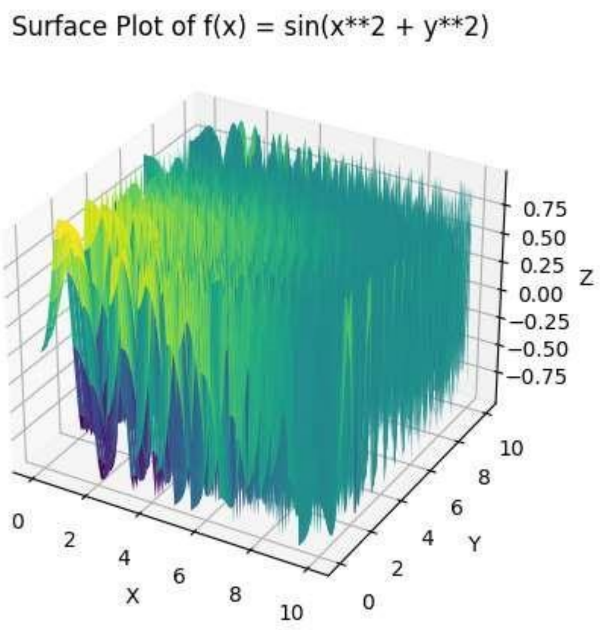
fig.add_subplot(111, projection='3d') ax.plot_surface(X,

Y, Z, cmap='viridis') # Plot the surface ax.set_xlabel('X')

ax.set_ylabel('Y') ax.set_zlabel('Z') ax.set_title('Surface

Plot of f(x) = sin(x**2 + y**2)') plt.show() # Show the plot

OUTPUT:



Q.3) Write a Python program to generate 3D plot of the functions z = sin x + cos y in the interval -10 < x,y < 10.

Syntax:

import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
Define the function def
func(x, y):
    return np.sin(x) + np.cos(y)
Generate x, y values in the interval from -10 to 10 x
= np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)
X, Y = np.meshgrid(x, y) # Create a grid of x, y values
Z = func(X, Y) # Compute z values using the function

Create a 3D plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d') ax.plot_surface(X,

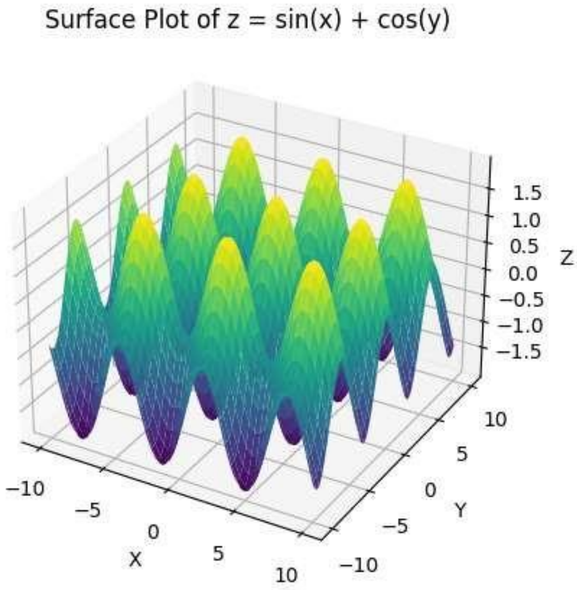
Y, Z, cmap='viridis') # Plot the surface ax.set_xlabel('X')

ax.set_ylabel('Y') ax.set_zlabel('Z')

ax.set_title('Surface Plot of z = sin(x) + cos(y)') plt.show()

Show the plot

OUTPUT:



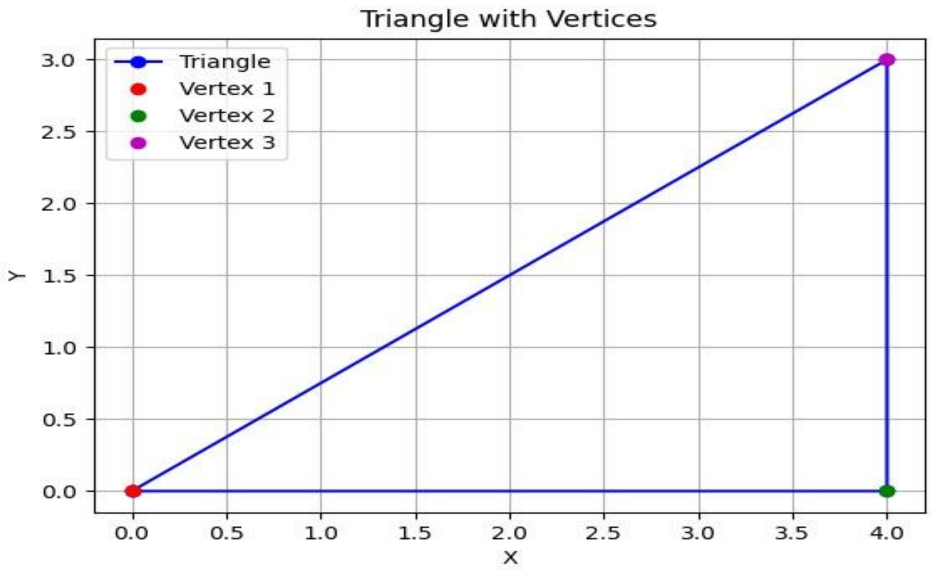
Q.4) Using python, generate triangle with vertices (0, 0), (4, 0), (4, 3) check whether the triangle is Right angle triangle.

Syntax:

```
import matplotlib.pyplot as plt #
Define the vertices of the triangle
vertex1 = (0, 0) vertex2 = (4, 0)
vertex3 = (4, 3)
# Extract x and y coordinates of the vertices x =
[vertex1[0], vertex2[0], vertex3[0], vertex1[0]] y =
[vertex1[1], vertex2[1], vertex3[1], vertex1[1]]
# Plot the triangle
```

```
plt.plot(x, y, 'b-o', label='Triangle')
plt.plot(vertex1[0], vertex1[1], 'ro', label='Vertex 1')
plt.plot(vertex2[0], vertex2[1], 'go', label='Vertex 2')
plt.plot(vertex3[0], vertex3[1], 'mo', label='Vertex 3')
plt.xlabel('X') plt.ylabel('Y')
plt.title('Triangle with Vertices')
plt.legend() plt.grid(True)
plt.show()
```

Output:



Q.5) Generate vector x in the interval [-7, 7] using numpy package with 50 subintervals. Syntax:

```
import numpy as np
# Define the interval and number of subintervals
start = -7 end = 7 num_subintervals = 50 #
Generate the vector x
x = np.linspace(start, end, num=num_subintervals+1)
# Print the vector x
print("Vector x:") print(x)
```

OUTPUT:

```
Vector x:
[-7.0000000e+00 -6.7200000e+00 -6.4400000e+00 -6.1600000e+00
 -5.8800000e+00 -5.6000000e+00 -5.3200000e+00 -5.0400000e+00
 -4.7600000e+00 -4.4800000e+00 -4.2000000e+00 -3.9200000e+00
```

```
 -3.6400000e+00 -3.3600000e+00 -3.0800000e+00 -2.8000000e+00
 -2.5200000e+00 -2.2400000e+00 -1.9600000e+00 -1.6800000e+00
 -1.4000000e+00 -1.1200000e+00 -8.4000000e-01 -5.6000000e-01
 -2.8000000e-01  8.8817842e-16  2.8000000e-01  5.6000000e-01
  8.4000000e-01  1.1200000e+00  1.4000000e+00  1.6800000e+00
  1.9600000e+00  2.2400000e+00  2.5200000e+00  2.8000000e+00
 3.0800000e+00  3.3600000e+00  3.6400000e+00  3.9200000e+00
 4.2000000e+00  4.4800000e+00  4.7600000e+00  5.0400000e+00
  5.3200000e+00  5.6000000e+00  5.8800000e+00  6.1600000e+00
  6.4400000e+00  6.7200000e+00  7.0000000e+00]
```

Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[6, 0], C[4,4].

Synatx:

```
import numpy as np

# Define the vertices of the triangle

A = np.array([0, 0])

B = np.array([6, 0])

C = np.array([4, 4])

# 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:

Side AB: 6.0

Side BC: 4.47213595499958
Side CA: 5.656854249492381

Area: 11.999999999999998

Perimeter: 16.12899020449196

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

Max Z = 5x +

3y Subjected to

x + y <= 7 2x +

5y <= 1 x > 0 y

> 0 Syntax:

from scipy.optimize import linprog

Objective function coefficients c

= [-5, -3]

Coefficient matrix of inequality constraints

A = [[1, 1],

[2, 5]]

Right-hand side of inequality constraints b

= [7, 1]

Bounds on variables

x_bounds = (0, None)

y_bounds = (0, None)

Solve the linear programming problem res = linprog(c, A_ub=A,

b_ub=b, bounds=[x_bounds, y_bounds])

Check if the optimization was successful if

res.success:

  print("Optimal solution found:")

print("x =", res.x[0])   print("y =",

res.x[1])   print("Maximum value of Z

=", -res.fun) else:

  print("Optimization failed. Message:", res.message)

OUTPUT:

Optimal solution found:

x = 0.5 y

= 0.0

Maximum value of Z = 2.5

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-5z-4w >= 10

-8x-3y+3z+2w <= 20

x + y <= 11 x >= 0,y>=
0,z>= 0,w>= 0 Syntax:

#BY using Pulp Module from pulp import LpProblem, LpMinimize,

LpVariable, lpSum, LpStatus, value

```
# Create the LPP problem
problem = LpProblem("LPP", LpMinimize)
# Define the variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) z =
LpVariable("z", lowBound=0) w =
LpVariable("w", lowBound=0) #
Define the objective function
objective = 4 * x + y + 3 * z + 5 *
w
problem += objective # Define the constraints
constraint1 = 4 * x + 6 * y - 5 * z - 4 * w >= 10
constraint2 = -8 * x - 3 * y + 3 * z + 2 * w <= 20
constraint3 = x + y <= 11 problem +=
constraint1 problem += constraint2 problem +=
constraint3
# Solve the LPP problem using the simplex method problem.solve()
# Check if the optimization was successful if
LpStatus[problem.status] == "Optimal":
   print("Optimal solution found:")
   print("x =", value(x))     print("y =", value(y))
print("z =", value(z))     print("w =", value(w))
print("Minimum value of Z =", value(objective))
else:     print("Optimization failed.")
```

OUTPUT :

Optimal solution found:

x = 0.0 y =

1.6666667 z =

0.0 w = 0.0

Minimum value of Z = 1.6666667

Q.9) Apply Python. Program in each of the following transformation on the point P[3,8]

(I)Refection through y-axis.

(II)Scaling in X-co-ordinate by factor 6.

(III) Rotation about origin through an angle 30^0
(IV) Reflection through the line y =- x Syntax:

```
import numpy as np # Point P
P = np.array([3, 8])
# Transformation I: Reflection through y-axis
P_reflection_y_axis = np.array([-P[0], P[1]])
# Transformation II: Scaling in X-coordinate by factor 6
P_scaling_x = np.array([6 * P[0], P[1]])
# Transformation III: Rotation about origin through an angle of 30 degrees theta
= np.deg2rad(30)  # Convert angle from degrees to radians
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                           [np.sin(theta), np.cos(theta)]])
P_rotation = np.dot(rotation_matrix, P)
# Transformation IV: Reflection through the line y = -x
reflection_line = np.array([[0, -1],
                           [-1, 0]])
P_reflection_line = np.dot(reflection_line, P)
# Print the results print("Original Point P:", P) print("Transformation I - Reflection through y-axis:", P_reflection_y_axis) print("Transformation II - Scaling in X-coordinate by factor 6:", P_scaling_x) print("Transformation III - Rotation about origin through an angle of 30 degrees:", P_rotation)
print("Transformation IV - Reflection through the line y = -x:",
P_reflection_line)
```

OUTPUT:

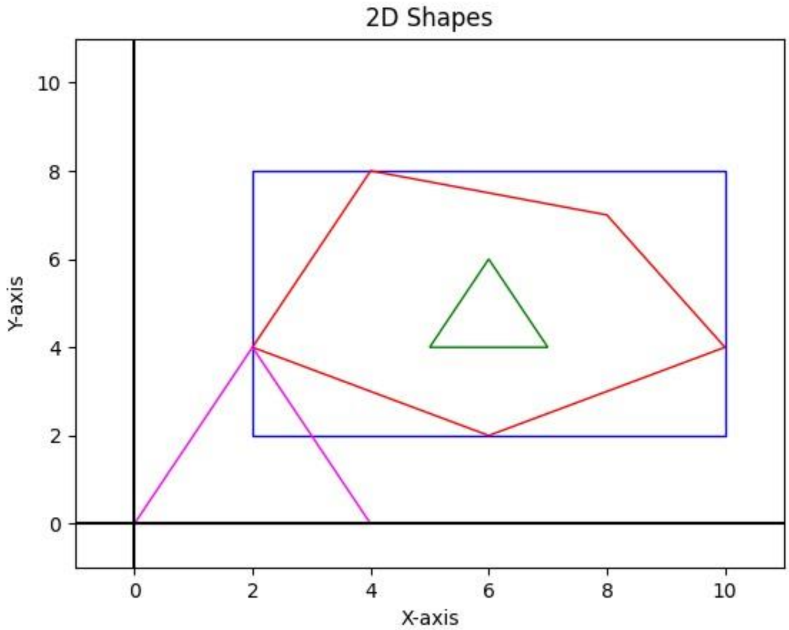
```
Original Point P: [3 8]
Transformation I - Reflection through y-axis: [-3  8]
Transformation II - Scaling in X-coordinate by factor 6: [18  8]
Transformation III - Rotation about origin through an angle of 30 degrees:
[1.40192379  8.42820323]
Transformation IV - Reflection through the line y = -x: [-8 -3]
```

Q.10) Write a python program to Plot 2D X-axis and Y-axis in black color. In the same diagram plot:-

(I) Green Triangle with vertices [5,4],[7,4],[6,6]
(II) Blue rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8].
(III) Red polygon with vertices [6, 2], [10, 4], [8, 7], [4, 8], [2, 4].
(IV) Isosceles triangle with vertices [0, 0], [4, 0], [2, 4].

Syntax:

```
import matplotlib.pyplot as plt
# Create figure and axis fig,
ax = plt.subplots()
# Plot X-axis and Y-axis in black color
ax.axhline(0, color='black') ax.axvline(0,
color='black')
# Green Triangle with vertices [5,4],[7,4],[6,6]
green_triangle = plt.Polygon([[5, 4], [7, 4], [6, 6]], edgecolor='green',
facecolor='none')
ax.add_patch(green_triangle)
# Blue Rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8]
blue_rectangle = plt.Polygon([[2, 2], [10, 2], [10, 8], [2, 8]], edgecolor='blue',
facecolor='none')
ax.add_patch(blue_rectangle)
# Red Polygon with vertices [6, 2], [10, 4], [8, 7], [4, 8], [2, 4]
red_polygon = plt.Polygon([[6, 2], [10, 4], [8, 7], [4, 8], [2, 4]], edgecolor='red',
facecolor='none') ax.add_patch(red_polygon)
# Isosceles Triangle with vertices [0, 0], [4, 0], [2, 4]
isosceles_triangle = plt.Polygon([[0, 0], [4, 0], [2, 4]], edgecolor='magenta',
facecolor='none')
ax.add_patch(isosceles_triangle)
# Set axis limits
ax.set_xlim([-1, 11])
ax.set_ylim([-1, 11]) #
Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('2D Shapes')
# Show the plot
plt.show()
```



OUTPUT:

SLIP-6

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')

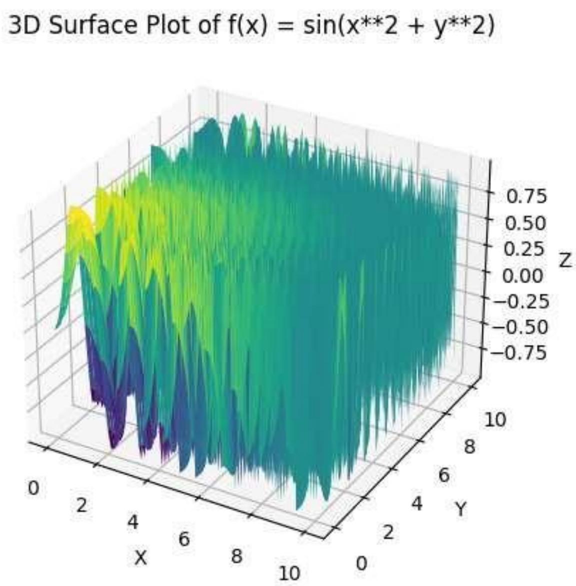
Add labels and title ax.set_xlabel('X')

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()

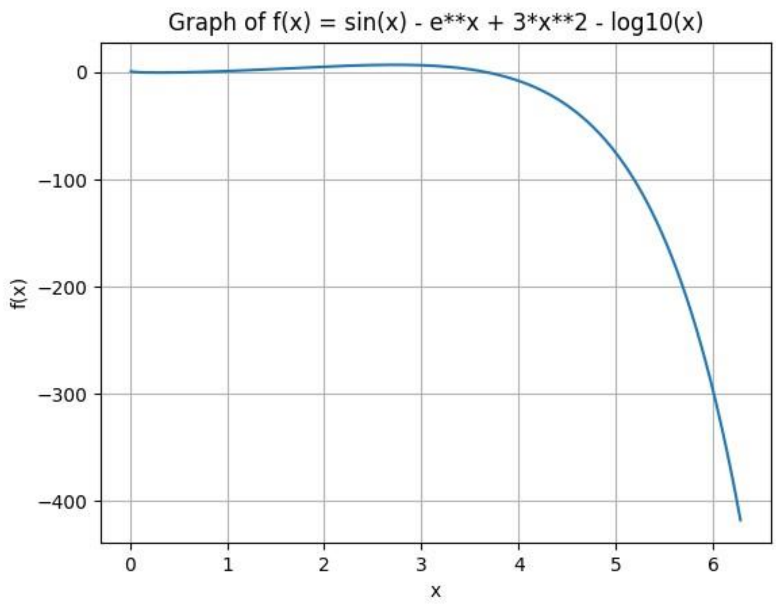
OUTPUT:



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

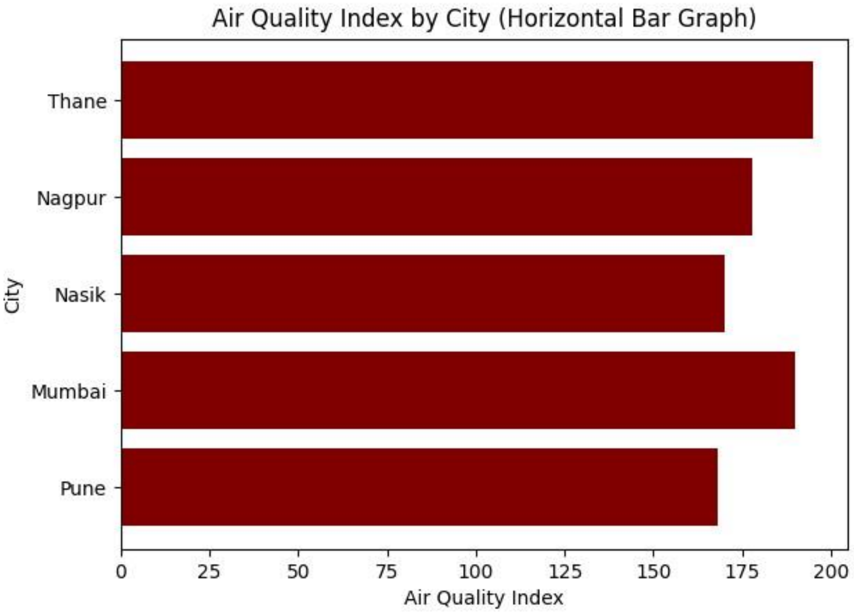
Quality Index') plt.ylabel('City')

plt.title('Air Quality Index by City

(Horizontal Bar Graph)')

Show the plot plt.show()

OUTPUT:

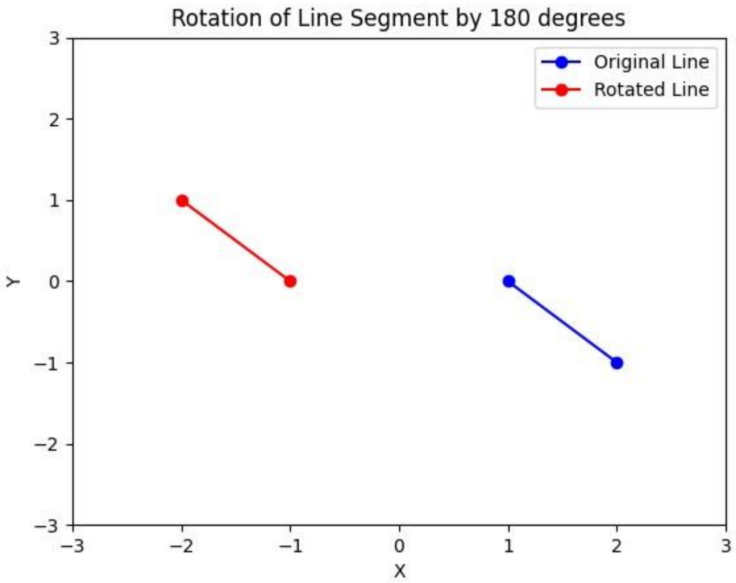


Q.4) Using python, rotate the line segment by 180° having end points (1, 0) and (2, -1)

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()
```

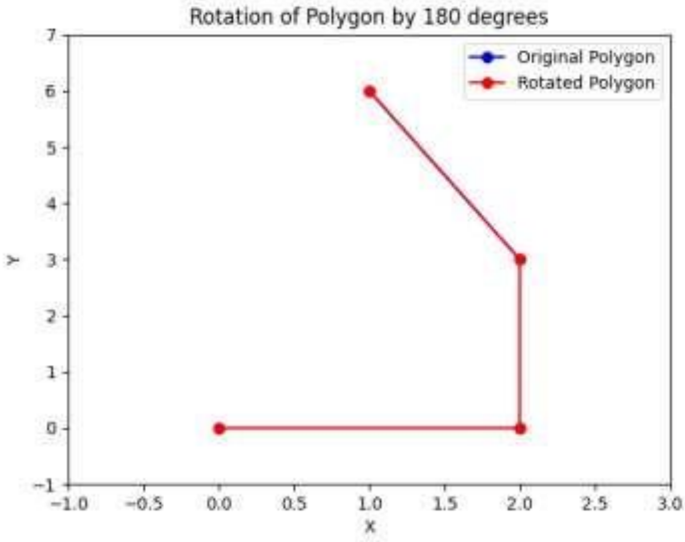
Output:



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):

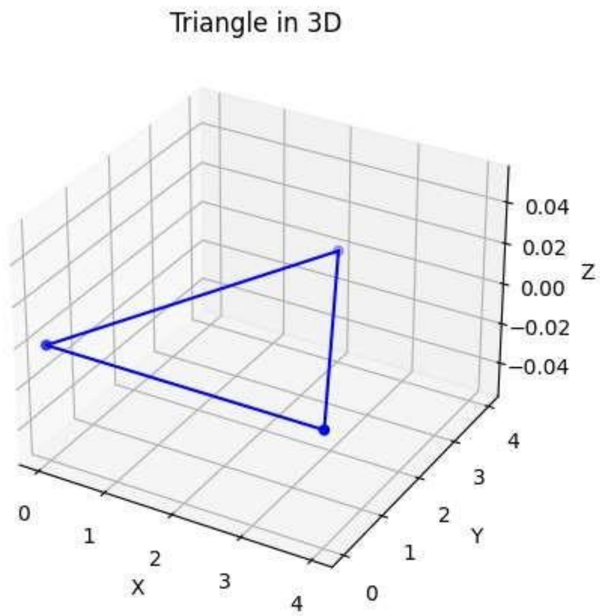
```
    ax.plot([vertices[i, 0], vertices[(i+1)%3, 0]],
            [vertices[i, 1], vertices[(i+1)%3, 1]],
            [vertices[i, 2], vertices[(i+1)%3, 2]], c='blue')
```

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°.
(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]

SLIP-7

Q.1) Plot the graph of f(x) = x**4 in [0, 5] with red dashed line with circle markers.

Syntax: import numpy as np

import matplotlib.pyplot as plt #

Define the function f(x) = x**4

def f(x):

    return x**4

Generate x values in the interval [0, 5] x

= np.linspace(0, 5, 100)

Generate y values using the function f(x) y

= f(x)

Plot the graph with red dashed line and circle markers plt.plot(x,

y, 'r--o', markersize=6)

Set x-axis label

plt.xlabel('x') # Set y-axis

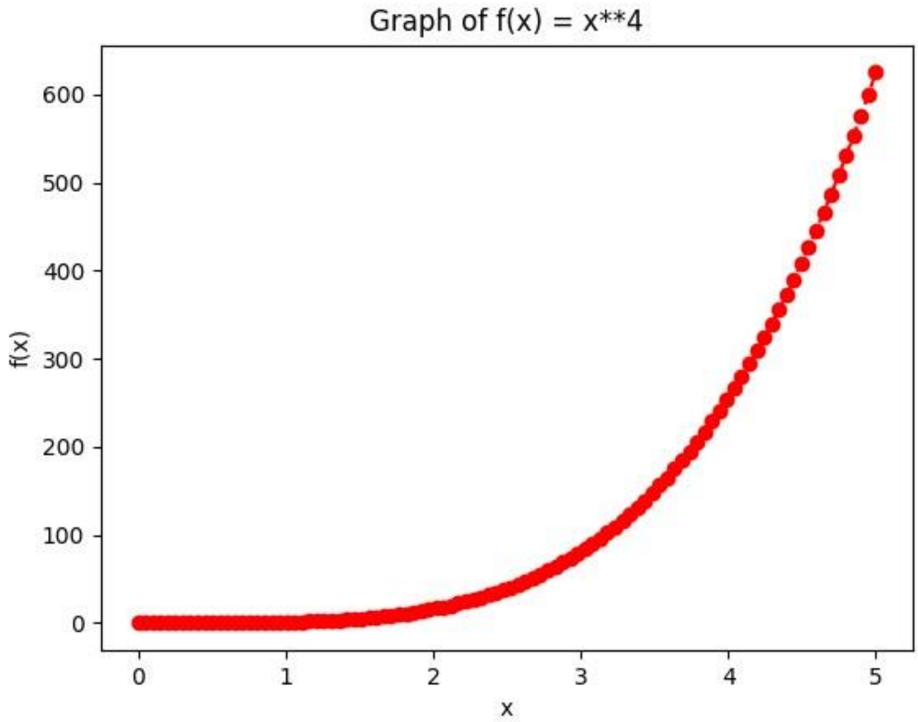
label plt.ylabel('f(x)') # Set

title plt.title('Graph of f(x) =

x**4')

Show the plot plt.show()

OUTPUT:



Q.2) 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 # Generate x and y values in the

interval [0,10] x = np.linspace(0, 10, 100) y =

np.linspace(0, 10, 100) # Create a grid of x

and y values

X, Y = np.meshgrid(x, y)

Compute z values using the function f(x, y) = sin(x^2 + y^2)

Z = np.sin(X**2 + Y**2) # Create a 3D

plot fig = plt.figure() ax =

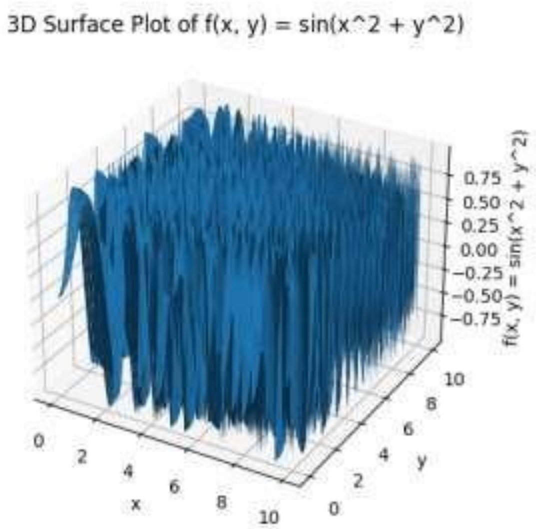
fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, Z) # Set labels and

title ax.set_xlabel('x') ax.set_ylabel('y')

ax.set_zlabel('f(x, y) = sin(x^2 + y^2)')

ax.set_title('3D Surface Plot of f(x, y) =
sin(x^2 + y^2)')
# Show the plot plt.show()



OUTPUT:

Q.3) Write python program to draw rectangle with vertices [1, 0], [2, 1], [1, 2] and [0, 1], its rotation.

Syntax: import
matplotlib.pyplot as plt import
numpy as np

Define the rectangle vertices vertices =

np.array([[1, 0], [2, 1], [1, 2], [0, 1], [1, 0]])

Extract x and y coordinates of the vertices

x = vertices[:, 0] y = vertices[:, 1]

Plot the rectangle plt.plot(x, y, '-o', label='Original Rectangle') # Calculate the rotation angle in radians theta = np.radians(45) # Rotate the rectangle vertices

rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
[np.sin(theta), np.cos(theta)]])

rotated_vertices = np.dot(vertices, rotation_matrix) # Extract x and y coordinates of the rotated vertices rotated_x = rotated_vertices[:, 0] rotated_y = rotated_vertices[:, 1] # Plot the rotated rectangle plt.plot(rotated_x, rotated_y, '-o', label='Rotated Rectangle')

Set x-axis label

plt.xlabel('x') # Set y-axis label plt.ylabel('y') # Set title

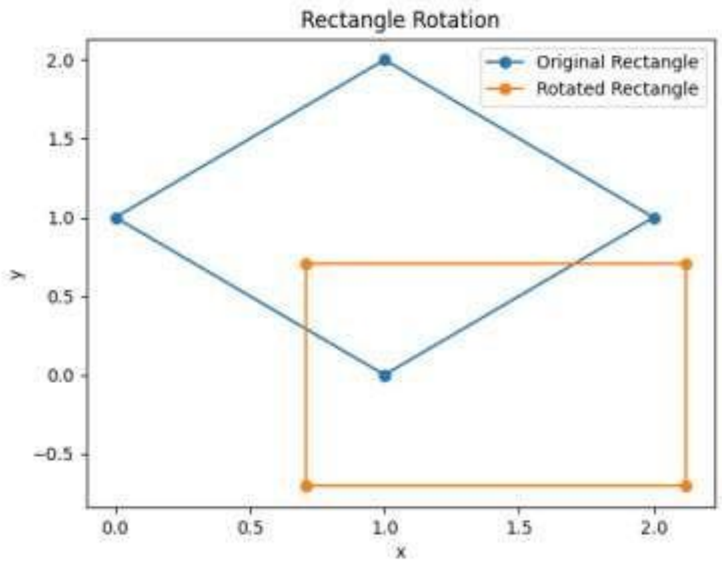
plt.title('Rectangle Rotation')

Add legend

plt.legend() # Show the plot

plt.show()

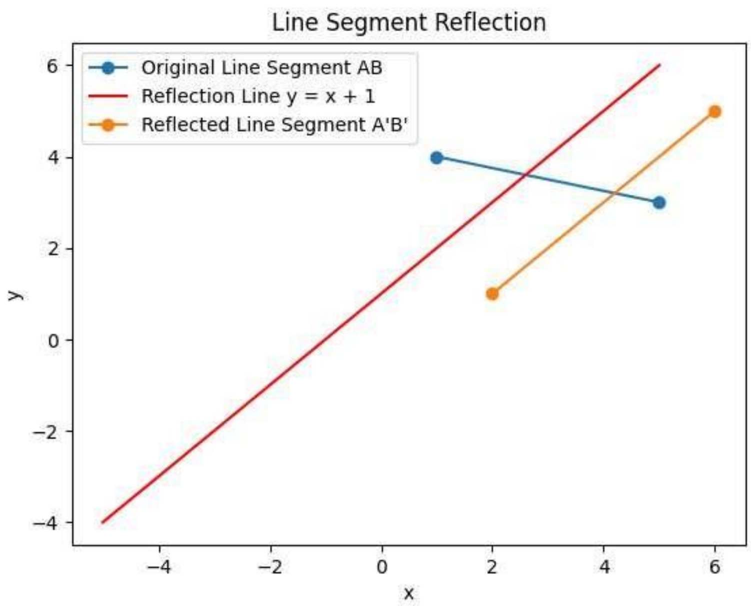
OUTPUT:



Q.4) Write a Python program to reflect the line segment joining the points A[5, 3] & B[1, 4] through the line y = x + 1.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np # Define
the points A and B
A = np.array([5, 3])
B = np.array([1, 4])
# Define the equation of the reflection line
reflection_line = lambda x: x + 1 # Plot
the original line segment AB
plt.plot([A[0], B[0]], [A[1], B[1]], '-o', label='Original Line Segment AB')
# Plot the reflection line
x_vals = np.linspace(-5, 5, 100) # Generate x values for the plot
plt.plot(x_vals, reflection_line(x_vals), '-r', label='Reflection Line y = x + 1')
# Calculate the reflected points reflected_A =
np.array([reflection_line(A[0]), A[0]]) reflected_B =
np.array([reflection_line(B[0]), B[0]])
# Plot the reflected line segment A'B'
plt.plot([reflected_A[0], reflected_B[0]], [reflected_A[1], reflected_B[1]], '-o',
label='Reflected Line Segment A\'B\'')
# Set x-axis label
plt.xlabel('x') #
Set y-axis label
plt.ylabel('y') #
Set title
plt.title('Line Segment Reflection')
# Add legend
plt.legend() #
Show the plot
plt.show()
```



Output:

Q.5) Using sympy declare the points P(5, 2), Q(5, -2), R(5, 0), check whether these points arc collinear. Declare the ray passing through the points P and Q, find the length of this ray between P and Q. Also find slope of this ray. Syntax:

from sympy import Point, Line #

Define the points P, Q, and R

P = Point(5, 2)

Q = Point(5, -2)

R = Point(5, 0)

```
# Check if points P, Q, and R are collinear
line_PQ = Line(P, Q) line_PR = Line(P,
R) collinear =
line_PQ.is_parallel(line_PR)
# Print the result if
collinear:
   print("Points P, Q, and R are collinear") else:
   print("Points P, Q, and R are not collinear")
# Calculate the length of the ray PQ
length_PQ = P.distance(Q) # Calculate
the slope of the ray PQ slope_PQ = (Q.y -
P.y) / (Q.x - P.x) # Print the length and
slope of the ray PQ print("Length of the
ray PQ:", length_PQ)
print("Slope of the ray PQ:", slope_PQ)
```

OUTPUT:

```
 Points P, Q, and R are collinear
Length of the ray PQ: 4
Slope of the ray PQ: zoo
```

Q.6) Write a Python program in 3D to rotate the point (1, 0, 0) through X Plane in anticlockwise direction (Rotation through Z axis) by an angle of 90°.

```
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
# Define the point to be rotated point
= np.array([1, 0, 0])
# Define the rotation angle in degrees angle
= np.radians(90)
```

Define the rotation matrix for rotating around the Z axis rotation_matrix

= np.array([[np.cos(angle), -np.sin(angle), 0],

[np.sin(angle), np.cos(angle), 0],

[0, 0, 1]])

Perform the rotation rotated_point =

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], c='r',

marker='o', label='Original Point')

Plot the rotated point ax.scatter(rotated_point[0], rotated_point[1],

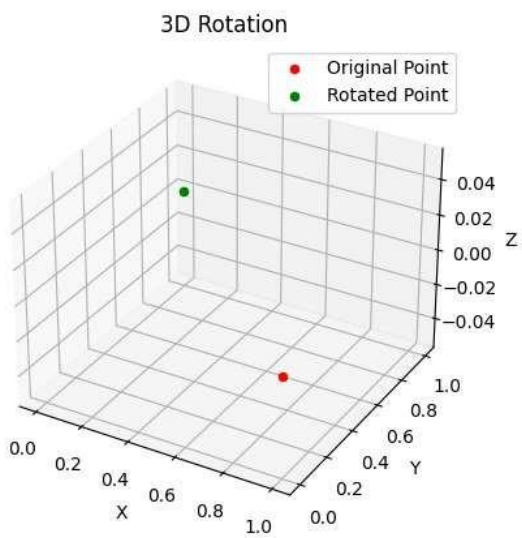
rotated_point[2], c='g', marker='o', label='Rotated Point') # Set the axes labels

ax.set_xlabel('X') ax.set_ylabel('Y') ax.set_zlabel('Z') # Set the plot title

ax.set_title('3D Rotation') # Set the plot legend ax.legend() # Show the plot

plt.show()

OUTPUT:



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

Max Z = 3.5x +2 y

Subjected to
x + y >= 5 x

>= 4 y <= 2

x > 0 , y > 0

Syntax:

import numpy as np from

scipy.optimize import linprog #

Coefficients of the objective function c

= [-3.5, -2]

Coefficients of the inequality constraints

A = [[-1, -1], [-1, 0], [0, 1]] b = [-5, -4, 2]

Bounds on the variables

x_bounds = (0, None)

y_bounds = (0, None)

Solve the linear programming problem result = linprog(c, A_ub=A,

b_ub=b, bounds=[x_bounds, y_bounds]) if result.success:

print("Optimal solution found:")

print("x =", result.x[0])   print("y =",

result.x[1])   print("Maximum value of Z

=", -result.fun) else:

print("Optimal solution not found.")

OUTPUT:

Optimal solution not found.

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 + 2y + z
subject to x + 2y +
2x <= 1 3x + 2y + z
>= 8 x + y <= 11

x>= 0, y>= 0 ,
z>=0

Syntax:

```
from pulp import *
# Create a minimization problem
prob = LpProblem("LP Problem", LpMinimize)
# Define decision variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) z =
LpVariable("z", lowBound=0)
# Objective function prob
+= x + 2 * y + z, "Z"
# Constraints prob += x + 2 * y + 2 * x <=
1, "constraint1" prob += 3 * x + 2 * y + z >=
8, "constraint2" prob += x + y <= 11,
"constraint3"
# Solve the problem using the simplex method
prob.solve(PULP_CBC_CMD())
# Print the results print("Status:",
LpStatus[prob.status]) if prob.status
== LpStatusOptimal:
   print("Optimal Solution:")     print("x =", value(x))     print("y
=", value(y))     print("z =", value(z))     print("Optimal
Objective Value (Z) =", value(prob.objective))
```

OUTPUT:

```
Status: Optimal
Optimal Solution:
x = 0.33333333 y
= 0.0 z = 7.0
Optimal Objective Value (Z) = 7.33333333
```

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 3.
(III) Rotation about origin through an angle pi
(IV) Shearing in both X and Y Direction by -2 and 4 unit Respectively.

Syntax: import
numpy as np #
Point P
P = np.array([4, -2])
# (I) Reflection through y-axis
reflection_y_axis = np.array([-1, 1]) # Reflection matrix through y-axis
P_reflected_y_axis = np.dot(reflection_y_axis, P) print("Reflection
through y-axis:", P_reflected_y_axis)
# (II) Scaling in X-coordinate by factor 3
scaling_x = np.array([3, 1]) # Scaling matrix in X-coordinate by factor 3
P_scaled_x = np.dot(scaling_x, P)
print("Scaling in X-coordinate by factor 3:", P_scaled_x)
# (III) Rotation about origin through an angle pi angle_pi
= np.pi # Angle in radians
rotation_pi = np.array([[np.cos(angle_pi), -np.sin(angle_pi)],
                        [np.sin(angle_pi), np.cos(angle_pi)]]) # Rotation matrix about
origin by angle pi
P_rotated_pi = np.dot(rotation_pi, P)
print("Rotation about origin through angle pi:", P_rotated_pi)
# (IV) Shearing in both X and Y Direction by -2 and 4 units respectively
shear_x = np.array([1, -2]) # Shearing matrix in X-direction by -2 units shear_y
= np.array([4, 1]) # Shearing matrix in Y-direction by 4 units P_sheared =
np.dot(shear_x, P) + np.dot(shear_y, P)
print("Shearing in both X and Y Direction by -2 and 4 units respectively:",
P_sheared)

OUTPUT:
Reflection through y-axis: -6
Scaling in X-coordinate by factor 3: 10
Rotation about origin through angle pi: [-4. 2.]
Shearing in both X and Y Direction by -2 and 4 units respectively: 22

Q.10) Find the combined transformation of line segment between the points A[4,1] & B[3,2] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi/4. (II) Shearing in Y direction by 7 units.
(III) Scaling in X – direction by 5 units
(IV) Reflection through y – axis Syntax:

import numpy as np #
Points A and B
A = np.array([4, -1])

```
B = np.array([3, 2])
# (I) Rotation about origin through an angle pi/4 angle_pi_4
= np.pi / 4  # Angle in radians
rotation_pi_4 = np.array([[np.cos(angle_pi_4), -np.sin(angle_pi_4)],
                          [np.sin(angle_pi_4), np.cos(angle_pi_4)]])  # Rotation matrix about origin by angle
pi/4
A_rotated = np.dot(rotation_pi_4, A)
B_rotated = np.dot(rotation_pi_4, B) #
(II) Shearing in Y direction by 7 units
shear_y_7 = np.array([0, 7])  # Shearing matrix in Y-direction by 7 units
A_sheared = A_rotated + np.dot(shear_y_7, A_rotated)
B_sheared = B_rotated + np.dot(shear_y_7, B_rotated)
# (III) Scaling in X direction by 5 units
scaling_x_5 = np.array([5, 1])  # Scaling matrix in X-direction by 5 units
A_scaled_x = np.dot(scaling_x_5, A_sheared)
B_scaled_x = np.dot(scaling_x_5, B_sheared)
# (IV) Reflection through y-axis
reflection_y_axis = np.array([-1, 1])  # Reflection matrix through y-axis
A_reflected_y_axis = np.dot(reflection_y_axis, A_scaled_x)
B_reflected_y_axis = np.dot(reflection_y_axis, B_scaled_x)
print("Line segment after applying the sequence of transformations:")
print("A:", A_reflected_y_axis) print("B:", B_reflected_y_axis)
```

OUTPUT:

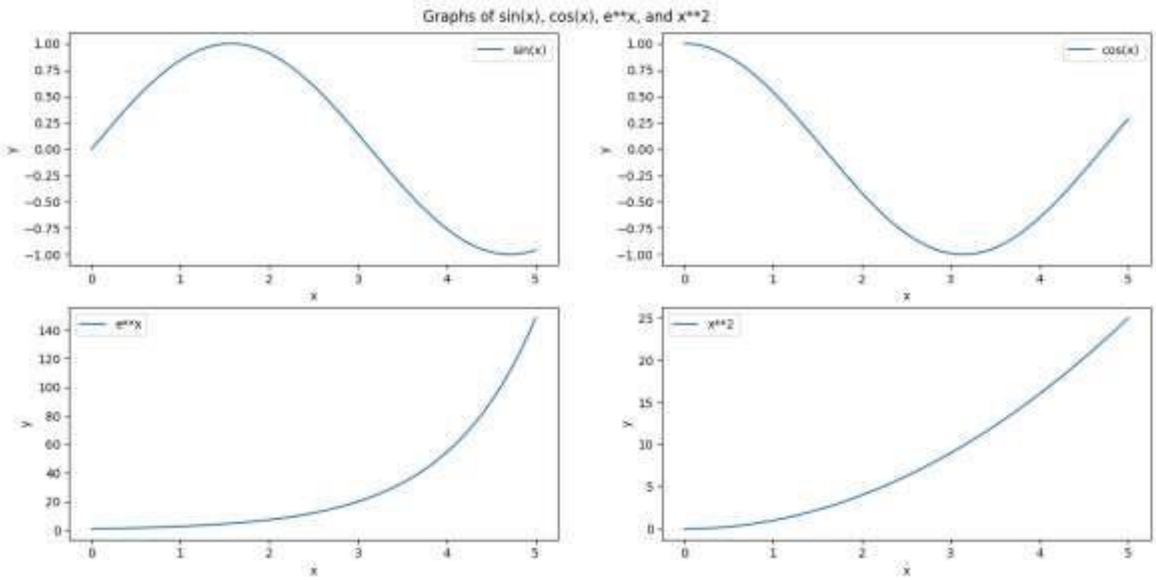
```
Line segment after applying the sequence of transformations:
A: [-108.8944443  108.8944443]
B: [-155.56349186  155.56349186]
```

SLIP-8

Q.1) Plot the graphs of sin x, cos x, e**x and x**2 in [0, 5] in one figure with (2 x 2) subplot Syntax: import numpy as np import matplotlib.pyplot as plt

# Generate x values x =
np.linspace(0, 5, 500)
# Compute y values for sin(x), cos(x), e**x, x**2 y1
= np.sin(x) y2 = np.cos(x) y3 = np.exp(x) y4 = x**2
# Create subplots fig, axs = plt.subplots(2, 2,
figsize=(10, 8)) fig.suptitle('Graphs of sin(x), cos(x),
e**x, and x**2')
# Plot sin(x) axs[0, 0].plot(x, y1,
label='sin(x)') axs[0, 0].legend() #
Plot cos(x) axs[0, 1].plot(x, y2,
label='cos(x)') axs[0, 1].legend() #
Plot e**x axs[1, 0].plot(x, y3,
label='e**x') axs[1, 0].legend()
# Plot x**2 axs[1, 1].plot(x, y4,
label='x**2') axs[1, 1].legend()
# Set x and y axis labels for all subplots
for ax in axs.flat: ax.set_xlabel('x')
ax.set_ylabel('y')
# Adjust spacing between subplots
fig.tight_layout() # Show the plot
plt.show()

OUTPUT:



Q.2) Using Python plot the graph of function f(x) = cos(x) in the interval [0, 2*pi].

Syntax:

import numpy as np import

matplotlib.pyplot as plt

Generate x values x =

np.linspace(0, 2*np.pi, 500) #

Compute y values for cos(x)

y = np.cos(x) # Plot f(x) =

cos(x) plt.plot(x, y)

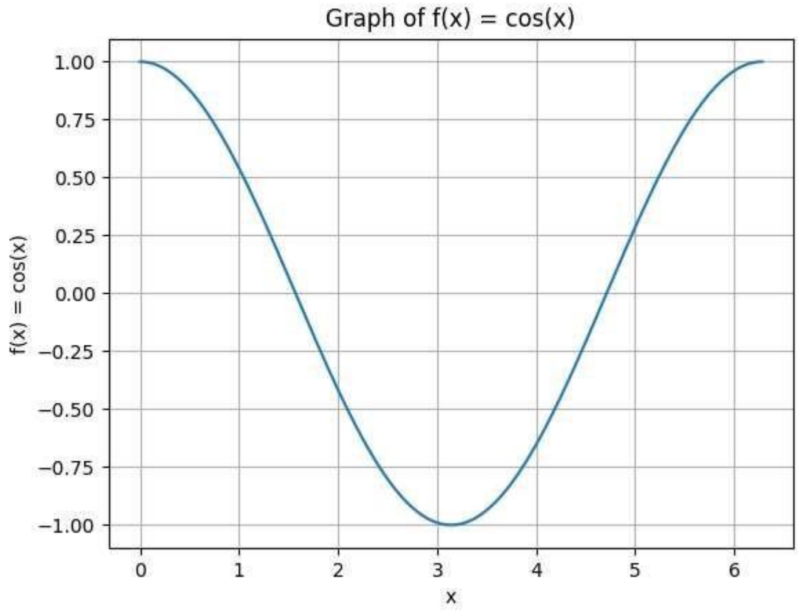
plt.xlabel('x') plt.ylabel('f(x) =

cos(x)') plt.title('Graph of f(x) =

cos(x)') plt.grid(True)

plt.show()

OUTPUT:



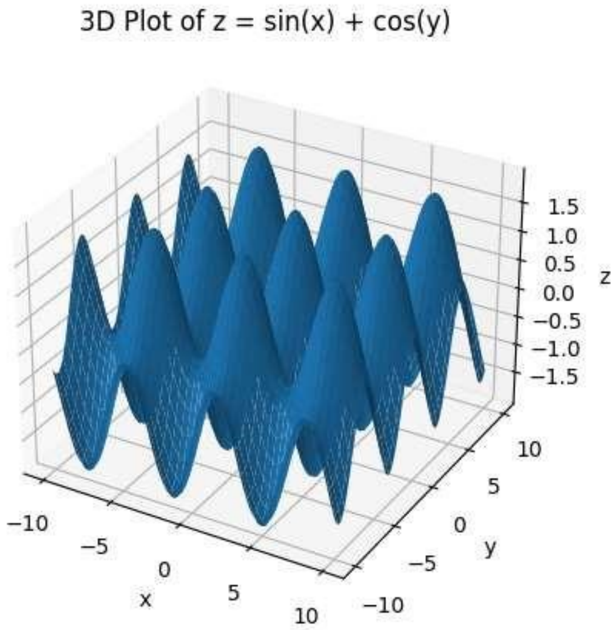
Q.3) Write a Python program to generate 3D plot of the functions z = sinx + cosy in -10 < x, y < 10.

Syntax:

import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
Generate x and y values x
= np.linspace(-10, 10, 100) y
= np.linspace(-10, 10, 100) #
Create a meshgrid of x and y
values

X, Y = np.meshgrid(x, y)
# Compute z values for the function z = sin(x) + cos(y)
Z = np.sin(X) + np.cos(Y) # Create a 3D
plot fig = plt.figure() ax =
fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z) ax.set_xlabel('x')
ax.set_ylabel('y') ax.set_zlabel('z')
ax.set_title('3D Plot of z = sin(x) + cos(y)') plt.show()

OUTPUT:



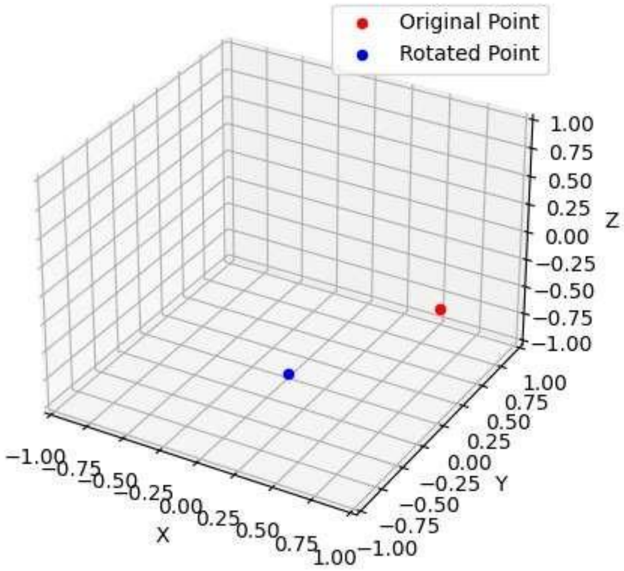
Q.4) Write a Python program in 3D to rotate the point (1, 0, 0) through XZ Plane in anticlockwise direction (Rotation through Y axis) by an angle of 90°.

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Point to rotate point =
np.array([1, 0, 0]) #
Rotation angle in radians
angle = np.deg2rad(90)
# Rotation matrix for Y axis (anticlockwise)
rotation_matrix = np.array([
[np.cos(angle), 0, np.sin(angle)],
   [0, 1, 0],
   [-np.sin(angle), 0, np.cos(angle)]
])
# Apply rotation
rotated_point = np.dot(rotation_matrix, point)
# Create 3D plot fig
= plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Plot original point
ax.scatter(point[0], point[1], point[2], c='r', marker='o', label='Original Point')
# Plot rotated point
ax.scatter(rotated_point[0], rotated_point[1], rotated_point[2], c='b', marker='o',
label='Rotated Point') # Set plot limits ax.set_xlim([-1, 1]) ax.set_ylim([-1, 1])
ax.set_zlim([-1, 1]) # Set plot labels ax.set_xlabel('X') ax.set_ylabel('Y')
ax.set_zlabel('Z') # Add legnd ax.legend()
```

# Show the plot
plt.show()

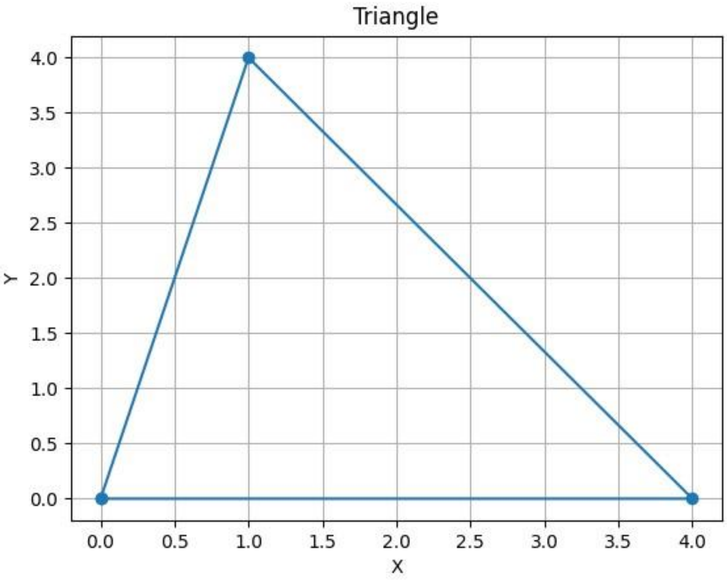
Output:



Q.5) Using python, generate triangle with vertices (0, 0), (4, 0), (1, 4), check whether the triangle is Scalene triangle. Syntax:

import matplotlib.pyplot as plt
# Vertices of the triangle
vertex1 = (0, 0) vertex2 = (4,
0) vertex3 = (1, 4) # Plot the
triangle
plt.plot(*zip(vertex1, vertex2, vertex3, vertex1), marker='o')
plt.xlabel('X') plt.ylabel('Y') plt.title('Triangle')
plt.grid(True)
plt.show()

OUTPUT:



Q.6) Write a Python program to find the area and perimeter of the ABC, where A[0, 0] B[6, 0], C[4,4].

Synatx:

import numpy as np

Define the vertices of the triangle

A = np.array([0, 0])

B = np.array([6, 0])

C = np.array([4, 4])

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: 6.0

Side BC: 4.47213595499958

Side CA: 5.656854249492381 Area:

11.999999999999998

Perimeter: 16.12899020449196

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 = 3x + 5y + 4z
subject to
2x + 3y <= 8
2y + 5z <= 10 3x +
2y + 5z <= 15 x=>0,
y=>0 Syntax:
from pulp import *
# Create a PuLP minimization problem prob
= LpProblem("LPP", LpMinimize)
Define decision variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) z =
LpVariable("z", lowBound=0) # Define
objective function prob += 3*x + 5*y + 4*z,

"Z" # Define constraints prob += 2*x + 3*y
<= 8, "Constraint1" prob += 2*y + 5*z <= 10,
"Constraint2" prob += 3*x + 2*y + 5*z <= 15,
"Constraint3"
Solve the problem prob.solve()
Print the status of the solution print("Status:",
LpStatus[prob.status])
# If the problem is solved, print the optimal solution and the optimal value of Z
if prob.status == LpStatusOptimal:
   print("Optimal Solution:")
print("x =", value(x))
print("y =", value(y))   print("z
=", value(z))
   print("Z =", value(prob.objective))

OUTPUT:
Status: Optimal
Optimal Solution:
x = 0.0 y = 0.0 z
= 0.0 Z = 0.0
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 5.
(III) Rotation about origin through an angle pi/2..
(IV)Shearing in X direction by 7/2 units Syntax:
import numpy as np
# Initial point P
P = np.array([4, -2])
# (I) Reflection through Y-axis
P_reflect_y = np.array([-P[0], P[1]])
# (II) Scaling in X-coordinate by factor 5
P_scale_x = np.array([5 * P[0], P[1]])
# (III) Rotation about origin through an angle pi/2 angle
= np.pi / 2
P_rotate = np.array([P[0] * np.cos(angle) - P[1] * np.sin(angle), P[0] * np.sin(angle) + P[1] * np.cos(angle)]) # (IV) Shearing in X-direction by 7/2 units
shearing_factor = 7 / 2
P_shear_x = np.array([P[0] + shearing_factor * P[1], P[1]])

# Print the transformed points print("Initial point P:", P)
print("Reflection through Y-axis:", P_reflect_y)
print("Scaling in X-coordinate by factor 5:", P_scale_x)
print("Rotation about origin through an angle pi/2:", P_rotate)
print("Shearing in X-direction by 7/2 units:", P_shear_x)
OUTPUT:
Initial point P: [4 -2]
Reflection through Y-axis: [-4 -2]
Scaling in X-coordinate by factor 5: [20 -2]
Rotation about origin through an angle pi/2: [2. 4.]
Shearing in X-direction by 7/2 units: [-3. -2.]

Q.10) Find the combined transformation of the line segment between the point A[7, -2] & B[6, 2] by using Python program for the following sequence of transformation:-

(I) Rotation about origin through an angle pi/3.
(II) Scaling in X-Coordinate by 7 units.
(III) Uniform scaling by -4 units (IV) Reflection through the line X - axis

Syntax:

```
import numpy as np #
Define the point P
P = np.array([4, -2])
# Define the transformation functions def
rotate_about_origin(point, angle):    #
Rotation about origin through an angle
cos_theta = np.cos(angle)    sin_theta =
np.sin(angle)
x = point[0]
y = point[1]
   x_rotated = x * cos_theta - y * sin_theta
y_rotated = x * sin_theta + y * cos_theta
return np.array([x_rotated, y_rotated]) def
scale_x(point, factor):    # Scaling in X-
coordinate    x_scaled = point[0] * factor
y_scaled = point[1]    return
np.array([x_scaled, y_scaled]) def
uniform_scale(point, factor):
   # Uniform scaling    x_scaled = point[0]
* factor    y_scaled = point[1] * factor
return np.array([x_scaled, y_scaled]) def
reflect_x_axis(point):    # Reflection
```

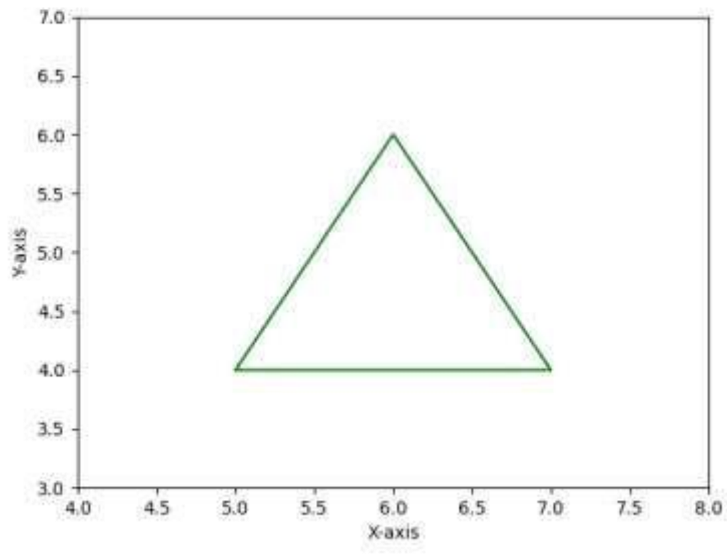
through X-axis   x_reflected = point[0]
y_reflected = -point[1]   return
np.array([x_reflected, y_reflected]) # Apply
the transformations to the point P angle =
np.pi / 3
P_rotated = rotate_about_origin(P, angle)
P_scaled_x = scale_x(P_rotated, 7)
P_uniform_scaled = uniform_scale(P_scaled_x, -4)
P_reflected_x_axis = reflect_x_axis(P_uniform_scaled)
# Print the transformed points print("Point P: ", P)
print("After Rotation: ", P_rotated) print("After Scaling in X-Coordinate: ", P_scaled_x) print("After Uniform Scaling: ", P_uniform_scaled) print("After Reflection through X-axis: ", P_reflected_x_axis)

OUTPUT:

```
Point P:  [ 4 -2]
After Rotation:  [3.73205081 2.46410162]
After Scaling in X-Coordinate:  [26.12435565  2.46410162]
After Uniform Scaling:  [-104.49742261   -9.85640646]
After Reflection through X-axis:  [-104.49742261    9.85640646]
```

SLIP-9

Q.1) Write n python program to Plot 2D X-axis and Y-axis black color and in the same diagram plot green triangle with vertices [5,4],[7,4],[6,6] Syntax: import matplotlib.pyplot as plt # Define the vertices of the triangle triangle_vertices = [[5, 4], [7, 4], [6, 6]]
# Extract the x and y coordinates of the triangle vertices
x = [vertex[0] for vertex in triangle_vertices] y = [vertex[1] for vertex in triangle_vertices] # Plot the X-axis and Y-axis in black color plt.axhline(0, color='black') plt.axvline(0, color='black') # Plot the triangle with green color plt.plot(x + [x[0]], y + [y[0]], 'g') # Set the plot limits and labels plt.xlim(4, 8) plt.ylim(3, 7) plt.xlabel('X-axis') plt.ylabel('Y-axis') # Show the plot



plt.show()

OUTPUT:

Q.2) Write n program in python to rotate the point. through YZ-plane in anticlockwise direction. (Rotation through Y-axis by an angle of 90°.) Syntax:

import math import numpy as np import

matplotlib.pyplot as plt from

mpl_toolkits.mplot3d import Axes3D

Function to rotate a point (x, y, z) through YZ-plane by an angle of 90° def

rotate_yz_plane(point):

x, y, z = point new_y = y * math.cos(math.radians(90)) - z *

math.sin(math.radians(90)) new_z = y * math.sin(math.radians(90)) + z *

math.cos(math.radians(90)) return [x, new_y, new_z]

Point to rotate point

= [1, 2, 3]

Call the rotation function rotated_point

= rotate_yz_plane(point)

Original point coordinates x_original,

y_original, z_original = point # Rotated

point coordinates x_rotated, y_rotated,

z_rotated = rotated_point

Create a 3D plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

Plot original point as a red dot ax.scatter(x_original, y_original, z_original,

color='red', label='Original Point')

Plot rotated point as a blue dot ax.scatter(x_rotated, y_rotated, z_rotated,

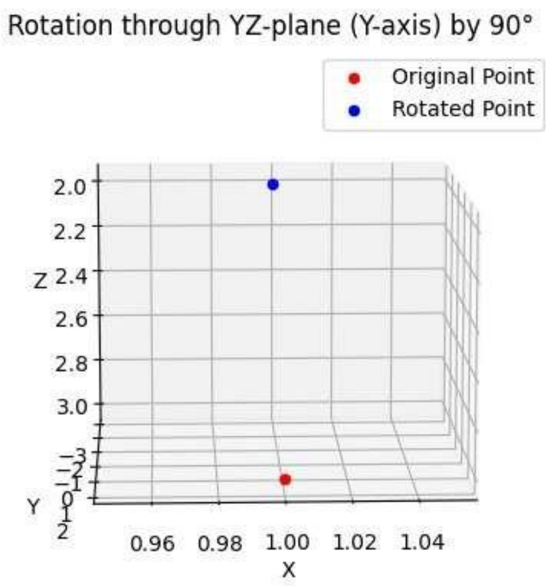
color='blue', label='Rotated Point') ax.set_xlabel('X') ax.set_ylabel('Y')

ax.set_zlabel('Z')

ax.set_title('Rotation through YZ-plane (Y-axis) by 90°')
ax.legend()

plt.show()

OUTPUT:



Q.3) Using Python plot the graph of function f(x) = cos(x) on the interval (0, 2*pi).

Syntax:
import numpy as np import

matplotlib.pyplot as plt

Generate x values from 0 to 2*pi with a step of 0.01 x

= np.arange(0, 2*np.pi, 0.01)

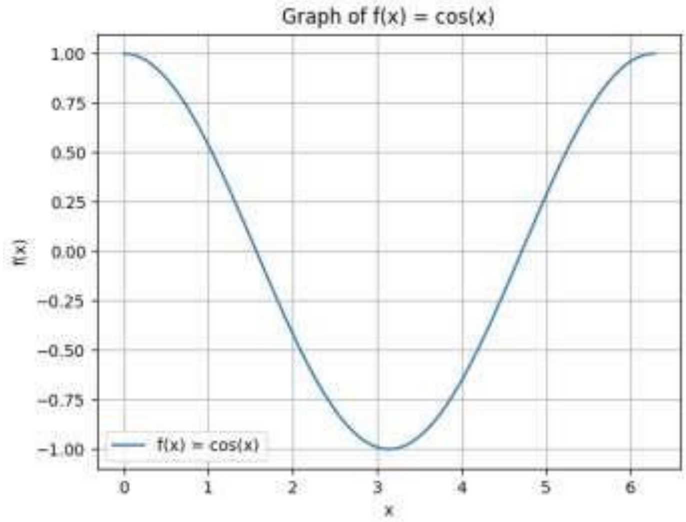
Compute the corresponding y values for f(x) = cos(x)

y = np.cos(x) # Create a plot plt.plot(x, y, label='f(x) =

cos(x)') plt.xlabel('x') plt.ylabel('f(x)') plt.title('Graph

of f(x) = cos(x)') plt.legend() plt.grid(True) plt.show()

OUTPUT:



Q.4) Write a python program to rotate the ray by 90° having starting point (1,0) and (2,-1)

Syntax:

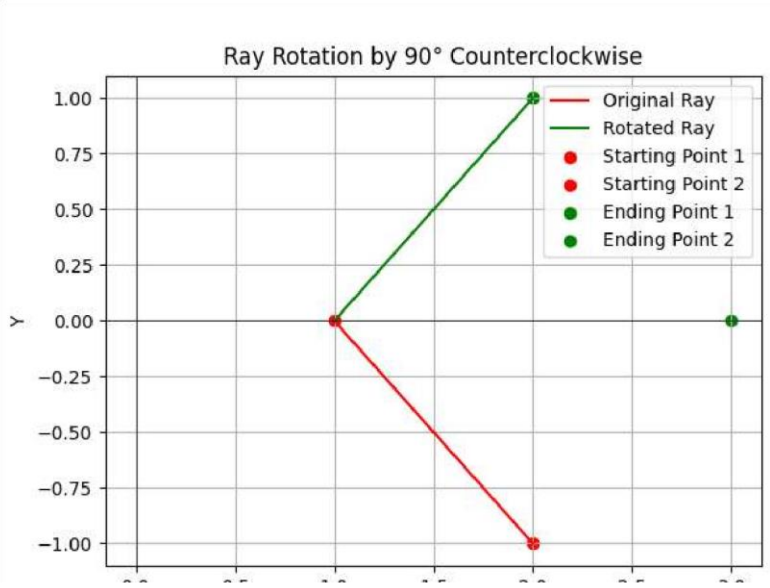
```
import numpy as np
import matplotlib.pyplot as plt #
Define the starting points of the ray
start_point_1 = np.array([1, 0])
start_point_2 = np.array([2, -1]) #
Compute the direction vector of the
ray direction_vector = start_point_2
- start_point_1 # Perform the rotation
by 90° counterclockwise
rotation_matrix = np.array([[0, -1],
[1, 0]])
rotated_direction_vector = np.dot(rotation_matrix, direction_vector)
# Compute the ending point of the rotated ray end_point_1
= start_point_1 + rotated_direction_vector end_point_2 =
start_point_2 + rotated_direction_vector
# Plot the original and rotated rays
plt.plot([start_point_1[0], start_point_2[0]], [start_point_1[1], start_point_2[1]],
'r', label='Original Ray')
plt.plot([start_point_1[0], end_point_1[0]], [start_point_1[1], end_point_1[1]],
'g', label='Rotated Ray')
plt.scatter(start_point_1[0], start_point_1[1], c='r', marker='o', label='Starting
Point 1')
plt.scatter(start_point_2[0], start_point_2[1], c='r', marker='o', label='Starting
Point 2')
```

```
plt.scatter(end_point_1[0], end_point_1[1], c='g', marker='o', label='Ending
Point 1')
plt.scatter(end_point_2[0], end_point_2[1], c='g', marker='o', label='Ending
Point 2') plt.axhline(0, color='k', linewidth=0.5) plt.axvline(0, color='k',
linewidth=0.5)
plt.xlabel('X')
plt.ylabel('Y') plt.legend()
plt.title('Ray Rotation by 90° Counterclockwise')
plt.grid(True) plt.show()
```

Output:

Q.5) Using sympy declare the points A(0, 7), B(5, 2). Declare the line segment passing through them. Find the length and midpoint of the line segment passing through points A and B. Syntax:

```
from sympy import Point, Line
# Declare the points A and B
A = Point(0, 7)
B = Point(5, 2)
# Declare the line passing through points A and B line_AB
= Line(A, B)
# Calculate the length of the line segment AB length_AB
= A.distance(B)
# Calculate the midpoint of the line segment AB midpoint_AB
= ((A[0] + B[0]) / 2, (A[1] + B[1]) / 2)
# Print the results print("Point
A: {}".format(A)) print("Point
```



B: {}".format(B))
print("Line segment AB: {}".format(line_AB)) print("Length of line segment AB: {}".format(length_AB)) print("Midpoint of line segment AB: {}".format(midpoint_AB))

OUTPUT:
Point A: Point2D(0, 7)
Point B: Point2D(5, 2)
Line segment AB: Line2D(Point2D(0, 7), Point2D(5, 2))
Length of line segment AB: 5*sqrt(2)
Midpoint of line segment AB: (5/2, 9/2)

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

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 – 5z – 4w >= 20
-3x – 2y + 2z + w <= 10 -8x
– 3y + 3z + 2w <= 20 x=>0,
y=>0,z=> 0,w>= Syntax:
from pulp import *
# Define the decision variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) z =
LpVariable("z", lowBound=0) w =
LpVariable("w", lowBound=0) #
Define the objective function
objective = 4 * x + y + 3 * z + 5 * w
# Define the constraints constraint1 = 4 * x + 6 *
y - 5 * z - 4 * w >= 20 constraint2 = -3 * x - 2 *

y + 2 * z + w <= 10 constraint3 = -8 * x - 3 * y
+ 3 * z + 2 * w <= 20
# Create the LP problem
problem = LpProblem("Linear_Programming_Problem", LpMinimize)
# Add the objective function and constraints to the problem
problem += objective problem += constraint1 problem +=
constraint2 problem += constraint3 # Solve the LP problem
status = problem.solve() # Check the status of the solution
if status == LpStatusOptimal:
  # Get the optimal values of the decision variables
opt_x = value(x)   opt_y = value(y)   opt_z =
value(z)   opt_w = value(w)
  # Get the optimal value of the objective function   opt_z =
value(objective)   # Print the optimal solution   print("Optimal
Solution:")   print("x = {}".format(opt_x))   print("y =
{}".format(opt_y))   print("z = {}".format(opt_z))   print("w =
{}".format(opt_w))   print("Optimal value of the objective
function: {}".format(opt_z)) else:
  print("No optimal solution found.")
OUTPUT:
Optimal Solution:
x = 0.0 y =
3.3333333 z =
3.3333333 w =
0.0
Optimal value of the objective function: 3.3333333

Q.9) Apply Python. Program in each of the following transformation on the point P[-2,4]
(I) Shearing in Y direction by 7 units.
(II)Scaling in X and Y direction by 7/2 and 7 unit respectively.
(III) Shearing in X and Y direction by 4 and 7 unit respectively.
(IV)Rotation about origin by an angle 60^{o} Syntax:
import numpy as np
# Define the original point P
P = np.array([-2, 4])
# Transformation 1: Shearing in Y direction by 7 units
shearing_Y = np.array([[1, 0], [7, 1]]) P_transformed1
= np.dot(shearing_Y, P)

# Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively
scaling_XY = np.array([[7/2, 0], [0, 7]]) P_transformed2 = np.dot(scaling_XY, P)
# Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively
shearing_XY = np.array([[1, 4], [7, 1]]) P_transformed3 = np.dot(shearing_XY, P)
# Transformation 4: Rotation about origin by an angle of 60 degrees angle = np.radians(60)
rotation = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
P_transformed4 = np.dot(rotation, P) # Print the transformed points
print("Original Point P: {}".format(P)) print("Transformation 1: Shearing in Y direction by 7 units: {}".format(P_transformed1))
print("Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively: {}".format(P_transformed2))
print("Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively: {}".format(P_transformed3)) print("Transformation 4: Rotation about origin by an angle of 60 degrees: {}".format(P_transformed4))

OUTPUT:
Original Point P: [-2 4]
Transformation 1: Shearing in Y direction by 7 units: [-2 -10]
Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively: [-7. 28.]
Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively: [14 -10]
Transformation 4: Rotation about origin by an angle of 60 degrees: [-4.46410162
0.26794919]

Q.10) Find the combined transformation of the line segment between the point A[5,3] & B[1, 4] by using Python program for the following sequence of transformation:-

(I) Rotate about origin through an angle pi/3.
(II) Uniform scaling by -.5 units
(III) scaling in Y – axis by 5 units
(IV) Shearing in X and Y direction by 3 and 4 nits respectively.

Syntax: import
numpy as np
# Define the original points A and B
A = np.array([5, 3])
B = np.array([1, 4])
# Transformation 1: Rotate about origin through an angle of pi/3 angle
= np.pi / 3
rotation = np.array([[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]])
A_transformed1 = np.dot(rotation, A)
B_transformed1 = np.dot(rotation, B)
# Transformation 2: Uniform scaling by -0.5 units scaling_uniform
= np.array([[-0.5, 0], [0, -0.5]])
A_transformed2 = np.dot(scaling_uniform, A_transformed1)
B_transformed2 = np.dot(scaling_uniform, B_transformed1)
# Transformation 3: Scaling in Y-axis by 5 units scaling_Y
= np.array([[1, 0], [0, 5]])
A_transformed3 = np.dot(scaling_Y, A_transformed2)
B_transformed3 = np.dot(scaling_Y, B_transformed2)
# Transformation 4: Shearing in X and Y direction by 3 and 4 units respectively
shearing_XY = np.array([[1, 3], [4, 1]])
A_transformed4 = np.dot(shearing_XY, A_transformed3)
B_transformed4 = np.dot(shearing_XY, B_transformed3)
# Print the transformed points print("Original
Point A: {}".format(A)) print("Original Point
B: {}".format(B))
print("Transformation 1: Rotate about origin through an angle of pi/3")
print("A_transformed1: {}".format(A_transformed1))
print("B_transformed1: {}".format(B_transformed1)) print("Transformation
2: Uniform scaling by -0.5 units") print("A_transformed2:
{}".format(A_transformed2)) print("B_transformed2:
{}".format(B_transformed2)) print("Transformation 3: Scaling in Y-axis by
5 units") print("A_transformed3: {}".format(A_transformed3))
print("B_transformed3: {}".format(B_transformed3))
print("Transformation 4: Shearing in X and Y direction by 3 and 4 units respectively") print("A_transformed4: {}".format(A_transformed4))
print("B_transformed4: {}".format(B_transformed4))

OUTPUT:
Original Point P: [-2 4]

Transformation 1: Shearing in Y direction by 7 units: [-2 -10]
Transformation 2: Scaling in X and Y direction by 7/2 and 7 units respectively:
[-7. 28.]
Transformation 3: Shearing in X and Y direction by 4 and 7 units respectively: [
14 -10]
Transformation 4: Rotation about origin by an angle of 60 degrees: [-
4.46410162 0.26794919]
PS E:\Python 2nd Sem Practical> python -u "e:\Python 2nd Sem
Practical\tempCodeRunnerFile.py"
Original Point A: [5 3]
Original Point B: [1 4]
Transformation 1: Rotate about origin through an angle of pi/3
A_transformed1: [-0.09807621 5.83012702]
B_transformed1: [-2.96410162 2.8660254]
Transformation 2: Uniform scaling by -0.5 units
A_transformed2: [0.04903811 -2.91506351]
B_transformed2: [1.48205081 -1.4330127]
Transformation 3: Scaling in Y-axis by 5 units
A_transformed3: [0.04903811 -14.57531755]
B_transformed3: [1.48205081 -7.16506351]
Transformation 4: Shearing in X and Y direction by 3 and 4 units respectively
A_transformed4: [-43.67691454 -14.37916512]
B_transformed4: [-20.01313972 -1.23686028]

SLIP-10

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')

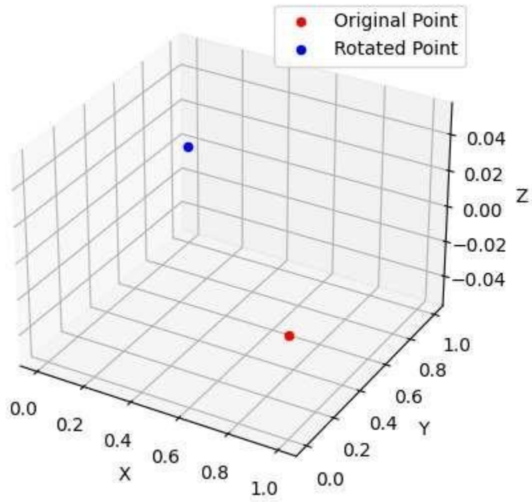
Plot the rotated point ax.scatter(point_rotated[0], point_rotated[1],

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

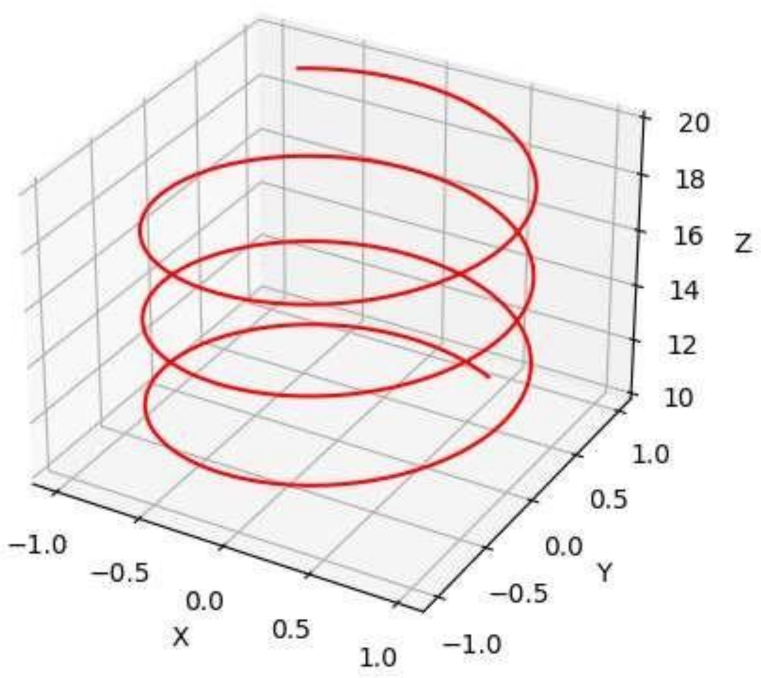
mpl_toolkits.mplot3d import Axes3D

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:

3D Line Graph: (cos(2x), sin(2x), x)



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 =

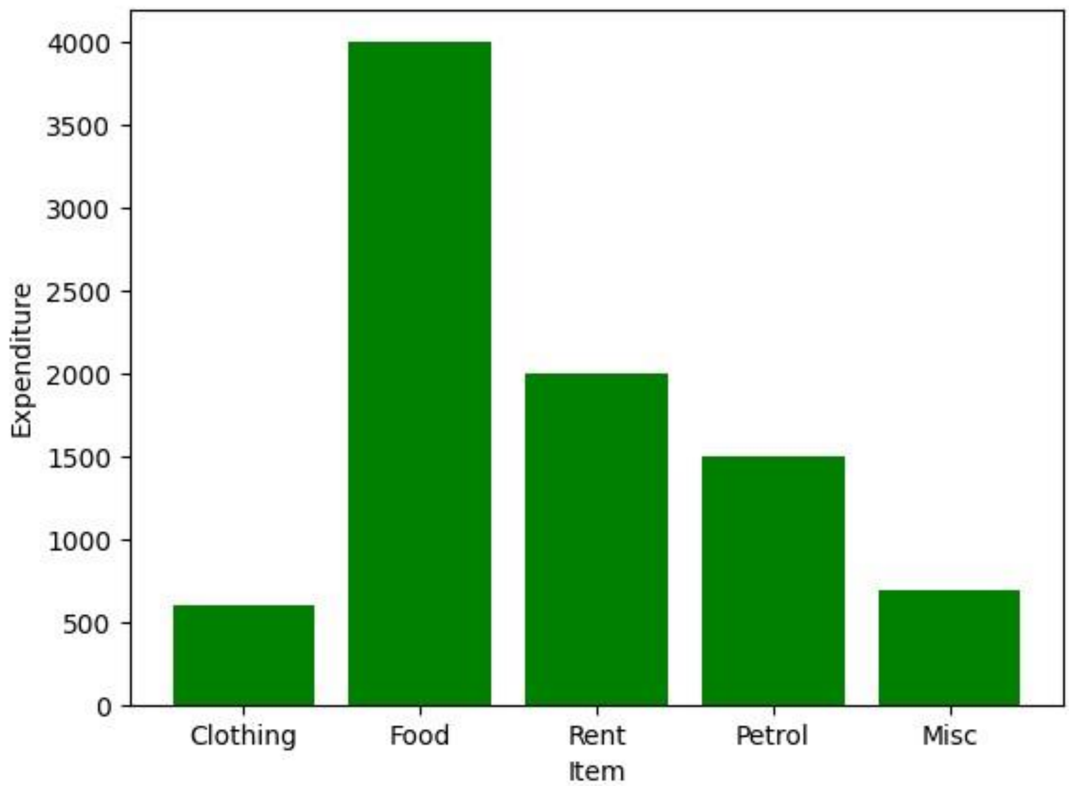
[600,4000,200,1500,]

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
```

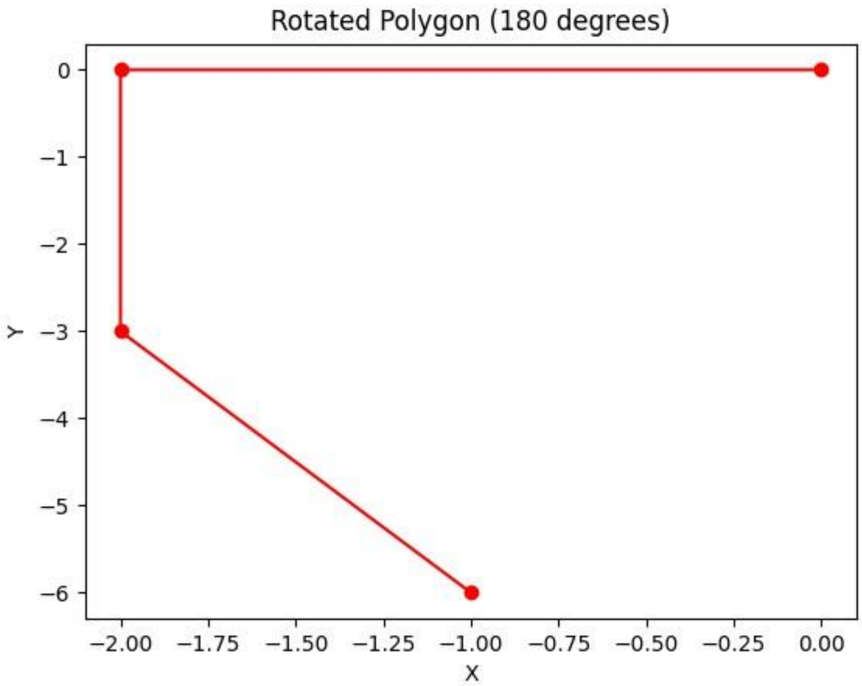
Point B: ", B_rotated) print("Rotated
Point C: ", C_rotated) Output:

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')
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: 7.500000000000036

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 45° 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]

SLIP-11

Q.1) Write a python program to plot 3D axes with labels as X – axis and Y – axis
And z axis and also plot following point. With given coordinate in the same graph (70,-25,15) as a diamond in black color Syntax: import matplotlib.pyplot as plt import numpy as np # Create a 3D plot figure fig = plt.figure() ax = fig.add_subplot(111, projection='3d')

Define the point coordinates

x = 70 y = -25 z = 15

Plot the point as a diamond shape in black color ax.plot([x],

[y], [z], marker='D', color='black')

Set labels for the axes

ax.set_xlabel('X-axis')

ax.set_ylabel('Y-axis')

ax.set_zlabel('Z-axis') #

Set limits for the axes

ax.set_xlim([0, 100])

ax.set_ylim([-30, 30])

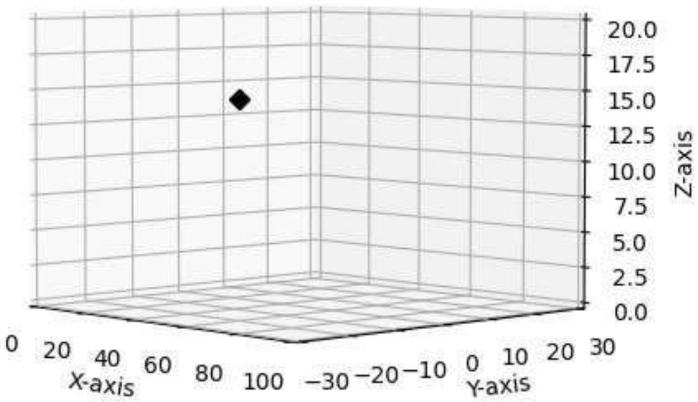
ax.set_zlim([0, 20]) #

Display the plot

plt.show()

plt.show()

OUTPUT:



Q.2) Plot the graph of y = e**-x in [-5,5] with red dashed line with Upward pointing Triangle Syntax: import matplotlib.pyplot as plt import numpy as np

Generate x values in the range [-5,5]

x = np.linspace(-5, 5, 100) # Compute

y values using y = e**-x y = np.exp(-

x)

Create a figure and axis fig,

ax = plt.subplots()

Plot the graph with red dashed line and upward pointing triangles as markers

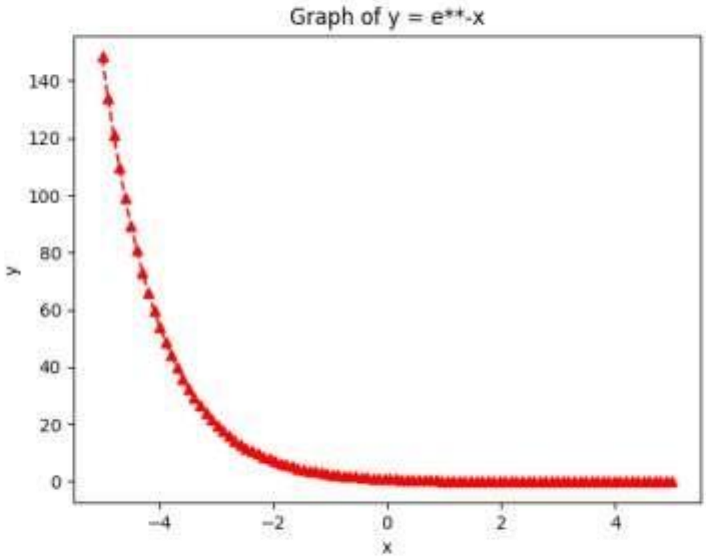
ax.plot(x, y, 'r--', marker='^') # Set labels for the x-axis and y-axis

ax.set_xlabel('x') ax.set_ylabel('y') # Set title for the plot ax.set_title('Graph of y = e**-x')

Display the plot

plt.show() plt.show()

OUTPUT:



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

Subject	Maths	Science	English	Marathi	Hindi
Percentage of passing	68	90	70	85	91

Syntax:

import matplotlib.pyplot as plt

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

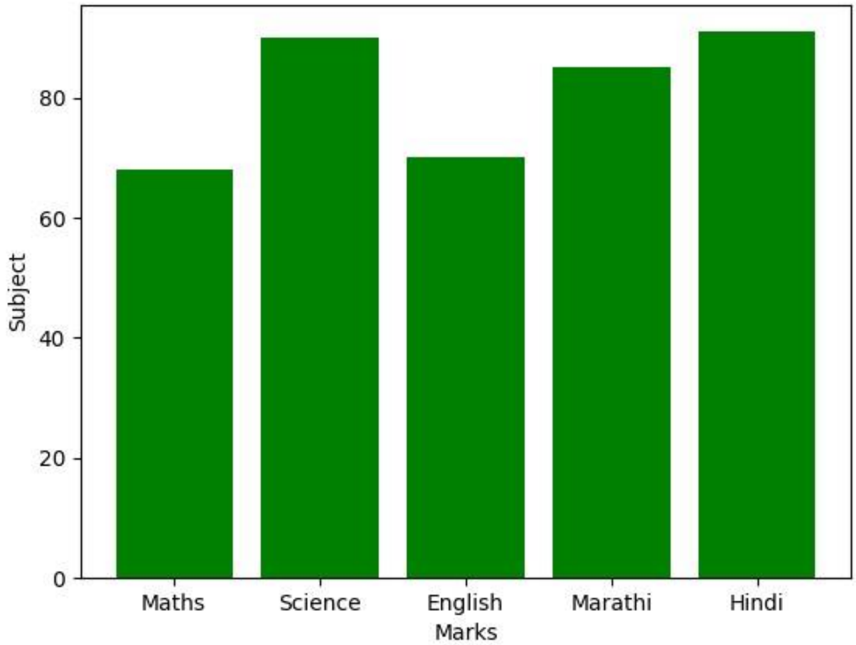
[68,90,70,85,91]

tick_label=['Maths','Science','English','Marathi',’Hindi'] 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 reflect the ABC through the line y = 3 where A(1, 0), B(2, -2), C(-1, 2).

Syntax:

import numpy as np

Define the reflection line y = 3 reflection_line

= 3

Define the points A, B, and C

A = np.array([1, 0])

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

C = np.array([-1, 2])

Compute the reflected points A', B', and C'

Ap = np.array([A[0], 2 * reflection_line - A[1]])

Bp = np.array([B[0], 2 * reflection_line - B[1]])

Cp = np.array([C[0], 2 * reflection_line - C[1]])

Print the original points and reflected points

print("Original Points:") print("A: ", A)

print("B: ", B) print("C: ", C) print("Reflected

Points:") print("A':", Ap) print("B':", Bp)

print("C':", Cp) Output:

Original Points:

A: [1 0]

B: [2 -2]

C: [-1 2]

Reflected Points:

A': [1 6]

B': [2 8]

C': [-1 4]

Q.6) Write a python program to draw a polygon with 6 sides and radius 1 centered at (1,2) and find its area and perimeter Synatx: import math import matplotlib.pyplot as plt import numpy as np

Define the center of the hexagon

center = np.array([1, 2]) # Define the radius of the hexagon radius = 1

Calculate the coordinates of the vertices of the hexagon

angle_deg = np.linspace(0, 360, 7)[:-1] angle_rad = np.deg2rad(angle_deg) x_coords = center[0] + radius * np.cos(angle_rad) y_coords = center[1] + radius * np.sin(angle_rad)

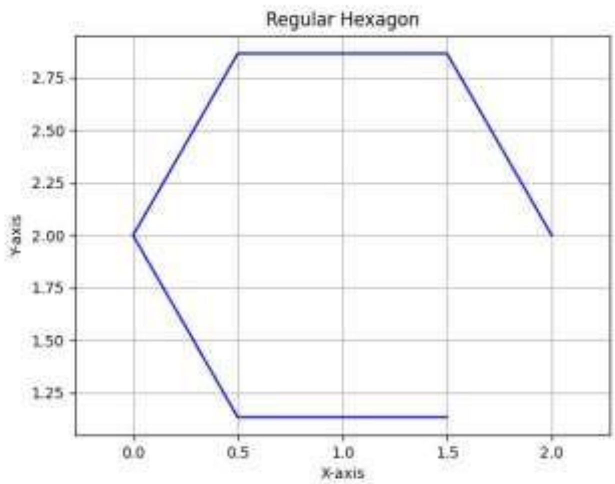
Plot the hexagon plt.plot(x_coords, y_coords, 'b-') plt.xlabel('X-axis') plt.ylabel('Y-axis') plt.title('Regular Hexagon') plt.axis('equal') plt.grid(True) plt.show()

Calculate the area of the hexagon

side_length = 2 * radius * np.sin(np.pi / 3)

area = (3 * np.sqrt(3) * side_length ** 2) / 2

Calculate the perimeter of the hexagon perimeter = 6 * side_length



Print the area and perimeter print("Area of

the hexagon:", area) print("Perimeter of the

hexagon:", perimeter)

OUTPUT:

Area of the hexagon: 7.794228634059947

Perimeter of the hexagon: 10.392304845413264

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

Max Z = x + y
Subjected to
x >= 6 y >= 6
x+y >= 11
x > 0 , y > 0

Syntax:

from pulp import *

Create a maximization problem prob =

LpProblem("Maximization Problem", LpMaximize)

Define 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 >= 6,

"Constraint 1" prob += y >= 6,

"Constraint 2" prob += x + y >= 11,

"Constraint 3"

Solve the problem

prob.solve()

Print the status of the problem

print("Status:",

LpStatus[prob.status]) # Print the

optimal solution print("Optimal

Solution:") print("x =", value(x))

print("y =", value(y))

Print the optimal objective value

print("Z =", value(prob.objective))

OUTPUT:
Status: Unbounded Optimal
Solution:
x = 0.0 y
= 0.0 Z
= 0.0

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 = 3x+5y + 4z subject
to
2x+ 3y <= 8
2y + 5z <= 10 3x
+ 2y + 4z <= 15
x>=0,y>=0,z>=0 Syntax:
from pulp import *
# Create a minimization problem
prob = LpProblem("Minimization Problem", LpMinimize)
# Define 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 +
5*y + 4*z, "Z" # Define the constraints prob
+= 2*x + 3*y <= 8, "Constraint 1" prob +=
2*y + 5*z <= 10, "Constraint 2" prob += 3*x +
2*y + 4*z <= 15, "Constraint 3"
# Solve the problem prob.solve()
# Print the status of the problem print("Status:",
LpStatus[prob.status])
# Print the optimal solution
print("Optimal Solution:")
print("x =", value(x))
print("y =", value(y)) print("z
=", value(z))
# Print the optimal objective value
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) Reflection through x – axis
(II) Scaling in X – coordinate by 6 factor
(III) Shearing in x direction by 4 unit
(IV) Rotation About origin through an angle 30 Syntax:

```
import math #
Initial point
point = (-2, 4)
x, y = point
# Transformation 1: Reflection through x-axis point_reflection_x_axis
= (x, -y)
# Transformation 2: Scaling in X-coordinate by 6 factor scale_factor
= 6
point_scaling_x = (x * scale_factor, y)
# Transformation 3: Shearing in x-direction by 4 units shear_factor
= 4
point_shearing_x = (x + shear_factor * y, y)
# Transformation 4: Rotation about origin through an angle of 30 degrees angle
= 30
```

angle_rad = math.radians(angle)
point_rotation = (x * math.cos(angle_rad) - y * math.sin(angle_rad), x *
math.sin(angle_rad) + y * math.cos(angle_rad))
# Print the transformed points
print("Transformation 1: Reflection through x-axis")
print("x =", point_reflection_x_axis[0]) print("y =",
point_reflection_x_axis[1])
print("\nTransformation 2: Scaling in X-coordinate by 6 factor")
print("x =", point_scaling_x[0]) print("y =", point_scaling_x[1])
print("\nTransformation 3: Shearing in x-direction by 4 units")
print("x =", point_shearing_x[0]) print("y =",
point_shearing_x[1])
print("\nTransformation 4: Rotation about origin through an angle of 30 degrees")
print("x =", point_rotation[0]) print("y =", point_rotation[1])

OUTPUT:

Transformation 1: Reflection through x-axis
x = -2
y = -4
Transformation 2: Scaling in X-coordinate by 6 factor
x = -12
y = 4
Transformation 3: Shearing in x-direction by 4 units
x = 14
y = 4
Transformation 4: Rotation about origin through an angle of 30 degrees
x = -3.732050807568877 y = 2.464101615137755

Q.10) Find the combined transformation between the point by using Python program for the following sequence of transformation:- (I) Rotation about origin through an angle pi/2.

(II) Uniform scaling by -6.4units
(III) Scaling in x & y-Coordinate by 3 &5 units respectively.
(IV) Shearing in X – Direction by 6 unit.

Syntax:

import math #
Initial point
point = (3, 5)
x, y = point
Transformation 1: Rotation about origin through an angle of pi/2 angle_rad
= math.pi/2

point_rotation = (x * math.cos(angle_rad) - y * math.sin(angle_rad), x * math.sin(angle_rad) + y * math.cos(angle_rad)) # Transformation 2: Uniform scaling by -6.4 units scale_factor_uniform = -6.4 point_uniform_scaling = (x * scale_factor_uniform, y * scale_factor_uniform) # Transformation 3: Scaling in x & y-coordinate by 3 & 5 units respectively scale_factor_x = 3 scale_factor_y = 5 point_scaling = (x * scale_factor_x, y * scale_factor_y) # Transformation 4: Shearing in X-Direction by 6 units shear_factor_x = 6
point_shearing_x = (x + shear_factor_x * y, y)
# Combined Transformation
point_combined_transformation = point_rotation
point_combined_transformation = (point_combined_transformation[0] * scale_factor_uniform, point_combined_transformation[1] * scale_factor_uniform)
point_combined_transformation = (point_combined_transformation[0] * scale_factor_x, point_combined_transformation[1] * scale_factor_y)
point_combined_transformation = (point_combined_transformation[0] + shear_factor_x * point_combined_transformation[1], point_combined_transformation[1]) # Print the transformed points
print("Transformation 1: Rotation about origin through an angle of pi/2")
print("x =", point_rotation[0]) print("y =", point_rotation[1])
print("\nTransformation 2: Uniform scaling by -6.4 units")
print("x =", point_uniform_scaling[0]) print("y =",
point_uniform_scaling[1])
print("\nTransformation 3: Scaling in x & y-coordinate by 3 & 5 units respectively") print("x =", point_scaling[0]) print("y =", point_scaling[1])
print("\nTransformation 4: Shearing in X-Direction by 6 units")
print("x =", point_shearing_x[0]) print("y =",
point_shearing_x[1]) print("\nCombined Transformation:")
print("x =", point_combined_transformation[0]) print("y =",
point_combined_transformation[1])
OUTPUT:
Transformation 1: Rotation about origin through an angle of pi/2 x
= -5.0
y = 3.0000000000000004
Transformation 2: Uniform scaling by -6.4 units
x = -19.200000000000003
y = -32.0
Transformation 3: Scaling in x & y-coordinate by 3 & 5 units respectively
x = 9 y = 25

Transformation 4: Shearing in X-Direction by 6 units
x = 33
y = 5
Combined Transformation:
x = -480.0000000000001 y
= -96.00000000000001

SLIP-12

Q.1) write a python program to plot the graph of y = x**3 + 10*x - 5, for x belongs [-10, 10] in red color.

Syntax: import numpy as np

import matplotlib.pyplot as plt

Define the equation y = x**3 + 10*x - 5 def

equation(x):

    return x**3 + 10*x - 5

Generate x values in the range [-10, 10] x

= np.linspace(-10, 10, 500)

Evaluate the y values using the equation

y = equation(x) # Create the plot

plt.plot(x, y, color='red')

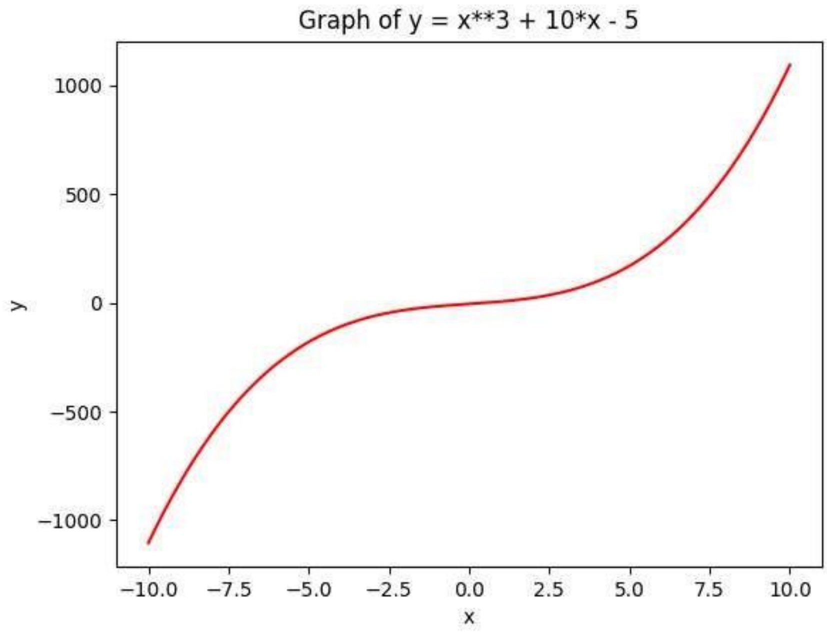
Set the plot title and axis labels

plt.title("Graph of y = x**3 + 10*x - 5")

plt.xlabel("x") plt.ylabel("y") # Show

the plot plt.show()

OUTPUT:



Q.2) write a python program in 3D to rotate the point (1, 0, 0) through XZ-plane in clockwise direction (rotation through Y- axis by an angle of 90°).

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
# Define the point to rotate point
= np.array([1, 0, 0])
# Define the rotation angle in radians
theta = np.radians(90) # Create the 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')
# Perform the rotation rotated_point =
np.dot(np.array([[np.cos(theta), 0, np.sin(theta)],
[0, 1, 0],
                    [-np.sin(theta), 0, np.cos(theta)]]), point)
```

Plot the rotated point ax.scatter(rotated_point[0], rotated_point[1],

rotated_point[2], color='blue', label='Rotated Point')

Set the plot title and axis labels

ax.set_title('Rotation of Point in 3D')

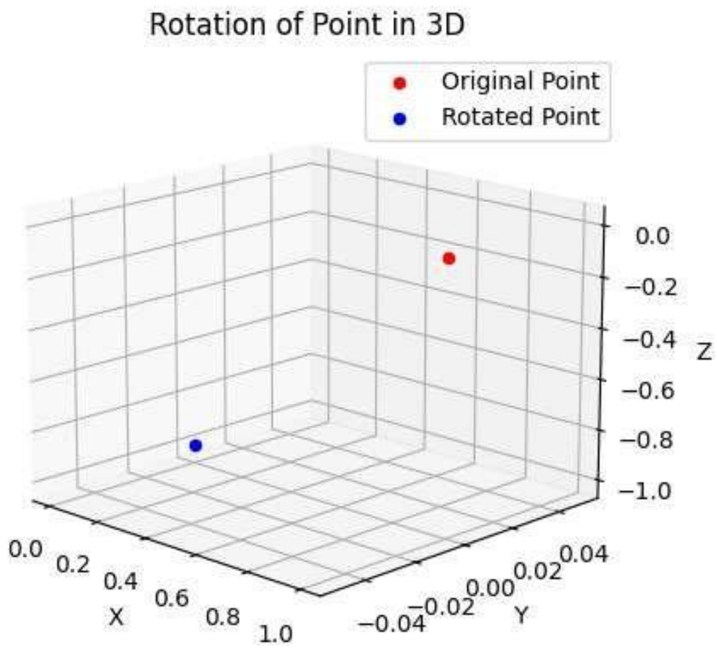
ax.set_xlabel('X') ax.set_ylabel('Y')

ax.set_zlabel('Z') # Add a legend

ax.legend() # Show the plot

plt.show()

OUTPUT:



Q.3) Using Python plot the graph of function f(x) = x**2 on the interval (-2,2).
Syntax:

import numpy as np import

matplotlib.pyplot as plt #

Define the function f(x) = x^2

def f(x):

    return x**2

Generate x values in the range (-2,2) with a step of 0.1 x

= np.arange(-2, 2, 0.1)

Calculate y values using the function f(x)

y = f(x) # Create the plot plt.plot(x, y,

label='f(x) = x^2') # Set the plot title and

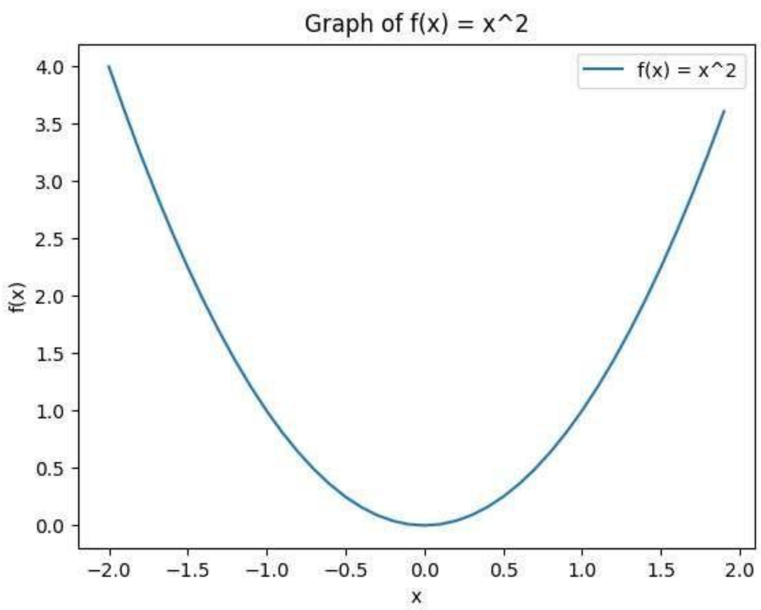
axis labels plt.title('Graph of f(x) = x^2')

plt.xlabel('x') plt.ylabel('f(x)') # Add a

legend plt.legend() # Show the plot

plt.show()

OUTPUT:



Q.4) Write a python program to rotate the segment by 180° having endpoints (1,0) and (2,-1) Syntax:

import math
# Define the endpoints of the line segment
x1, y1 = 1, 0 x2, y2 = 2, -1 # Perform the
rotation x1_rotated = -x1 y1_rotated = -y1
x2_rotated = -x2 y2_rotated = -y2
# Print the original and rotated endpoints print("Original Endpoint
1: ({}, {})".format(x1, y1)) print("Original Endpoint 2: ({},
{})".format(x2, y2)) print("Rotated Endpoint 1: ({},
{})".format(x1_rotated, y1_rotated)) print("Rotated Endpoint 2: ({},
{})".format(x2_rotated, y2_rotated))

Output:
Original Endpoint 1: (1, 0)
Original Endpoint 2: (2, -1)
Rotated Endpoint 1: (-1, 0)
Rotated Endpoint 2: (-2, 1)

Q.5) Write a python program to draw a polygon with 8 sides and radius 5 centered at origin and find its area and perimeter Syntax: import matplotlib.pyplot as plt import numpy as np

Number of sides in the polygon

num_sides = 8 # Radius of the

polygon radius = 5

# Calculate the angle between each pair of vertices
angle = 2 * np.pi / num_sides

Generate the x and y coordinates of the vertices x =

[radius * np.cos(i * angle) for i in range(num_sides)] y =

[radius * np.sin(i * angle) for i in range(num_sides)] #

Add the first vertex again to close the polygon

x.append(x[0])

y.append(y[0]) # Plot the polygon plt.plot(x, y, 'bo-') # 'bo-' specifies blue color, circle marker, and solid line # Set the aspect ratio to 'equal' to ensure the polygon is displayed as a regular shape plt.axis('equal')

Set the labels for the axes

plt.xlabel('X') plt.ylabel('Y')

Set the title of the plot plt.title('Octagon with

Radius 5 Centered at Origin')

Show the plot plt.show()

Calculate the area of the polygon area = 0.5 *

num_sides * radius ** 2 * np.sin(angle)

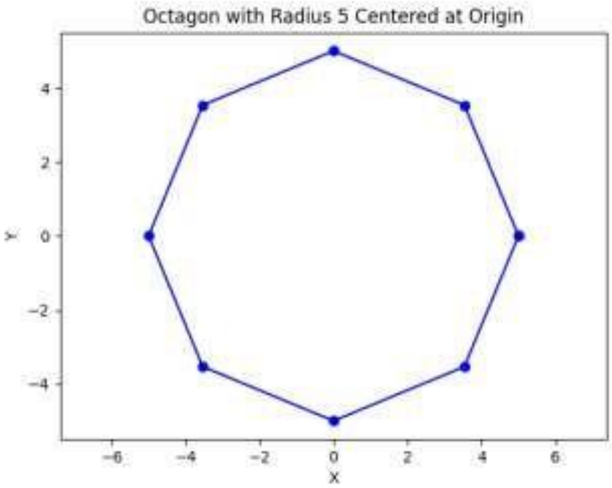
Calculate the perimeter of the polygon

perimeter = num_sides * radius # Print the calculated area and perimeter print('Area

of the octagon:', area) print('Perimeter of

the octagon:', perimeter) Output:

Area of the octagon: 70.71067811865476
Perimeter of the octagon: 40



Q.6) Write a python program to find the area and perimeter of the XYZ, where X(l, 2), Y(2, -2), Z(-1,2).

Synatx:

import math #

Input coordinates

X = [1, 2]

Y = [2, -2]

Z = [-1, 2]

Calculate distances between points def

distance(p1, p2):

    return math.sqrt((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2)

Calculate lengths of sides

XY = distance(X, Y)

YZ = distance(Y, Z)

XZ = distance(X, Z) #

Calculate perimeter

perimeter = XY + YZ + XZ

Calculate area using Heron's formula s =

perimeter / 2 area = math.sqrt(s * (s - XY) * (s -

YZ) * (s - XZ))

Print results print("Length

of XY: ", XY) print("Length

of YZ: ", YZ) print("Length

of XZ: ", XZ)

print("Perimeter: ",

perimeter) print("Area: ",

area))

OUTPUT:

Length of XY: 4.123105625617661

Length of YZ: 5.0

Length of XZ: 2.0

Perimeter: 11.123105625617661

Area: 4.000000000000003

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

Max Z = 3.5x +2y
Subjected to x + y
>= 5 x >= 4 y <=
2 x > 0 , y > 0
Syntax:

from pulp import * #
Create the problem
prob = LpProblem("Linear Programming Problem", LpMaximize)
# Define the decision variables x
= LpVariable("x", lowBound=0) y

= LpVariable("y", lowBound=0) #
Define the objective function
objective = 3.5 * x + 2 * y prob
+= objective # Define the
constraints prob += x + y >= 5
prob += x >= 4 prob += y <= 2 #
Solve the problem prob.solve()
Print the results
print("Status:", LpStatus[prob.status]) print("Optimal
Solution:") print("x =", value(x)) print("y =", value(y))
print("Optimal Objective Value: Z =", value(objective))
OUTPUT:
Status: Unbounded
Optimal Solution:
x = 5.0 y = 0.0
Optimal Objective Value: Z = 17.5

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+5y + 4z subject
to
2x+ 3y <= 8
2y + 5z <= 10   3x
+ 2y + 4z <= 15
x>=0,y>=0,z>=0 Syntax:
from pulp import *
# Create a minimization problem
prob = LpProblem("Minimization Problem", LpMinimize)
# Define 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 +
5*y + 4*z, "Z" # Define the constraints prob
+= 2*x + 3*y <= 8, "Constraint 1" prob +=
2*y + 5*z <= 10, "Constraint 2" prob += 3*x +
2*y + 4*z <= 15, "Constraint 3"
Solve the problem prob.solve()
Print the status of the problem print("Status:",
LpStatus[prob.status])

# Print the optimal solution
print("Optimal Solution:")
print("x =", value(x))
print("y =", value(y)) print("z
=", value(z))
# Print the optimal objective value
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) Reflection through y – axis
(II) Scaling in X – coordinate by 6 factor
(III) Scaling in Y – coordinate by factor 4.1 (IV) Shearing in X Direction by 7/2 units Syntax:

```
# Initial point
x = -2 y = 4
# (I) Reflection through y-axis
print("Point after reflection through y-axis:")
x = -x y = y
print("x =", x) print("y =", y)
# (II) Scaling in X-coordinate by 6 factor
print("\nPoint after scaling in X-coordinate by 6 factor:")
x = x * 6 y = y
print("x =", x) print("y =", y)
# (III) Scaling in Y-coordinate by factor 4.1
print("\nPoint after scaling in Y-coordinate by factor 4.1:")
x = x y = y * 4.1 print("x =", x) print("y =", y)
# (IV) Shearing in X Direction by 7/2 units
print("\nPoint after shearing in X Direction by 7/2 units:")
x = x + (7/2) * y y = y
print("x =", x) print("y =", y)
```

OUTPUT:
Point after reflection through y-axis:
x = 2 y = 4
Point after scaling in X-coordinate by 6 factor:
x = 12

y = 4
Point after scaling in Y-coordinate by factor 4.1:
x = 12
y = 16.4
Point after shearing in X Direction by 7/2 units:
x = -55.7 y = 16.4
Q.10) Find the combined transformation on line segment between the point A[4,1] & B[-3,0] by using Python program for the following sequence of transformation:-
(I) Rotation about origin through an angle pi/4.
(II) Uniform scaling by 7.3units (III)
Scaling in X Coordinate by 3 units.
(IV) Shearing in X – Direction by 1/2 unit.
Syntax: import
numpy as np

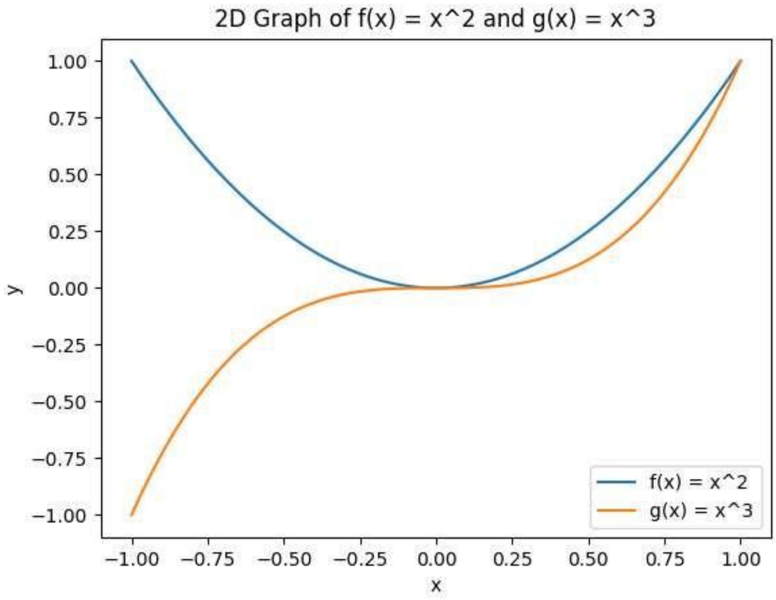
```
# Initial points
A = np.array([4, 1])
B = np.array([-3, 0])
# (I) Rotation about origin through an angle pi/4 theta
= np.pi/4
rot_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                       [np.sin(theta), np.cos(theta)]])
A = np.dot(rot_matrix, A) B
= np.dot(rot_matrix, B)
print("Points after rotation about origin through angle pi/4:")
print("A =", A) print("B =", B)
# (II) Uniform scaling by 7.3 units
scale_factor = 7.3 A
= A * scale_factor B
= B * scale_factor
print("\nPoints after uniform scaling by 7.3 units:")
print("A =", A) print("B =", B)
# (III) Scaling in X Coordinate by 3 units
scale_x = 3 A[0] =
A[0] * scale_x B[0] =
B[0] * scale_x
print("\nPoints after scaling in X Coordinate by 3 units:")
print("A =", A) print("B =", B)
# (IV) Shearing in X Direction by 1/2 unit
shear_x = 1/2
```

A[0] = A[0] + shear_x * A[1] B[0]
= B[0] + shear_x * B[1]
print("\nPoints after shearing in X Direction by 1/2 unit:") print("A
=", A)
print("B =", B)
OUTPUT:
Points after rotation about origin through angle pi/4:
A = [2.12132034 3.53553391]
B = [-2.12132034 -2.12132034]
Points after uniform scaling by 7.3 units: A
= [15.48563851 25.80939751]
B = [-15.48563851 -15.48563851]
Points after scaling in X Coordinate by 3 units:
A = [46.45691552 25.80939751]
B = [-46.45691552 -15.48563851]
Points after shearing in X Direction by 1/2 unit:
A = [59.36161428 25.80939751]
B = [-54.19973478 -15.48563851]

SLIP-13

Q.1) Write a Python program to plot 2D graph of the functions $f(x) = x^2$ and $g(x) = x^3$ in [-1, 1] Syntax: import matplotlib.pyplot as plt import numpy as np def f(x):

```
return x**2 def
g(x):
   return x**3
# Generate x values in the range [-1, 1] x
= np.linspace(-1, 1, 100)
# Calculate y values for f(x) and g(x)
y_f = f(x) y_g = g(x)
# Create a figure and axes fig,
ax = plt.subplots()
# Plot f(x) and g(x) on the same graph
ax.plot(x, y_f, label='f(x) = x^2') ax.plot(x,
y_g, label='g(x) = x^3')
# Add labels and legend
ax.set_xlabel('x')
ax.set_ylabel('y') ax.legend()
# Set title ax.set_title('2D Graph of f(x) = x^2 and
g(x) = x^3')
# Show the plot plt.show()
```



OUTPUT:

Q.2) 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 values for x and y

x = np.linspace(-6, 6, 100) y

= np.linspace(-6, 6, 100)

X, Y = np.meshgrid(x, y)

Calculate values for z based on the parabola equation

Z = X**2 + Y**2

# Create a 3D figure
fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

Plot the surface plot surf =

ax.plot_surface(X, Y, Z, cmap='viridis')

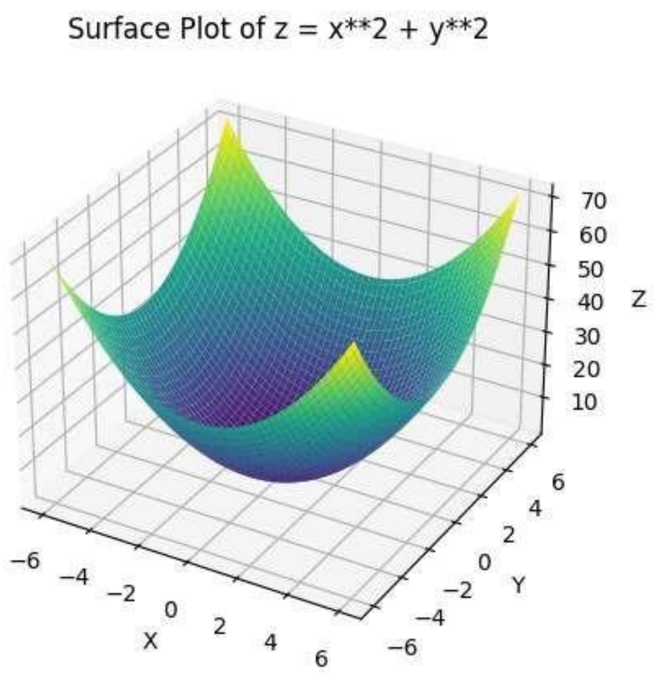
Set labels for x, y, z axes ax.set_xlabel('X')

ax.set_ylabel('Y') ax.set_zlabel('Z') # Set title

for the graph ax.set_title('Surface Plot of z =

x**2 + y**2')

Show the plot plt.show()



OUTPUT:

Q.3) 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:

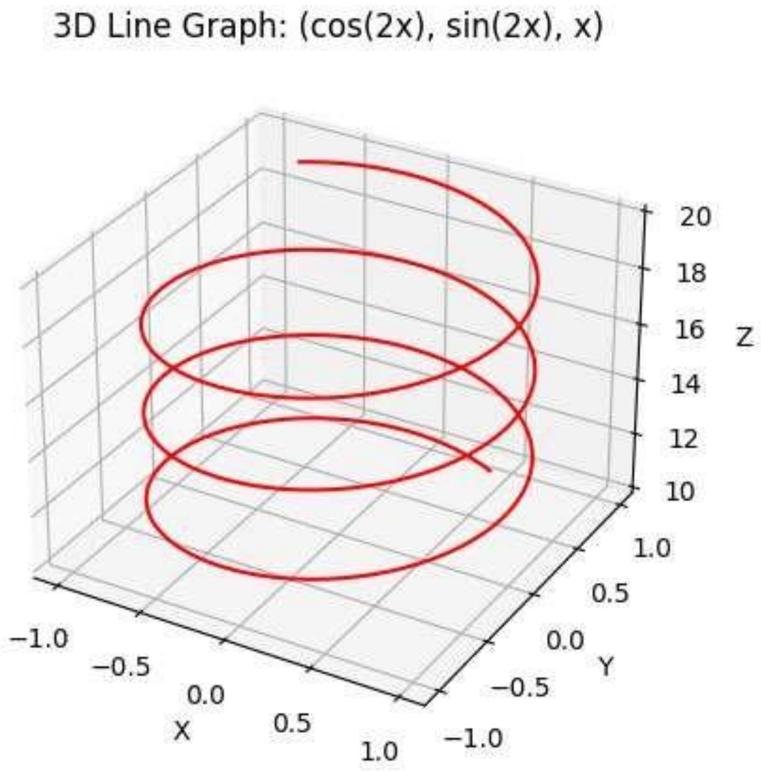
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D #
Generate values for x x = np.linspace(10,
20, 500)
# Calculate parametric equations for x, y, z
y = np.sin(2 * x) z = x
x = np.cos(2 * x) # Create a 3D figure fig
= plt.figure() ax = fig.add_subplot(111,
projection='3d')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

A(1,0),D(2, -1),C(-1,3).

Syntax:

def reflect_point(point, line_y):
x, y = point   y_reflected = 2 *
line_y - y   return x,
y_reflected # Define the points
A, D, and C
A = (1, 0)
D = (2, -1)
C = (-1, 3)

Define the line of reflection line_y
= 3
# Reflect the points A, D, and C through the line of reflection
A_reflected = reflect_point(A, line_y)
D_reflected = reflect_point(D, line_y)
C_reflected = reflect_point(C, line_y)
# Print the reflected points
print("Original Points:")
print("A:", A) print("D:", D)
print("C:", C) print("Reflected
Points:") print("A_reflected:",
A_reflected) print("D_reflected:",
D_reflected) print("C_reflected:",
C_reflected) Output:
Original Points:
A: (1, 0)
D: (2, -1)
C: (-1, 3)
Reflected Points:
A_reflected: (1, 6)
D_reflected: (2, 7)
C_reflected: (-1, 3)

Q.5) Using sympy declare the points P(5, 2), Q(5, -2), R(5, O), check whether these points are collinear. Declare the ray passing through the points P and Q, find the length of this ray between P and Q. Also find slope of this ray. Syntax:

from sympy import *
# Declare the points P, Q, and R
P = Point(5, 2)
Q = Point(5, -2)
R = Point(5, Symbol('O'))
# Check if points P, Q, and R are collinear
collinear = Point.is_collinear(P, Q, R) if
collinear:
   print("Points P, Q, and R are collinear.") else:
print("Points P, Q, and R are not collinear.") #
Declarethe ray passing through points P and Q
ray_PQ = Ray(P, Q)
# Find the length of the ray between points P and Q

length_PQ = ray_PQ.length
print("Length of ray PQ between points P and Q:", length_PQ)
# Find the slope of the ray PQ
slope_PQ = ray_PQ.slope print("Slope
of ray PQ:", slope_PQ)

OUTPUT:
Points P, Q, and R are collinear.
Length of ray PQ between points P and Q: 00
Slope of ray PQ: 00

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([4, 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: 4.0

Side BC: 3.1622776601683795

Side CA: 4.242640687119285

Area: 6.0000000000000036

Perimeter: 11.404918347287666

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

Max Z = 5x +

3y Subjected to

x + y <= 7 2x +

5y <= 1 x > 0 y

> 0 Syntax:

```
from scipy.optimize import linprog
# Objective function coefficients c
= [-5, -3]
# Coefficient matrix of inequality constraints
A = [[1, 1],
    [2, 5]]
# Right-hand side of inequality constraints b
= [7, 1]
# Bounds on variables
x_bounds = (0, None)
y_bounds = (0, None)
```

Solve the linear programming problem res = linprog(c, A_ub=A,

b_ub=b, bounds=[x_bounds, y_bounds])

Check if the optimization was successful if

res.success:

print("Optimal solution found:")

print("x =", res.x[0]) print("y =",

res.x[1]) print("Maximum value of Z

=", -res.fun) else:

print("Optimization failed. Message:", res.message)

OUTPUT:

Optimal solution found:

x = 0.5 y

= 0.0

Maximum value of Z = 2.5

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 + 2z <= 460 x + 4y <= 120

x=>0, y=>0,z=>0 Syntax:

from pulp import *

Create a minimization problem prob = LpProblem("Linear

Programming Problem", LpMinimize)

Define decision variables x =

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

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

LpVariable('z', lowBound=0) #

Define the objective function

prob += 3 * x + 2 * y + 5 * z #

Define the constraints

prob += x + 2 * y + z <= 430

prob += 3 * x + 2 * z <= 460

prob += x + 4 * y <= 120 #

Solve the problem

prob.solve()

Print the status of the problem print("Status:",

LpStatus[prob.status])

If the problem is solved, print the optimal solution and its value if

prob.status == LpStatusOptimal:

print("Optimal Solution:") print("x =",

value(x)) print("y =", value(y)) print("z =",

value(z)) print("Objective Value =",

value(prob.objective)) else:

print("No optimal solution found.")

OUTPUT:

Status: Optimal Optimal

Solution:

x = 0.0

y = 0.0 z

= 0.0

Objective Value = 0.0

Q.9) Apply Python. Program in each of the following transformation on the point P[-2,4]
(I)Shearing in Y direction by 7 units
(II)Scaling in X- and Y co-ordinate by 7/2 and 7 units respectively.

(III) Scaling in X- and Y co-ordinate by 4 and 7 units respectively. (IV) Rotation about origin by an angle 60^0

Syntax:

import math #
Original point P
P = [-2, 4]
# Transformation 1: Shearing in Y direction by 7 units shear_y
= 7
P_sheared_y = [P[0], P[1] + shear_y]
print("Point after Shearing in Y direction by 7 units:", P_sheared_y)
# Transformation 2: Scaling in X- and Y-coordinate by 7/2 and 7 units respectively
scale_x_1 = 7/2 scale_y_1 = 7
P_scaled_1 = [P_sheared_y[0] * scale_x_1, P_sheared_y[1] * scale_y_1]
print("Point after Scaling in X- and Y-coordinate by 7/2 and 7 units respectively:", P_scaled_1)
# Transformation 3: Scaling in X- and Y-coordinate by 4 and 7 units respectively
scale_x_2 = 4 scale_y_2 = 7
P_scaled_2 = [P_scaled_1[0] * scale_x_2, P_scaled_1[1] * scale_y_2]
print("Point after Scaling in X- and Y-coordinate by 4 and 7 units respectively:", P_scaled_2)
# Transformation 4: Rotation about origin by an angle of 60 degrees angle
= 60
angle_rad = math.radians(angle)
P_rotated = [P_scaled_2[0] * math.cos(angle_rad) - P_scaled_2[1] * math.sin(angle_rad), P_scaled_2[0] * math.sin(angle_rad) + P_scaled_2[1] * math.cos(angle_rad)] print("Point after Rotation about origin by an angle of 60 degrees:", P_rotated)

OUTPUT:

Point after Shearing in Y direction by 7 units: [-2, 11]
Point after Scaling in X- and Y-coordinate by 7/2 and 7 units respectively: [-7.0, 77]
Point after Scaling in X- and Y-coordinate by 4 and 7 units respectively: [-28.0, 539]
Point after Rotation about origin by an angle of 60 degrees: [-480.7876926398124, 245.25128869403576]

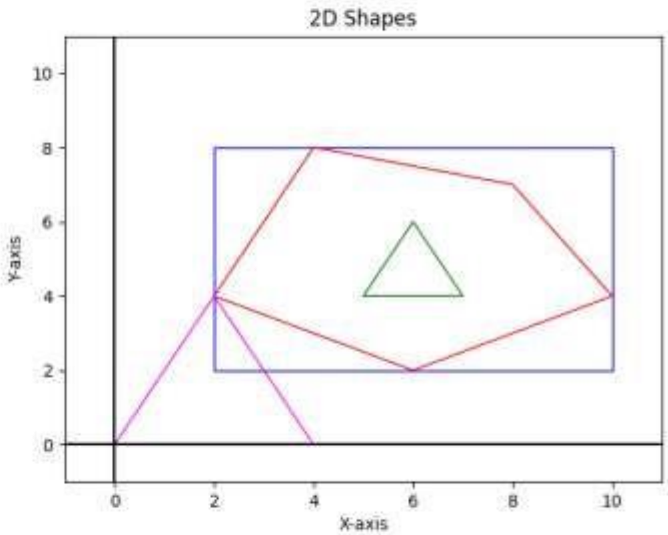
Q.10) Write a python program to Plot 2D X-axis and Y-axis in black color. In the same diagram plot:-

(I) Green Triangle with vertices [5,4],[7,4],[6,6]
(II) Blue rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8].
(III) Red polygon with vertices [6, 2], [10, 4], [8, 7], [4, 8], [2, 4].

(IV) Isosceles triangle with vertices [0, 0], [4, 0], [2, 4].

Syntax: import
matplotlib.pyplot as plt #
Create figure and axis fig, ax
= plt.subplots()
Plot X-axis and Y-axis in black color
ax.axhline(0, color='black') ax.axvline(0,
color='black')
# Green Triangle with vertices [5,4],[7,4],[6,6]
green_triangle = plt.Polygon([[5, 4], [7, 4], [6, 6]], edgecolor='green',
facecolor='none')
ax.add_patch(green_triangle)
# Blue Rectangle with vertices [2, 2], [10, 2], [10, 8], [2, 8]
blue_rectangle = plt.Polygon([[2, 2], [10, 2], [10, 8], [2, 8]], edgecolor='blue',
facecolor='none')
ax.add_patch(blue_rectangle)
# Red Polygon with vertices [6, 2], [10, 4], [8, 7], [4, 8], [2, 4]
red_polygon = plt.Polygon([[6, 2], [10, 4], [8, 7], [4, 8], [2, 4]], edgecolor='red',
facecolor='none') ax.add_patch(red_polygon)
# Isosceles Triangle with vertices [0, 0], [4, 0], [2, 4]
isosceles_triangle = plt.Polygon([[0, 0], [4, 0], [2, 4]], edgecolor='magenta',
facecolor='none')
ax.add_patch(isosceles_triangle)
# Set axis limits
ax.set_xlim([-1, 11])
ax.set_ylim([-1, 11]) #
Set labels and title
ax.set_xlabel('X-
axis')
ax.set_ylabel('Y-
axis')
ax.set_title('2D
Shapes') # Show
the plot
plt.show()

OUTPUT:

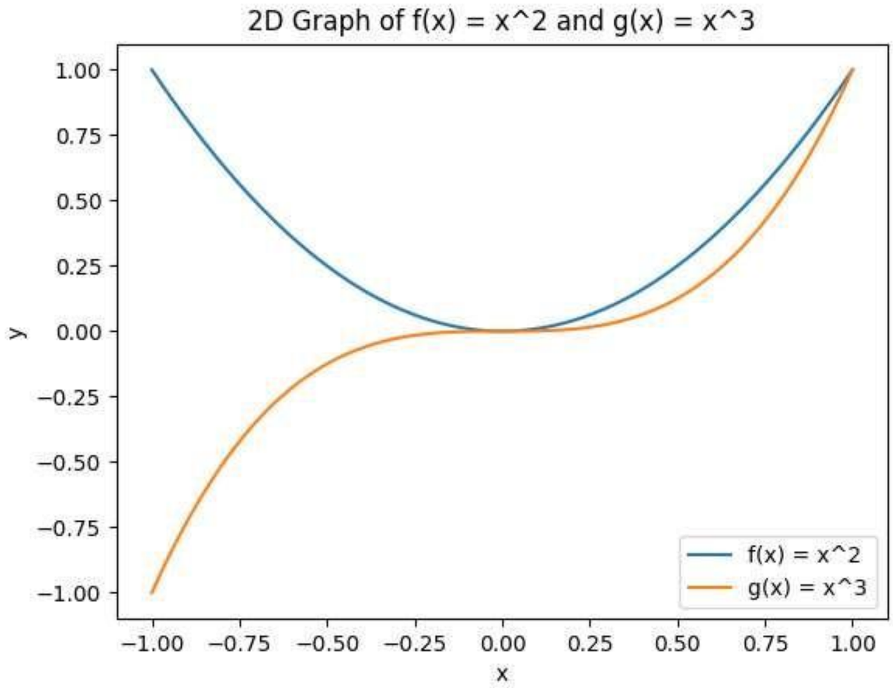


SLIP-14

Q.1)Write a Python program to plot 2D graph of the functions $f(x) = x^2$ and $g(x) = x^3$ in [-1, 1] Syntax: import matplotlib.pyplot as plt import numpy as np def

```
f(x):
    return x**2 def
g(x):
    return x**3
# Generate x values in the range [-1, 1] x
= np.linspace(-1, 1, 100)
# Calculate y values for f(x) and g(x)
y_f = f(x) y_g = g(x)
# Create a figure and axes fig,
ax = plt.subplots()
# Plot f(x) and g(x) on the same graph
ax.plot(x, y_f, label='f(x) = x^2') ax.plot(x,
y_g, label='g(x) = x^3')
# Add labels and legend
ax.set_xlabel('x')
ax.set_ylabel('y') ax.legend()
# Set title ax.set_title('2D Graph of f(x) = x^2 and
g(x) = x^3')
# Show the plot plt.show()
```

OUTPUT:



Q.2) Write a Python program to plot 3D graph of the function f(x) = e**x**3 in [-5, 5] with green dashed points line with upward pointing triangle.

Syntax:

import numpy as np import

matplotlib.pyplot as plt

Generate x values x =

np.linspace(-5, 5, 100)

Compute y values using the given function y

= np.exp(-x**2)

Create 3D plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

Plot the points with green dashed line and upward-pointing triangles ax.plot(x, y, np.zeros_like(x), linestyle='dashed', color='green', marker='^')

Set labels for axes ax.set_xlabel('x')

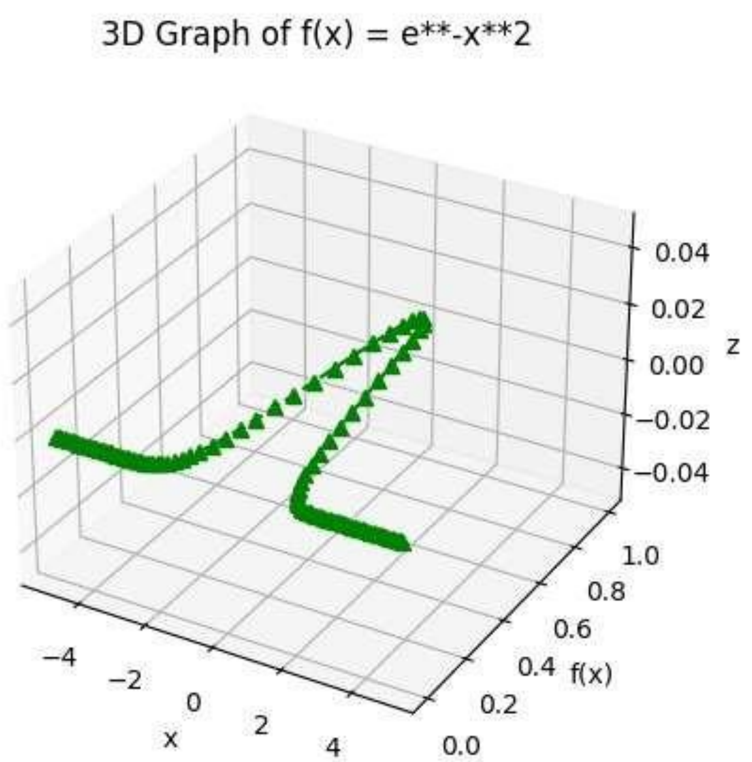
ax.set_ylabel('f(x)') ax.set_zlabel('z') #

Set title for the plot ax.set_title('3D Graph

of f(x) = e**-x**2')

Show the plot plt.show()

OUTPUT:



Q.3) Write a Python program to generate 3D plot of the fu nctions z = sin x +cos y in -5< x, y < 5.

Syntax:

import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D

Generate data x =

np.linspace(-5, 5, 100) y =

np.linspace(-5, 5, 100) X,

Y = np.meshgrid(x, y)

Z = np.sin(X) + np.cos(Y) # Create 3D

plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, Z, cmap='viridis')

ax.set_xlabel('X') ax.set_ylabel('Y')

ax.set_zlabel('Z') ax.set_title('3D Plot of z

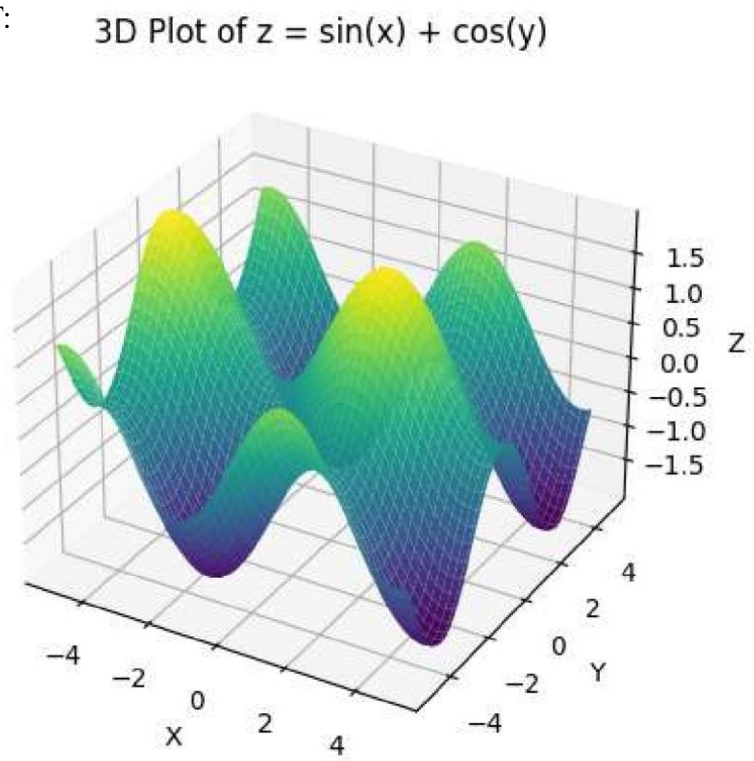
= sin(x) + cos(y)')

Q.4) write a Python program to reflect the line segment joining the points A[5, 3] and B[l, 4] through the line y = x + 1.

Syntax:

import numpy as np #
Define the points A and B
A = np.array([5, 3])
B = np.array([1, 4])
# Define the equation of the reflecting line def reflect(line,
point): m = line[0] c = line[1] x, y = point x_reflect =
plt.show()

OUTPUT:



```
(2 * m * (y - c) + x * (m ** 2 - 1)) / (m ** 2 + 1)     y_reflect = (2
* m * x + y * (1 - m ** 2) + 2 * c) / (m ** 2 + 1)     return
np.array([x_reflect, y_reflect])
# Define the equation of the reflecting line y = x + 1 line
= np.array([1, -1])
# Reflect points A and B through the reflecting line
A_reflected = reflect(line, A)
B_reflected = reflect(line, B) # Print the
reflected points print("Reflected Point
A':", A_reflected) print("Reflected Point
B':", B_reflected)
```

Output:

Reflected Point A': [4. 4.]

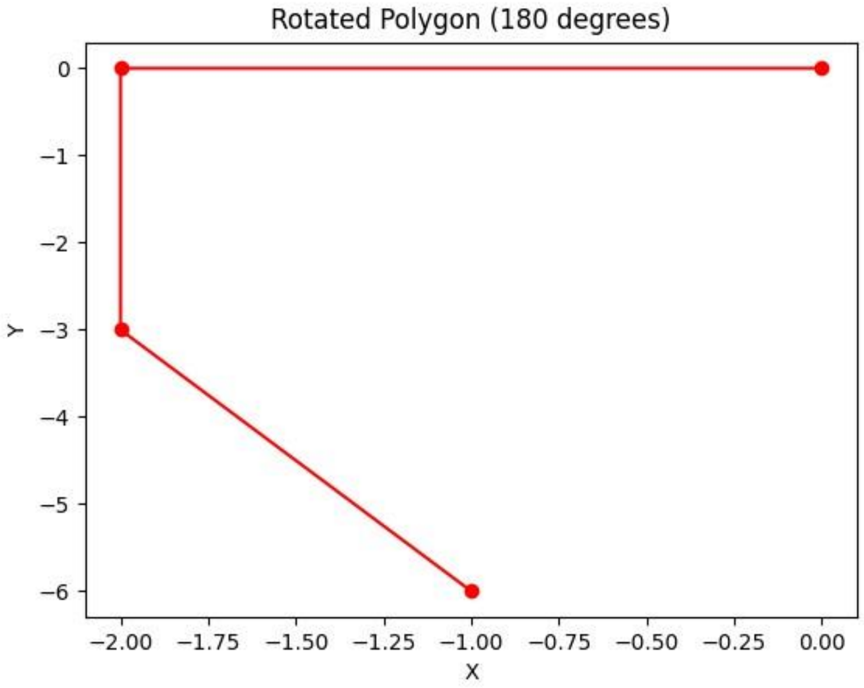
Reflected Point B': [5. 0.]

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')
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 = 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 = 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: Optimal
Status: Infeasible

Q.9) Apply each of the following Transformation on the point P[2, -3].
(I)Refection through X-axis.
(II)Scaling in X-co-ordinate by factor 2.
(III) Scaling in Y-co-ordinate by factor 1.5.
(IV) Reflection through the line y = x

Syntax:

```
import numpy as np # Point P
P = np.array([2, -3])
# Transformation 1: Reflection through X-axis
```

T1 = np.array([[1, 0], [0, -1]])
P_T1 = np.dot(T1, P)
# Transformation 2: Scaling in X-coordinate by factor 2
T2 = np.array([[2, 0], [0, 1]])
P_T2 = np.dot(T2, P)
# Transformation 3: Scaling in Y-coordinate by factor 1.5
T3 = np.array([[1, 0], [0, 1.5]])
P_T3 = np.dot(T3, P)
# Transformation 4: Reflection through the line y = x
T4 = np.array([[0, 1], [1, 0]])
P_T4 = np.dot(T4, P) #
Displaying the results
print("Original Point P: ", P)
print("Transformation 1: Reflection through X-axis: ", P_T1)
print("Transformation 2: Scaling in X-coordinate by factor 2: ", P_T2)
print("Transformation 3: Scaling in Y-coordinate by factor 1.5: ", P_T3)
print("Transformation 4: Reflection through the line y = x: ", P_T4)
OUTPUT:
Original Point P: [2 -3]
Transformation 1: Reflection through X-axis: [2 3]
Transformation 2: Scaling in X-coordinate by factor 2: [4 -3]
Transformation 3: Scaling in Y-coordinate by factor 1.5: [2. -4.5]
Transformation 4: Reflection through the line y = x: [-3 2]

Q.10) Apply each of the following Transformation on the point P[3, -1]. (I) Shearing in Y direction by 2 units.
(II) Scaling in X and Y direction by 1/2 and 3 units respectively.
(III) Shearing in both X and Y direction by -2 and 4 units respectively.
(IV) Rotation about origin by an angle 30 degrees.
Syntax: import
numpy as np import
math
# Point P
P = np.array([3, -1])
# Transformation 1: Shearing in Y direction by 2 units
T1 = np.array([[1, 0], [2, 1]])
P_T1 = np.dot(T1, P)
# Transformation 2: Scaling in X and Y direction by 1/2 and 3 units respectively
T2 = np.array([[1/2, 0], [0, 3]])
P_T2 = np.dot(T2, P)

# Transformation 3: Shearing in both X and Y direction by -2 and 4 units respectively
T3 = np.array([[1, -2], [4, 1]])
P_T3 = np.dot(T3, P)
# Transformation 4: Rotation about origin by an angle 30 degrees
theta = np.deg2rad(30) # Convert angle to radians
T4 = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
P_T4 = np.dot(T4, P) #
Displaying the results
print("Original Point P: ", P)
print("Transformation 1: Shearing in Y direction by 2 units: ", P_T1)
print("Transformation 2: Scaling in X and Y direction by 1/2 and 3 units respectively: ", P_T2)
print("Transformation 3: Shearing in both X and Y direction by -2 and 4 units respectively: ", P_T3)
print("Transformation 4: Rotation about origin by an angle 30 degrees: ", P_T4)
OUTPUT:
Original Point P: [3 -1]
Transformation 1: Shearing in Y direction by 2 units: [3 5]
Transformation 2: Scaling in X and Y direction by 1/2 and 3 units respectively: [1.5 -3.]
Transformation 3: Shearing in both X and Y direction by -2 and 4 units respectively: [5 11]
Transformation 4: Rotation about origin by an angle 30 degrees: [3.09807621 0.6339746]

SLIP-15

Q.1) Write the python program to find area of the triangle ABC where A[0,0],B[5,0],C[3,3] Syntax:

import math def calculate_area(x1, y1,

x2, y2, x3, y3):

"""Function to calculate area of a triangle given its three vertices."""

area = abs((x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2) return

area

Coordinates of vertices A, B, and C

Ax, Ay = 0, 0

Bx, By = 5, 0 Cx,

Cy = 3, 3

Call the function to calculate the area area =

calculate_area(Ax, Ay, Bx, By, Cx, Cy)

Print the result

print("Area of triangle ABC is:", area)

OUTPUT:

Area of triangle ABC is: 7.5

Q.2) Write the python program to plot the graphs of sin x, cox, e**x and x **2 in [0,5] in one figure with 2X2 subplots Syntax: import numpy as np import matplotlib.pyplot as plt

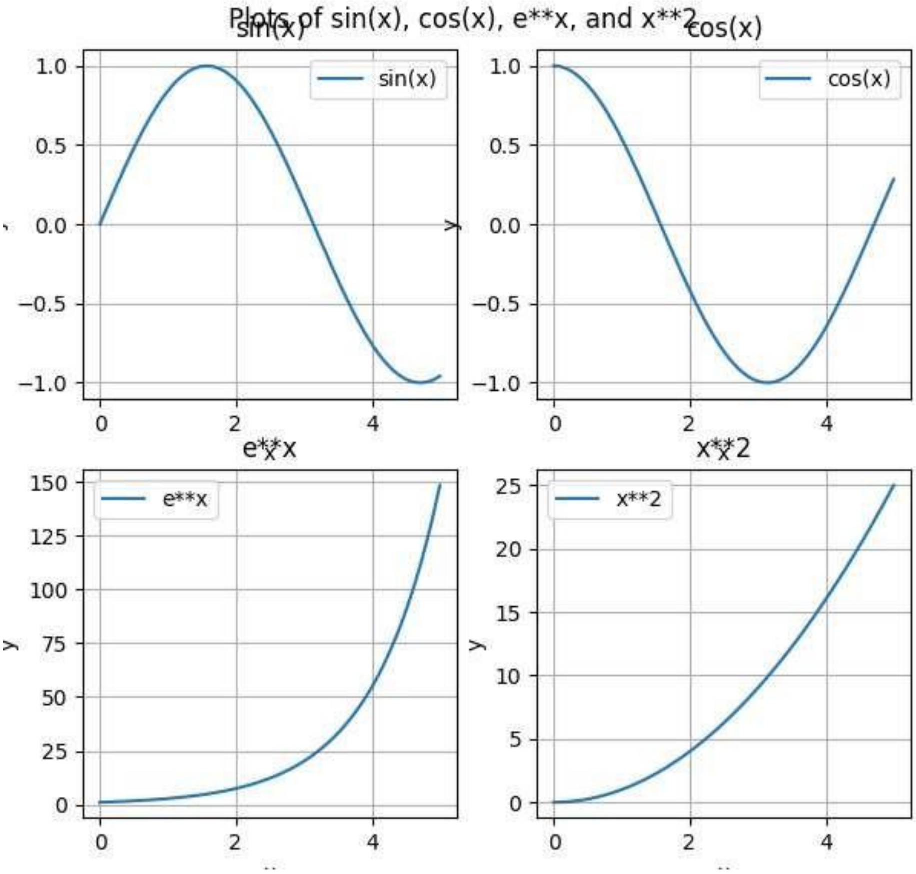
Generate x values in the interval [0, 5] x

= np.linspace(0, 5, 500)

Evaluate sin(x), cos(x), e**x, and x**2 for the x values

y1 = np.sin(x) y2 = np.cos(x) y3 = np.exp(x) y4 = x**2

```
# Create a 2x2 subplot figure fig, axs =
plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle('Plots of sin(x), cos(x), e**x, and x**2')
# Plot sin(x) in the top left subplot
axs[0, 0].plot(x, y1, label='sin(x)')
axs[0, 0].set_title('sin(x)') # Plot
cos(x) in the top right subplot axs[0,
1].plot(x, y2, label='cos(x)') axs[0,
1].set_title('cos(x)') # Plot e**x in
the bottom left subplot axs[1,
0].plot(x, y3, label='e**x') axs[1,
0].set_title('e**x')
# Plot x**2 in the bottom right subplot
axs[1, 1].plot(x, y4, label='x**2') axs[1,
1].set_title('x**2')
# Add labels, legends, and grids to all subplots
for ax in axs.flat:     ax.set_xlabel('x')
ax.set_ylabel('y')     ax.legend()
ax.grid(True)
# Adjust spacing between subplots
fig.tight_layout() #
Show the plot
plt.show()
```



OUTPUT:

Q.3) Write the python program to plot the graph of the function using def ()

$$x^2 + 4, if - 10 < x < 5$$

$f(x) = \{$

$3x + 9, if \quad 5 < x \geq 0$ Syntax:

import numpy as np import

matplotlib.pyplot as plt def

f(x):

"""Function to define f(x)."""

if -10 < x < 5:

return x**2 + 4

elif 5 <= x:

return 3*x + 9

else:

return None # Generate x values x = np.linspace(-11, 11, 500) #

Generate 500 points between -11 and 11

Calculate y values using f(x) y

= np.array([f(xi) for xi in x])

Create the plot

plt.plot(x, y, label='f(x)')

plt.xlabel('x')

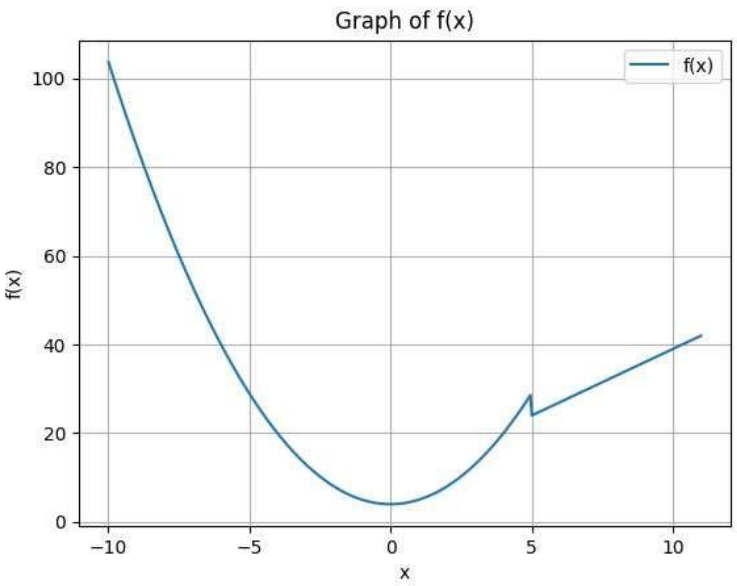
plt.ylabel('f(x)')

plt.title('Graph of f(x)')

plt.legend()

plt.grid(True) plt.show()

OUTPUT:



Q.4) write the

Python program to rotate the triangle ABC by 180 degree, where A [2,1] B[2, -2] & C[-1, 2].

Syntax:

import numpy as np
# Define the original triangle vertices
A = np.array([2, 1])
B = np.array([2, -2])
C = np.array([-1, 2])
# Define the rotation matrix for 180 degrees rotation_matrix
= np.array([[-1, 0],[0, -1]])
# Rotate the triangle vertices 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 rotated triangle vertices
print("Original Triangle Vertices:")
print("A:", A) print("B:", B)
print("C:", C) print("Rotated Triangle
Vertices:") print("A Rotated:",
A_rotated) print("B Rotated:",
B_rotated) print("C Rotated:",
C_rotated) Output:
Original Triangle Vertices:
A: [2 1]

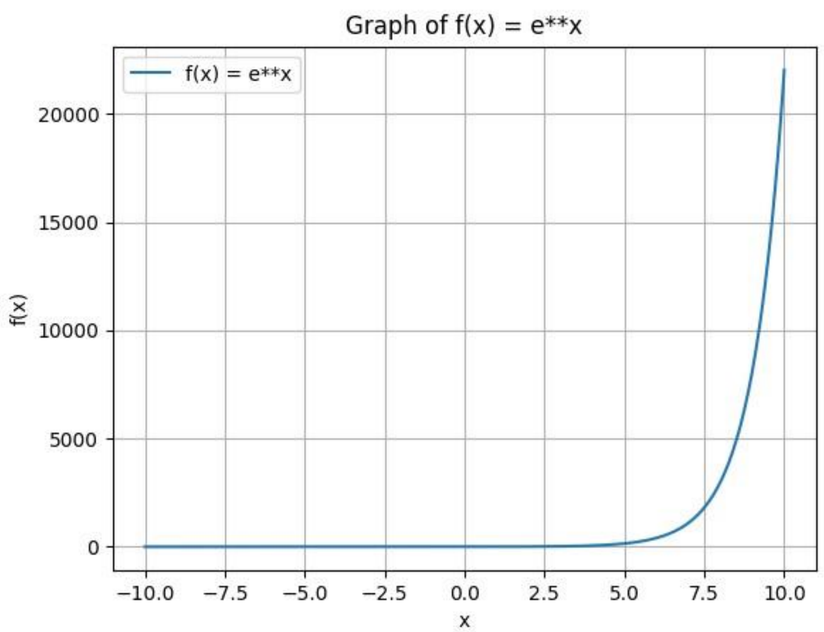
B: [2 -2]
C: [-1 2]
Rotated Triangle Vertices:
A Rotated: [-2 -1]
B Rotated: [-2 2]
C Rotated: [1 -2]

Q.5) Write the Python program to plot the graph of function f(x) = e**x in the interval [-10, 10].

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt # Define
the function f(x) = e**x def f(x):
return np.exp(x)
# Generate x values in the interval [-10, 10]
x = np.linspace(-10, 10, 500) # Evaluate
f(x) for the x values y = f(x) # Create a plot
plt.plot(x, y, label='f(x) = e**x')
plt.xlabel('x') plt.ylabel('f(x)')
plt.title('Graph of f(x) = e**x')
plt.legend() plt.grid(True)
plt.show()
```

OUTPUT:

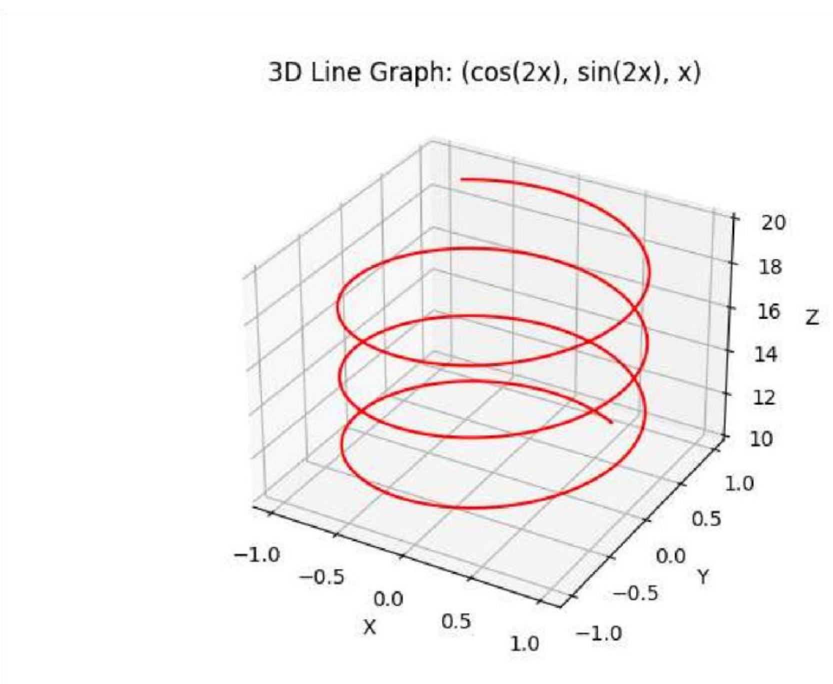


Q.6) Write a Python program to plot 3D line graph Whose parametric equation is (cos(2x),sin(2x),x) for 10 <= x <= 20 (in red color), with title of the graph

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

Max Z = 3.5x +

2y Subjected to x

+ y >= 5 x >=4

y<=5 x >= 0,y>=

0.

Syntax:

```
from pulp import * # Create the LP problem
problem = LpProblem("Maximize Z",
LpMaximize)
# Define the decision variables x =
LpVariable('x', lowBound=0)  # x >= 0 y =
LpVariable('y', lowBound=0)  # y >= 0 #
Define the objective function problem +=
3.5 * x + 2 * y # Define the constraints
problem += x + y >= 5 problem += x >= 4
problem += y <= 5 # Solve the LP
problem status = problem.solve() # Check
the solution status if status == 1:
   # Print the optimal solution
print("Optimal solution:")
print(f"x = {value(x)}")    print(f"y
= {value(y)}")    print(f"Z =
{value(problem.objective)}") else:
   print("No feasible solution found.")
```

OUTPUT:

No feasible solution found.

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: Optimal

Status: Infeasible

Q.9) Write a python program to find the combined transformation of the line segment between the points A[5,3] and B[1,4] for the following sequence of transformation

(I) First rotation about origin through an angle pi/e
(II)Followed by scaling in x co-ordinate by 5 units (III) Followed by reflection through the line y = -x Syntax:

```
import numpy as np

# Define points A and B as numpy arrays
A = np.array([5, 3])
B = np.array([1, 4])

# Print original points A and B
print("Original Points:") print("Point A: ({},
{})".format(A[0], A[1]))
print("Point B: ({}, {})".format(B[0], B[1]))

# Transformation I: Rotation about origin through an angle of pi/2 theta
= np.pi/2
rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)],
                            [np.sin(theta), np.cos(theta)]])
A_rotation = np.dot(rotation_matrix, A)
B_rotation = np.dot(rotation_matrix, B)

# Print points A and B after rotation print("\nPoints after
Rotation:") print("Point A: ({}, {})".format(A_rotation[0],
A_rotation[1])) print("Point B: ({}, {})".format(B_rotation[0],
B_rotation[1])) # Transformation II: Scaling in x-coordinate
by 5 units scaling_matrix = np.array([[5, 0],
                           [0, 1]])
A_scaling = np.dot(scaling_matrix, A_rotation)
```

B_scaling = np.dot(scaling_matrix, B_rotation)
# Print points A and B after scaling print("\nPoints after Scaling:") print("Point A: ({}, {})".format(A_scaling[0], A_scaling[1])) print("Point B: ({}, {})".format(B_scaling[0], B_scaling[1])) # Transformation III: Reflection through the line y = -x reflection_matrix = np.array([[0, -1], [-1, 0]])
A_reflection = np.dot(reflection_matrix, A_scaling)
B_reflection = np.dot(reflection_matrix, B_scaling)
# Print points A and B after reflection print("\nPoints after Reflection:") print("Point A: ({}, {})".format(A_reflection[0], A_reflection[1])) print("Point B: ({}, {})".format(B_reflection[0], B_reflection[1]))

OUTPUT:
Original Points:
Point A: (5, 3) Point
B: (1, 4) Points after
Rotation:
Point A: (-2.9999999999999996, 5.0) Point
B: (-4.0, 1.0000000000000002) Points
after Scaling:
Point A: (-14.999999999999998, 5.0) Point
B: (-20.0, 1.0000000000000002) Points
after Reflection:
Point A: (-5.0, 14.999999999999998)
Point B: (-1.0000000000000002, 20.0)

Q.10) Write the python program to apply each of the following transformation on the point P(-2,4)
(I) Reflection Through the line y = x+1
(II) Scaling in y-Coordinate by factor 1.5
(III) Shearing in x – Direction by 2 unit (IV) Rotation about origin by an angle 45 degree.

Syntax: import numpy as np
# Define the original point P
P = np.array([-2, 4]) #
Print the original point P
print("Original Point:")
print("Point P: ({}, {})".format(P[0], P[1]))
# Transformation I: Reflection through the line y = x + 1

```
reflection_matrix = np.array([[0, 1], [1, 0]])
P_reflection = np.dot(reflection_matrix, P) # Print the point P
after reflection print("\nPoint after Reflection:") print("Point P:
({}, {})".format(P_reflection[0], P_reflection[1])) #
Transformation II: Scaling in y-coordinate by factor 1.5
scaling_matrix = np.array([[1, 0], [0, 1.5]])
P_scaling = np.dot(scaling_matrix, P_reflection)
# Print the point P after scaling print("\nPoint after
Scaling:") print("Point P: ({}, {})".format(P_scaling[0],
P_scaling[1])) # Transformation III: Shearing in x-direction
by 2 units
shearing_matrix = np.array([[1, 2], [0, 1]])
P_shearing = np.dot(shearing_matrix, P_scaling)
# Print the point P after shearing print("\nPoint after Shearing:")
print("Point P: ({}, {})".format(P_shearing[0], P_shearing[1])) #
Transformation IV: Rotation about origin by an angle of 45 degrees
theta = np.deg2rad(45)
rotation_matrix =        np.array([[np.cos(theta),  -np.sin(theta)],
    [np.sin(theta), np.cos(theta)]])
P_rotation = np.dot(rotation_matrix, P_shearing)
# Print the point P after rotation print("\nPoint
after Rotation:")
print("Point P: ({}, {})".format(P_rotation[0], P_rotation[1]))
```

OUTPUT:
Original Point:
Point P: (-2, 4) Point
after Reflection: Point
P: (4, -2) Point after
Scaling: Point P: (4.0,
-3.0) Point after
Shearing: Point P: (-
2.0, -3.0) Point after
Rotation:
Point P: (0.7071067811865477, -3.5355339059327378)

SLIP-16

Q.1) Write a Python program to plot graph of the function f (x, y) = -x*2 – y**2 when -10<=x,y<= 10.

Syntax: import

matplotlib.pyplot as plt import

numpy as np # Define the

function def f(x, y):

    return -x**2 - y**2

Generate x and y values within the range of -10 to 10

x = np.linspace(-10, 10, 100) y = np.linspace(-10, 10,

100) # Create a grid of x and y values

X, Y = np.meshgrid(x, y)

Compute the values of f(x, y) for each (x, y) in the grid

Z = f(X, Y)

Plot the surface using matplotlib fig =

plt.figure() ax = fig.add_subplot(111,

projection='3d') ax.plot_surface(X, Y, Z,

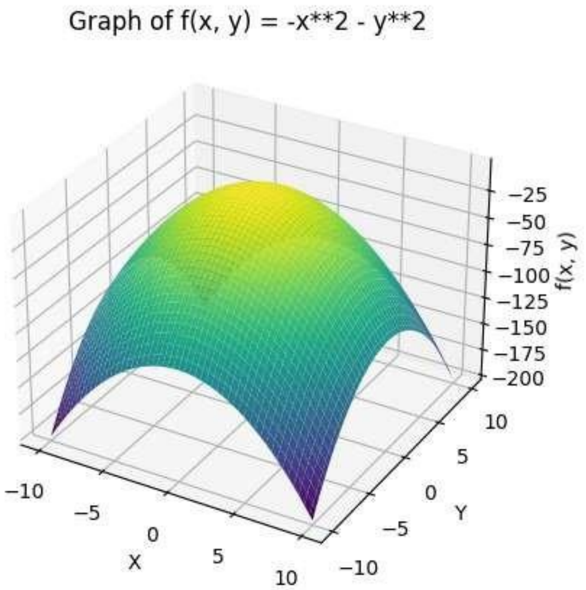
cmap='viridis') ax.set_xlabel('X')

ax.set_ylabel('Y') ax.set_zlabel('f(x, y)')

ax.set_title('Graph of f(x, y) = -x**2 - y**2')

plt.show()

OUTPUT:



Q.2) Write a Python program to plot graph of the function f(x) = log(3x^2) in [1,10] with black dashed points Syntax: import matplotlib.pyplot as plt import numpy as np # Define the function def f(x):

return np.log(3 * x**2)

Generate x values within the range of [1, 10] x

= np.linspace(1, 10, 100)

Compute the values of f(x) for each x in the range y

= f(x)

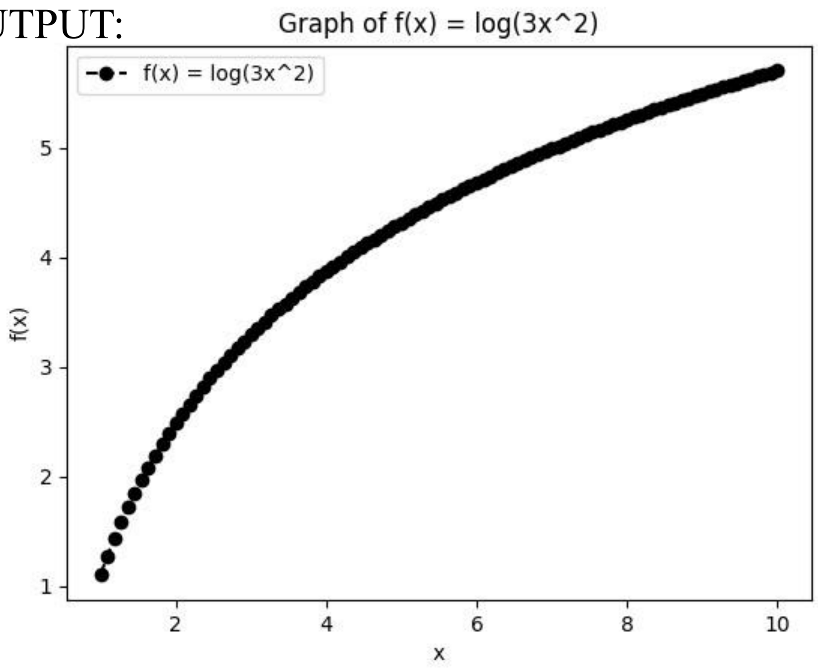
Plot the graph with black dashed points plt.plot(x, y,

'o--', color='black', label='f(x) = log(3x^2)')

plt.xlabel('x') plt.ylabel('f(x)') plt.title('Graph of f(x) =

log(3x^2)') plt.legend() plt.show()

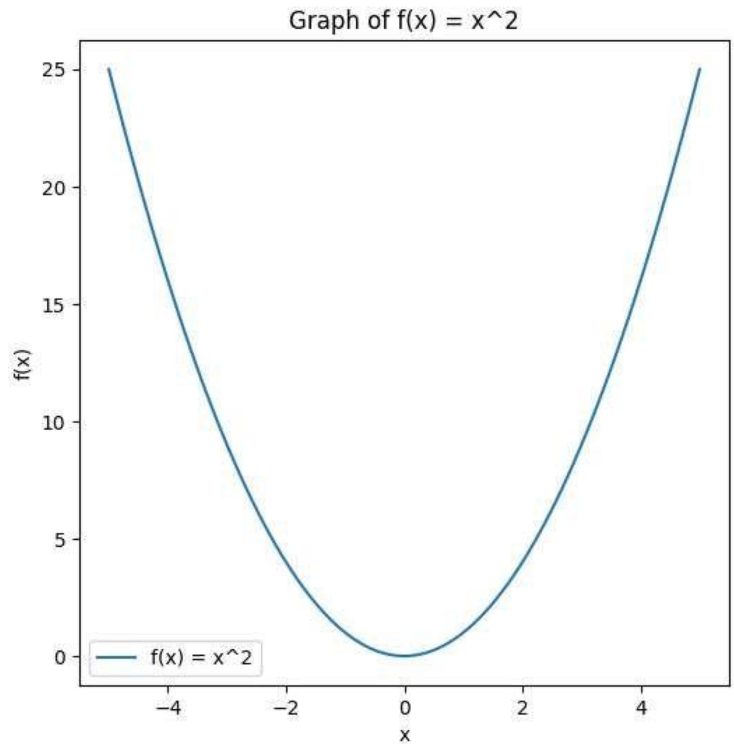
OUTPUT:



Q.3) Write python program to generate plot of the function f(x) = x^2, in the interval [-5,5] in figure of size 6X6 inches Syntax: import matplotlib.pyplot as plt import numpy as np # Define the function def f(x):

```
    return x**2
# Generate x values within the range of [-5, 5] x
= np.linspace(-5, 5, 100)
# Compute the values of f(x) for each x in the range y
= f(x)
# Create a figure with size 6x6 inches
fig = plt.figure(figsize=(6, 6)) # Plot
the graph of the function plt.plot(x, y,
label='f(x) = x^2') plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Graph of f(x) = x^2')
plt.legend() plt.show()
```

OUTPUT:



Q.4) Write a Python program to declare the line segment passing through the points A(0, 7), B(5, 2). Also find the length and midpoint of the line segment passing through points A and B.

Syntax:

import math
# Define the coordinates of points A and B
xA, yA = 0, 7 xB, yB = 5, 2
# Calculate the length of the line segment using the distance formula
length = math.sqrt((xB - xA)**2 + (yB - yA)**2) # Calculate the midpoint of the line segment midpoint_x = (xA + xB) / 2
midpoint_y = (yA + yB) / 2
# Print the equation of the line passing through A and B print("The equation of the line passing through A and B is: ") print(f"y - {yA} = {(yB - yA) / (xB - xA)}(x - {xA})") # Print the length and midpoint of the line segment print(f"Length of the line segment: {length}") print(f"Midpoint of the line segment: ({midpoint_x}, {midpoint_y})")

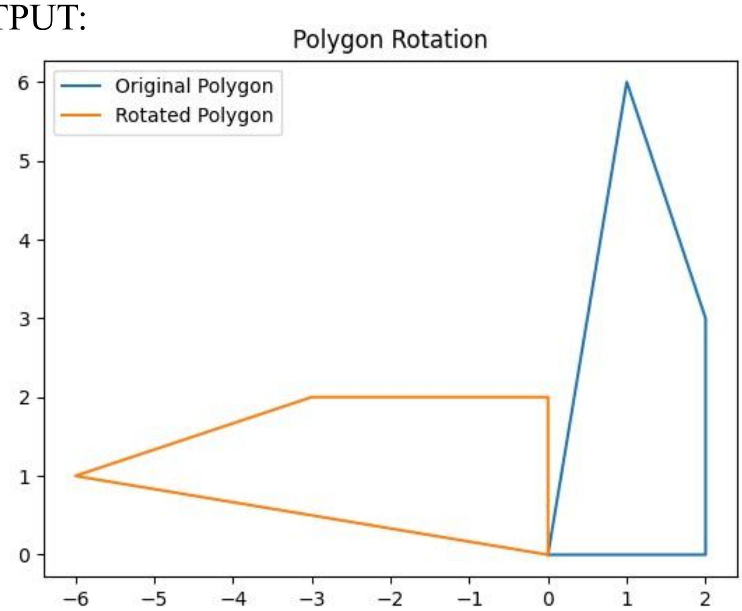
Output:

The equation of the line passing through A and B is: y
- 7 = -1.0(x - 0)
Length of the line segment: 7.0710678118654755
Midpoint of the line segment: (2.5, 4.5)

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 90°. 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], [0, 0]])
# Plot the original polygon
plt.plot(vertices[:, 0], vertices[:, 1], label='Original Polygon')
# Define the rotation angle in degrees rotation_angle
= 90
# Convert the rotation angle to radians
theta = np.radians(rotation_angle) #
Create the rotation matrix
rotation_matrix =       np.array([[np.cos(theta),  -np.sin(theta)],[np.sin(theta),
np.cos(theta)]])
# Apply the rotation matrix to the vertices of the polygon rotated_vertices
= np.dot(vertices, rotation_matrix.T)
# Plot the rotated polygon
plt.plot(rotated_vertices[:, 0], rotated_vertices[:, 1], label='Rotated Polygon')
# Set the aspect ratio to 'equal' for a square plot
plt.axis('equal') # Add legend and title
plt.legend()
plt.title('Polygon Rotation')
# Show the plot
plt.show()
```

OUTPUT:



Q.6) Write a Python program to Generate vector x in the interval [0, 15] using numpy package with 100 subintervals. import numpy as np import numpy as np

Define the start and end values of the interval

start = 0 end = 15

Define the number of subintervals num_subintervals

= 100

Generate the vector x with equally spaced values in the interval [0, 15] x

= np.linspace(start, end, num=num_subintervals+1)

Print the generated vector x

print("Generated vector x:") print(x)

OUTPUT:
Generated vector x:

```
[ 0.    0.15  0.3   0.45  0.6   0.75  0.9   1.05  1.2   1.35  1.5   1.65
  1.8   1.95  2.1   2.25  2.4   2.55  2.7   2.85  3.    3.15  3.3   3.45
  3.6   3.75  3.9   4.05  4.2   4.35  4.5   4.65  4.8   4.95  5.1   5.25
  5.4   5.55  5.7   5.85  6.    6.15  6.3   6.45  6.6   6.75  6.9   7.05
  7.2   7.35  7.5   7.65  7.8   7.95  8.1   8.25  8.4   8.55  8.7   8.85
```

9.   9.15 9.3   9.45 9.6   9.75 9.9 10.05 10.2 10.35 10.5 10.65
10.8 10.95 11.1 11.25 11.4 11.55 11.7 11.85 12.   12.15 12.3 12.45
12.6 12.75 12.9 13.05 13.2 13.35 13.5 13.65 13.8 13.95 14.1 14.25
14.4 14.55 14.7 14.85 15.]

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

Max Z = 3.5x +2 y

Subjected to x + y

>= 5 x >= 4 y <=

2 x > 0 , y > 0

Syntax:

```
import numpy as np from
scipy.optimize import linprog #
Coefficients of the objective function c
= [-3.5, -2]
# Coefficients of the inequality constraints
A = [[-1, -1], [-1, 0], [0, 1]] b = [-5, -4, 2]
# Bounds on the variables
x_bounds = (0, None) y_bounds
= (0, None)
# Solve the linear programming problem
result = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds]) if
result.success:
   print("Optimal solution found:")     print("x
=", result.x[0])     print("y =", result.x[1])
print("Maximum value of Z =", -result.fun)
else:
   print("Optimal solution not found.")
```

OUTPUT:

Optimal solution not found.

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 =
5x+3y subject
to x+y >= 5 x
>= 4 y <= 2
x>= 0, y>= 0

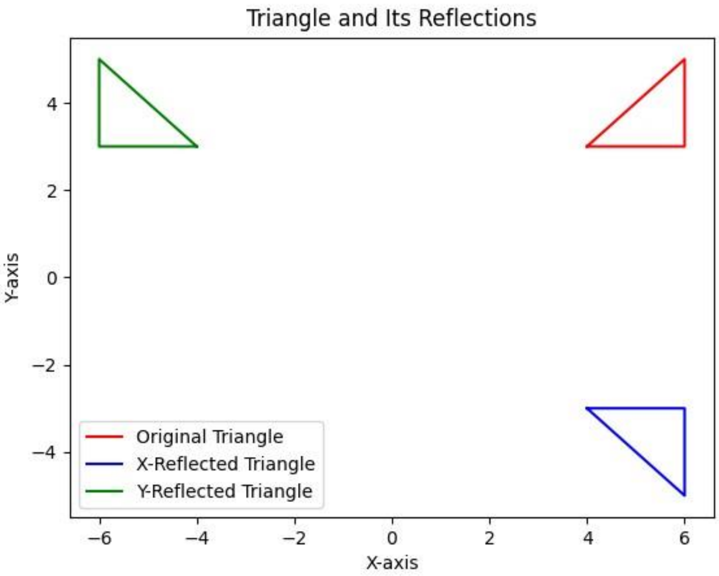
Syntax:

```
from pulp import * #
Create the LP problem
problem = LpProblem("LPP", LpMinimize)
# Define the variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) #
Define the objective function
problem += 5*x + 3*y
# Define the constraints
problem += x + y >= 5
problem += x >= 4
problem += y <= 2 #
Solve the LP problem
problem.solve()
# Print the status of the solution
print("Status:", LpStatus[problem.status])
# If the solution is optimal, print the optimal values of x, y, and Z
if problem.status == 1:     print("Optimal Solution:")     print("x
=", value(x))     print("y =", value(y))     print("Z =", 5*value(x)
+ 3*value(y)) else:
   print("No Optimal Solution Found.")
```

OUTPUT:
Status: Optimal
Optimal Solution:
x = 4.0 y = 1.0 Z
= 23.0

Q.9) Write a python program to plot the Triangle with vertices at [4, 3], [6, 3], [6, 5]. and its reflections through, 1) x-axis, 2) y-axis. All the figures must be in different colors, also plot the two axes. Syntax: import matplotlib.pyplot as plt import numpy as np

```
# Define the vertices of the original triangle
triangle_vertices = np.array([[4, 3], [6, 3], [6, 5], [4, 3]])
# Reflect the triangle through the x-axis
x_reflected_vertices = np.array([triangle_vertices[:, 0], -triangle_vertices[:,
1]]).T
# Reflect the triangle through the y-axis
y_reflected_vertices = np.array([-triangle_vertices[:, 0], triangle_vertices[:,
1]]).T
# Plot the original triangle in red color
plt.plot(triangle_vertices[:, 0], triangle_vertices[:, 1], 'r', label='Original
Triangle')
# Plot the x-reflected triangle in blue color
plt.plot(x_reflected_vertices[:, 0], x_reflected_vertices[:, 1], 'b',
label='XReflected Triangle')
# Plot the y-reflected triangle in green color
plt.plot(y_reflected_vertices[:, 0], y_reflected_vertices[:, 1], 'g',
label='YReflected Triangle')
# Set the axis labels and title
plt.xlabel('X-axis') plt.ylabel('Y-axis')
plt.title('Triangle and Its Reflections')
plt.legend() # Show the plot
```



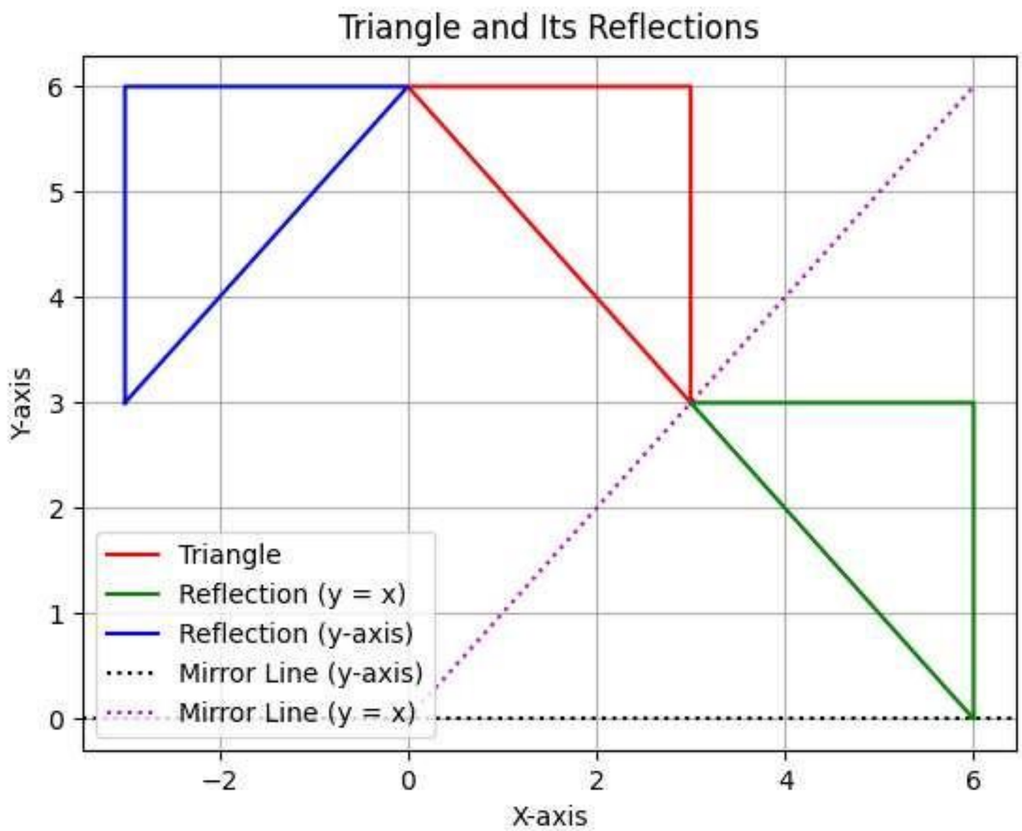

```
plt.show()
```

OUTPUT:

Q.10) Write a python program to plot the Triangle with vertices at [3, 3], [3, 6], [0, 6] and its reflections through, line y = x and y-axis. Also plot the mirror lines.

Syntax:

```
import matplotlib.pyplot as plt
import numpy as np #
Triangle vertices
triangle_vertices = np.array([[3, 3], [3, 6], [0, 6], [3, 3]])
# Reflection through y = x
reflection_y_equals_x = np.dot(triangle_vertices, np.array([[0, 1], [1, 0]]))
# Reflection through y-axis
reflection_y_axis = np.dot(triangle_vertices, np.array([[-1, 0], [0, 1]]))
# Plotting the triangle and its reflections plt.plot(triangle_vertices[:, 0],
triangle_vertices[:, 1], 'r-', label='Triangle') plt.plot(reflection_y_equals_x[:,
    0],     reflection_y_equals_x[:,  1],     'g-', label='Reflection (y = x)')
plt.plot(reflection_y_axis[:, 0], reflection_y_axis[:, 1], 'b-', label='Reflection
(yaxis)')
# Plotting the mirror lines
plt.axhline(0, color='k', linestyle=':', label='Mirror Line (y-axis)')
plt.plot(np.array([0, 6]), np.array([0, 6]), 'm:', label='Mirror Line (y = x)')
plt.xlabel('X-axis') plt.ylabel('Y-axis') plt.title('Triangle and Its Reflections')
plt.legend() plt.grid(True) plt.show()
```


Triangle and Its Reflections
Triangle
Reflection (y = x)
Reflection (y-axis)
Mirror Line (y-axis)
Mirror Line (y = x)
Y-axis
X-axis
0
1
2
3
4
5
6
−2
0
2
4
6

OUTPUT:

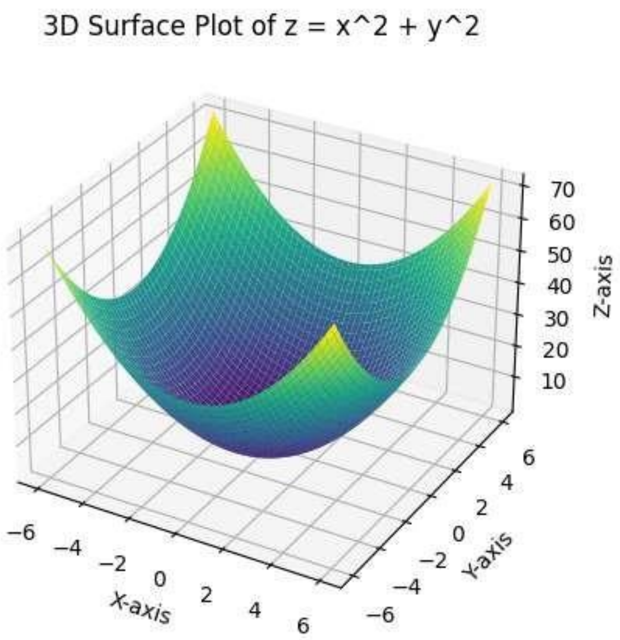
SLIP-17

Q.1) Write a python program to plot the 3D graph of the function z = x^2 + y^2 in -6 < x, y < 6 using surface plot.

Syntax: import

```
matplotlib.pyplot as plt import

numpy as np

# Create a meshgrid for x and y values

x = np.linspace(-6, 6, 100) y =

np.linspace(-6, 6, 100) X, Y =

np.meshgrid(x, y)

# Compute the values of z Z

= X**2 + Y**2

# Create a 3D surface 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-axis')

ax.set_ylabel('Y-axis') ax.set_zlabel('Z-axis')

ax.set_title('3D Surface Plot of z = x^2 + y^2')

# Show the plot plt.show()
```

OUTPUT:



Q.2) Write a python program to plot 3D contours for the function f(x,y) = log(x^2y2) when -5<=x,y<=5 with green color map Syntax: import matplotlib.pyplot as plt import numpy as np

Create a meshgrid for x and y values

x = np.linspace(-5, 5, 100) y =

np.linspace(-5, 5, 100) X, Y =

np.meshgrid(x, y)

Compute the values of f(x, y)

Z = np.log(X**2 * Y**2) # Create a 3D

contour plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

ax.contour(X, Y, Z, cmap='Greens')

Set labels and title ax.set_xlabel('X-

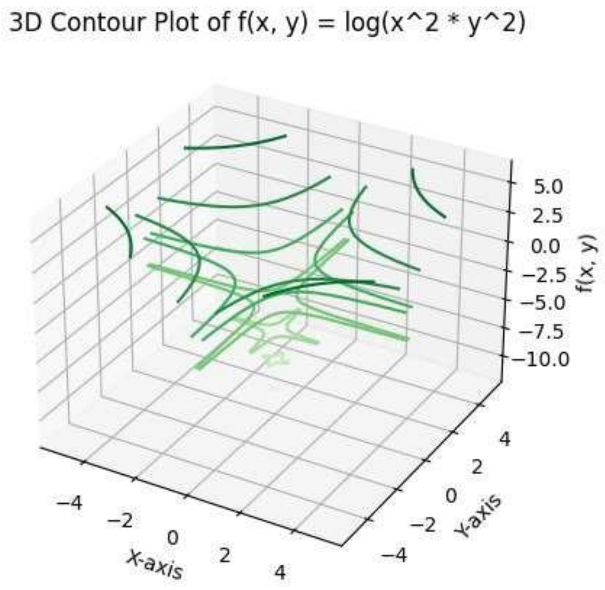
axis') ax.set_ylabel('Y-axis')

ax.set_zlabel('f(x, y)') ax.set_title('3D

Contour Plot of f(x, y) = log(x^2 *

y^2)')

Show the plot plt.show()



OUTPUT:

Q.3) Write a Python program to reflect the line segment joining the points A[-5, 2] and D[l, 3] through the line y = x.

Syntax: import

matplotlib.pyplot as plt import

numpy as np # Define the

points A and D

A = np.array([-5, 2])

D = np.array([1, 3]) # Define

the line y = x def

reflect_y_equals_x(point):

    return np.array([point[1], point[0]]) #

Reflect points A and D through the line y = x

A_reflected = reflect_y_equals_x(A)

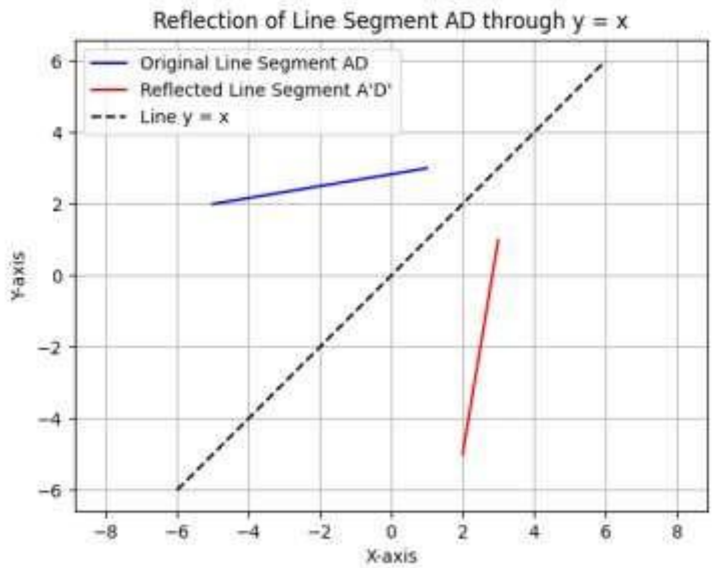
D_reflected = reflect_y_equals_x(D)

Plot the original line segment and its reflection

fig, ax = plt.subplots() ax.plot([A[0], D[0]], [A[1], D[1]], 'b', label='Original Line Segment AD') ax.plot([A_reflected[0], D_reflected[0]], [A_reflected[1], D_reflected[1]], 'r', label='Reflected Line Segment A\'D\'') ax.plot([-6, 6], [-6, 6], 'k--', label='Line y = x') # Plot the line y = x

ax.legend() ax.set_xlabel('X-axis') ax.set_ylabel('Y-axis') ax.set_title('Reflection of Line Segment AD through y = x') plt.axis('equal') plt.grid(True) plt.show()

OUTPUT:

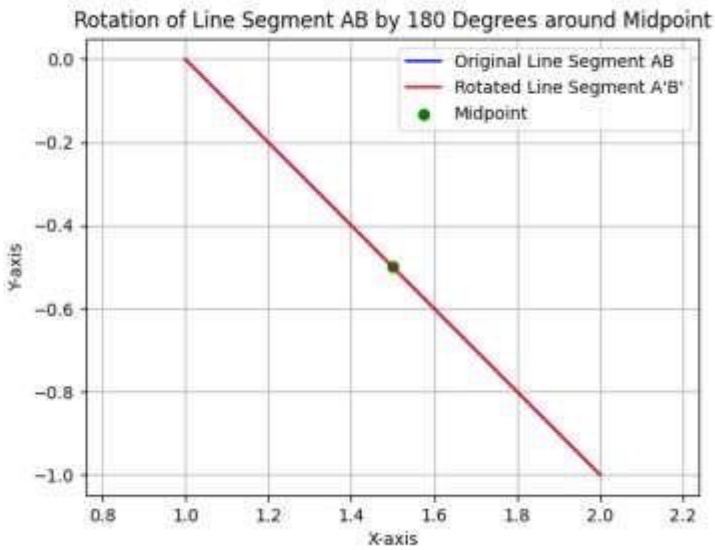


Q.4) write a python program to rotate line line segment by 180 degrees having end points (1, 0) and (2,-1).

Syntax:

import matplotlib.pyplot as plt import
numpy as np
# Define the end points of the line segment
A = np.array([1, 0])
B = np.array([2, -1])
# Find the midpoint of the line segment midpoint
= (A + B) / 2
# Define the rotation matrix for 180 degrees
rotation_matrix = np.array([[-1, 0],[0, -1]])
# Rotate the end points of the line segment around the midpoint
A_rotated = np.dot(rotation_matrix, A - midpoint) + midpoint
B_rotated = np.dot(rotation_matrix, B - midpoint) + midpoint # Plot the original line segment and its rotated version fig, ax = plt.subplots() ax.plot([A[0], B[0]], [A[1], B[1]], 'b', label='Original Line Segment AB') ax.plot([A_rotated[0], B_rotated[0]], [A_rotated[1], B_rotated[1]], 'r', label='Rotated Line Segment A\'B\'')

```
ax.scatter(midpoint[0], midpoint[1], color='g', marker='o', label='Midpoint')  #
Plot the midpoint ax.legend()
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Rotation of Line Segment AB by 180 Degrees around Midpoint')
plt.axis('equal')
plt.grid(True)
```



plt.show() Output:

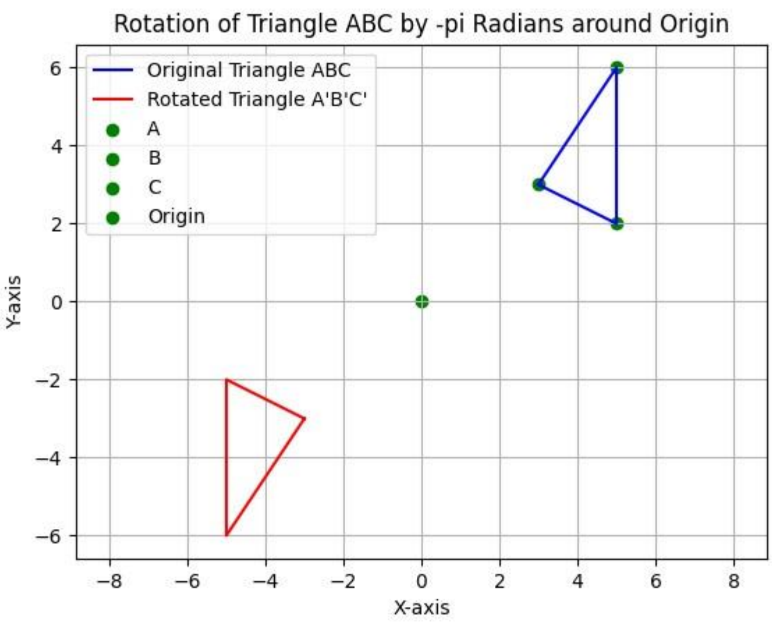
Q.5) Write a python program to plot triangle with vertices [3, 3], [5, 6], [5, 2], and its rotation about the origin by angle –pi radians.

Syntax:

```
import matplotlib.pyplot as plt import
numpy as np
# Define the vertices of the triangle
A = np.array([3, 3])
```

```
B = np.array([5, 6])
C = np.array([5, 2])
# Define the rotation angle in radians theta
= -np.pi
# Define the rotation matrix for the given angle
rotation_matrix =        np.array([[np.cos(theta),  -np.sin(theta)],[np.sin(theta),
np.cos(theta)]])
# Rotate the vertices of the triangle around the origin
A_rotated = np.dot(rotation_matrix, A)
B_rotated = np.dot(rotation_matrix, B)
C_rotated = np.dot(rotation_matrix, C)
# Plot the original triangle and its rotated version fig,
ax = plt.subplots()
ax.plot([A[0], B[0], C[0], A[0]], [A[1], B[1], C[1], A[1]], 'b', label='Original
Triangle ABC') ax.plot([A_rotated[0], B_rotated[0], C_rotated[0], A_rotated[0]],
[A_rotated[1], B_rotated[1], C_rotated[1], A_rotated[1]], 'r', label='Rotated
Triangle A\'B\'C\'') ax.scatter(A[0], A[1], color='g', marker='o', label='A')  # Plot
vertex A ax.scatter(B[0], B[1], color='g', marker='o', label='B')  # Plot vertex B
ax.scatter(C[0], C[1], color='g', marker='o', label='C')   # Plot vertex C
ax.scatter(0, 0, color='g', marker='o', label='Origin')  # Plot origin ax.legend()
ax.set_xlabel('X-axis') ax.set_ylabel('Y-axis')
ax.set_title('Rotation of Triangle ABC by -pi Radians around Origin')
plt.axis('equal') plt.grid(True)
plt.show()
```

OUTPUT:

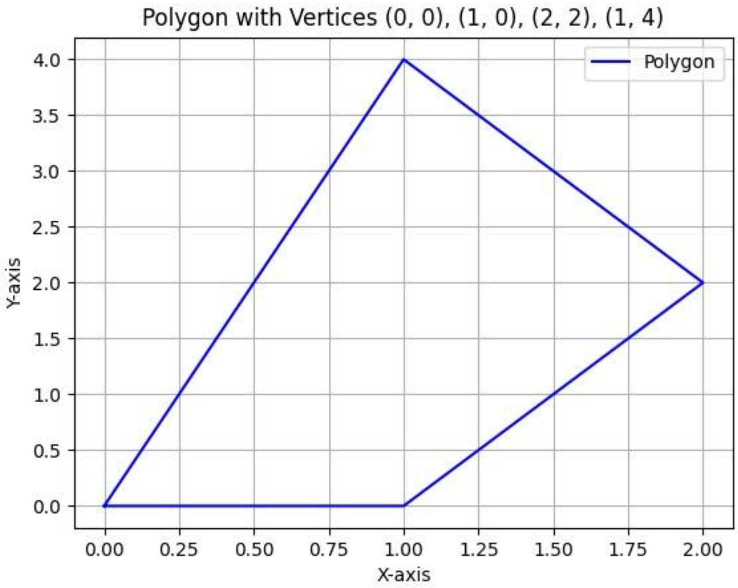


Q.6) Write a python program to draw a polygon with vertices (0, 0), (1, 0), (2, 2), (1, 4) and find its area and perimeter.

Synatx:

import matplotlib.pyplot as plt import
numpy as np
# Define the vertices of the polygon
vertices = np.array([[0, 0], [1, 0], [2, 2], [1, 4], [0, 0]])
# Extract x and y coordinates of the vertices
x = vertices[:, 0] y = vertices[:, 1]
# Plot the polygon fig, ax =
plt.subplots() ax.plot(x, y, 'b',
label='Polygon') # Calculate the
area of the polygon
area = 0.5 * np.abs(np.dot(x, np.roll(y, 1)) - np.dot(y, np.roll(x, 1)))
# Calculate the perimeter of the polygon
perimeter = np.sum(np.sqrt(np.diff(x) ** 2 + np.diff(y) ** 2))
# Print the calculated area and perimeter
print("Area of the polygon: ", area)
print("Perimeter of the polygon: ", perimeter)
ax.set_xlabel('X-axis') ax.set_ylabel('Y-axis')
ax.set_title('Polygon with Vertices (0, 0), (1, 0), (2, 2), (1, 4)') ax.legend()
plt.grid(True)
plt.show()

OUTPUT:



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

Max Z = 4x + y + 3z + 5w

Subjected to

4x + 6y - 5z - 4w >= -20 -8x

- 3y + 3z + 2w <= 5 20

x > 0 , y > 0 Syntax:

from pulp import * #
Create the LP problem
lp_problem = LpProblem("Linear_Programming_Problem",
LpMaximize)
# 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') w = LpVariable('w',
lowBound=0, cat='Continuous')
# Set the objective function
lp_problem += 4*x + y + 3*z + 5*w
Add the constraints lp_problem += 4*x +
6*y - 5*z - 4*w >= -20 lp_problem += -8*x
- 3*y + 3*z + 2*w <= 5 lp_problem += x >=
0
lp_problem += y >= 0 #
Solve the LP problem
lp_problem.solve()

```
# Print the status of the LP problem print("Status:
", LpStatus[lp_problem.status]) # Print the
optimal values of the decision variables
print("Optimal Values:") print("x = ", x.varValue)
print("y = ", y.varValue) print("z = ", z.varValue)
print("w = ", w.varValue)
# Print the optimal value of the objective function
print("Optimal Objective Function Value = ", lpSum([4*x, y, 3*z,
5*w]).getValue())
```

OUTPUT:
Status: Unbounded
Optimal Values: x =
0.83333333 y = 0.0
z = 0.0
w = 5.8333333

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

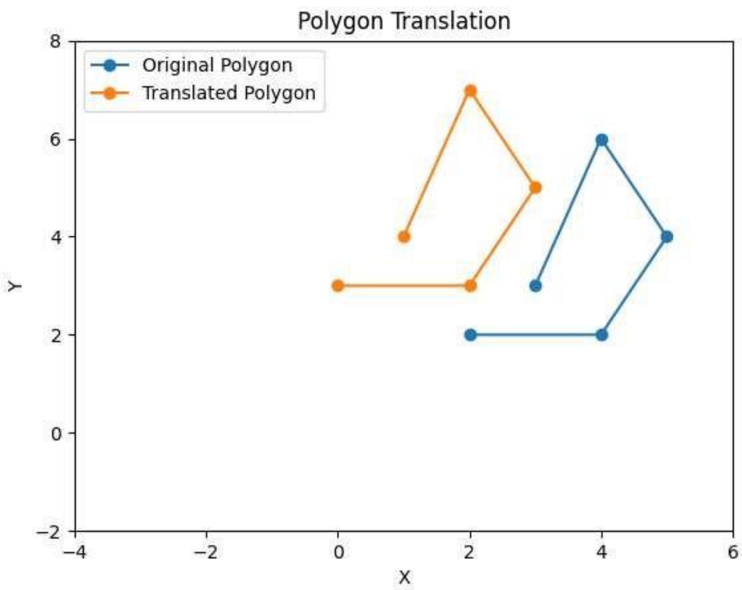
Q.9) Apply each of the following Transformation on the point P[2, -3].
(I)Refection through X-axis.
(II)Scaling in Y-coordinate by factor 1.5.
(III) Shearing in both X and Y direction by -2 and 4 units respectively.
(IV) Rotation about origin by an angle 30 degrees.
Syntax:

```
import numpy as np #
Define the original point P
P = np.array([2, -3])
# (I) Reflection through X-axis
reflection_X = np.array([[1, 0],[0, -1]])
P_reflection_X = np.dot(reflection_X, P) #
(II) Scaling in Y-coordinate by factor 1.5
scaling_Y = np.array([[1, 0],[0, 1.5]])
P_scaling_Y = np.dot(scaling_Y, P)
# (III) Shearing in both X and Y direction by -2 and 4 units respectively
shearing_XY = np.array([[1, -2],[4, 1]])
P_shearing_XY = np.dot(shearing_XY, P)
# (IV) Rotation about origin by an angle of 30 degrees angle
= np.deg2rad(30)
rotation = np.array([[np.cos(angle), -np.sin(angle)],[np.sin(angle),
np.cos(angle)]])
P_rotation = np.dot(rotation, P)
# Print the results print("Original
Point P:", P)
print("Result after reflection through X-axis:", P_reflection_X) print("Result
after scaling in Y-coordinate by factor 1.5:", P_scaling_Y) print("Result after
shearing in both X and Y direction by -2 and 4 units respectively:",
P_shearing_XY)
print("Result after rotation about origin by an angle of 30 degrees:", P_rotation)
```

OUTPUT:
Original Point P: [2 -3]
Result after reflection through X-axis: [2 3]
Result after scaling in Y-coordinate by factor 1.5: [2. -4.5]
Result after shearing in both X and Y direction by -2 and 4 units respectively: [8 5]
Result after rotation about origin by an angle of 30 degrees: [3.23205081 -1.59807621]

Q.10) Write a python program to draw polygon with vertices [3,3],[4,6],[5,4],[4,2] and [2,2] and its translation in x and y direction by factor 2 and 1 respectively Syntax: import matplotlib.pyplot as plt
import numpy as np
# Define the original vertices of the polygon
vertices = np.array([[3, 3],[4, 6],[5, 4],[4, 2],[2, 2]])
# Plot the original polygon
plt.plot(vertices[:, 0], vertices[:, 1], '-o', label='Original Polygon')
# Define the translation matrix
translation_matrix = np.array([[-2, 1]]) #
Perform the translation on the vertices
vertices_translated = vertices + translation_matrix
# Plot the translated polygon
plt.plot(vertices_translated[:, 0], vertices_translated[:, 1], '-o', label='Translated Polygon')
# Set plot title and labels
plt.title('Polygon Translation')
plt.xlabel('X') plt.ylabel('Y')
# Add legend
plt.legend() #
Set plot limits
plt.xlim(-4, 6)
plt.ylim(-2, 8)
# Show the plot
plt.show()

OUTPUT:



Polygon Translation
Original Polygon
Translated Polygon
8
6
4
2
0
−2
Y
−4
−2
0
2
4
6
X

SLIP-18

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

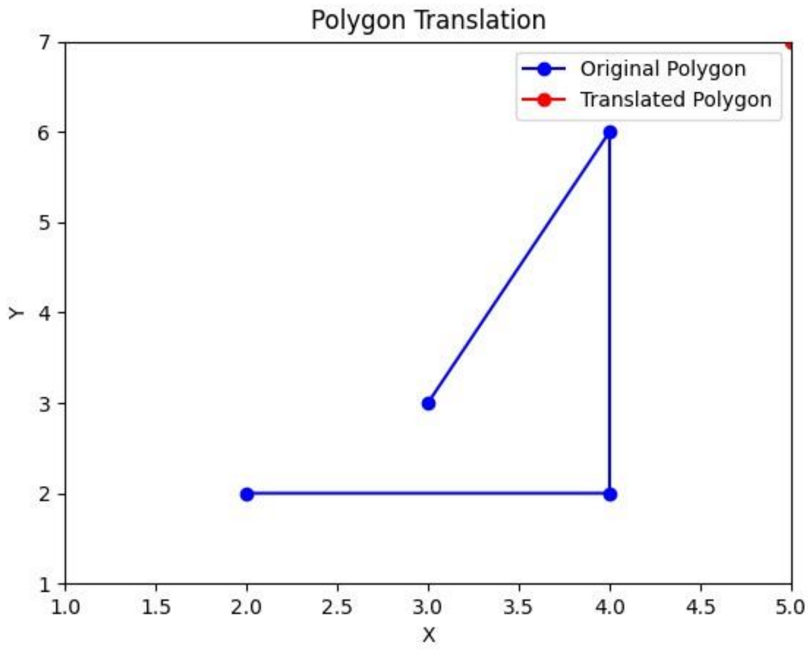
Translation') plt.xlabel('X')

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.

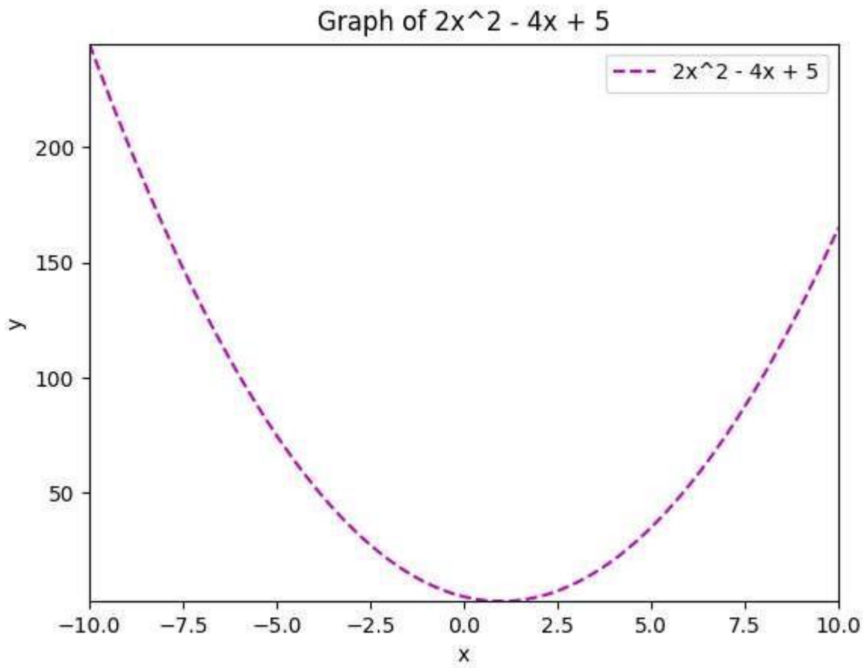
Syntax:

```
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

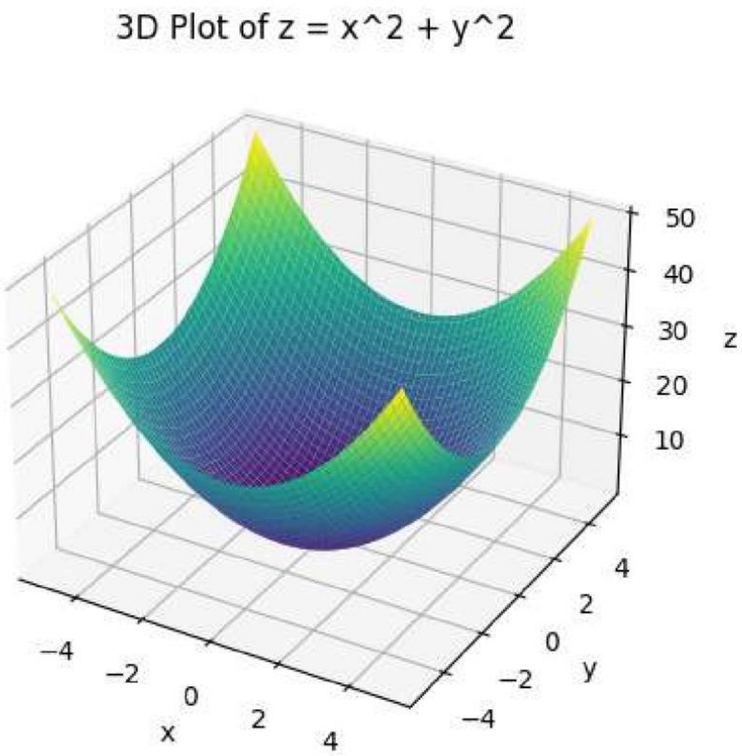
X, Y = np.meshgrid(x, y)

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]
```

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

Q.5)

```
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.]
```

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.

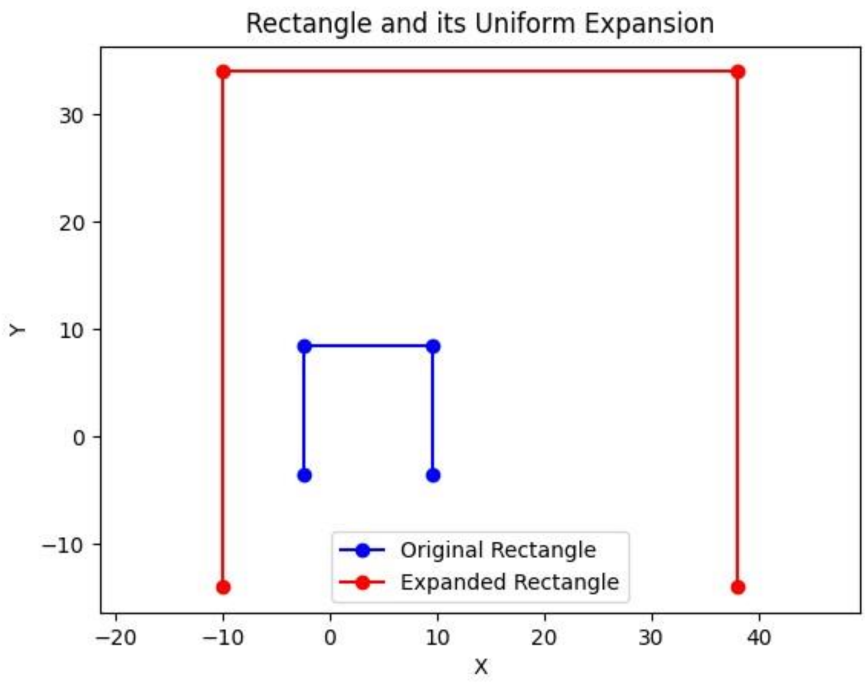
Synatx:

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

Q.6)

```
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:

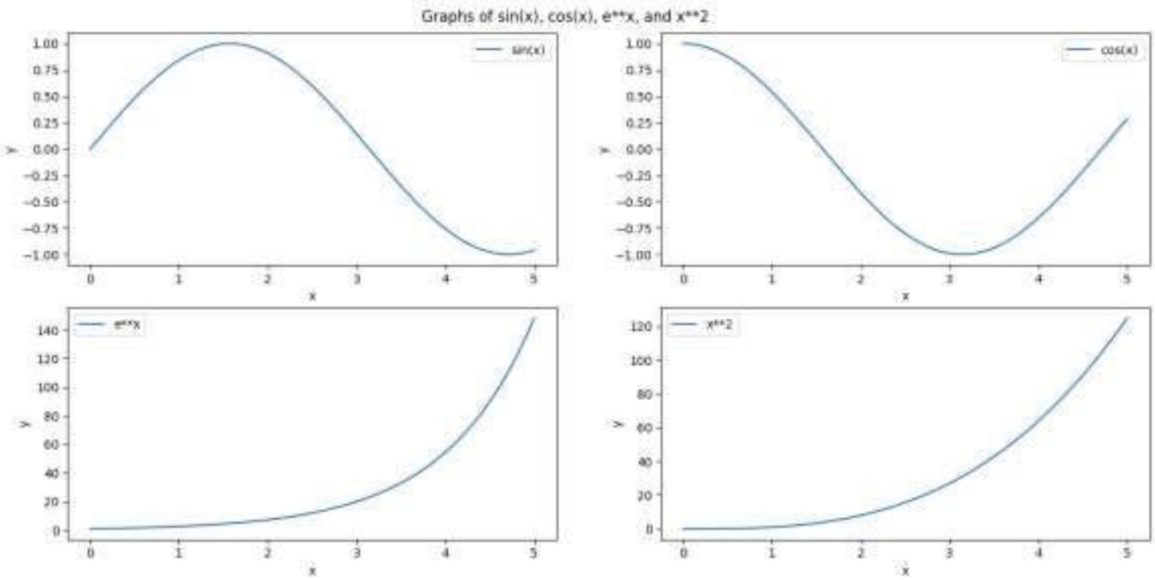
```
Original point P: [ 3 -1]
Transformation I - Reflection through Y-axis:
P1: [-3 -1]
Transformation II - Scaling in X and Y direction:
P2: [ 1.5 -3. ]
Transformation III - Shearing in X and Y direction:
P3: [ 5 11]
Transformation IV - Rotation about origin by 60 degrees:
P4: [2.3660254  2.09807621]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

SLIP-19

Q.1) Plot the graphs of sin x, cos x, e**x and x**3 in [O, 5] in one figure with (2 x 2) subplot Syntax: import numpy as np import matplotlib.pyplot as plt

# Generate x values x =
np.linspace(0, 5, 500)
# Compute y values for sin(x), cos(x), e**x, x**2 y1
= np.sin(x) y2 = np.cos(x) y3 = np.exp(x) y4 = x**3
# Create subplots fig, axs = plt.subplots(2, 2,
figsize=(10, 8)) fig.suptitle('Graphs of sin(x), cos(x),
e**x, and x**2')
# Plot sin(x) axs[0, 0].plot(x, y1,
label='sin(x)') axs[0, 0].legend() #
Plot cos(x) axs[0, 1].plot(x, y2,
label='cos(x)') axs[0, 1].legend() #
Plot e**x axs[1, 0].plot(x, y3,
label='e**x') axs[1, 0].legend()
# Plot x**2 axs[1, 1].plot(x, y4,
label='x**2') axs[1, 1].legend()
# Set x and y axis labels for all subplots
for ax in axs.flat: ax.set_xlabel('x')
ax.set_ylabel('y')
# Adjust spacing between subplots
fig.tight_layout() # Show the plot
plt.show()

OUTPUT:



Q.2) Write a python program to plot 30 Surface Plot of the function z = cos(|x| +|y|) in -1 < x,y1 < 1.

Syntax:

import numpy as np import

matplotlib.pyplot as plt

Generate 30 random x, y pairs within the range -1 < x, y < 1

np.random.seed(0) x_vals = np.random.uniform(-1, 1, size=30) y_vals = np.random.uniform(-1, 1, size=30)

Create a 2D grid of x, y values x, y = np.meshgrid(np.linspace(-1, 1, 100), np.linspace(-1, 1, 100))

Compute the z values for each x, y pair z = np.cos(np.abs(x) + np.abs(y))

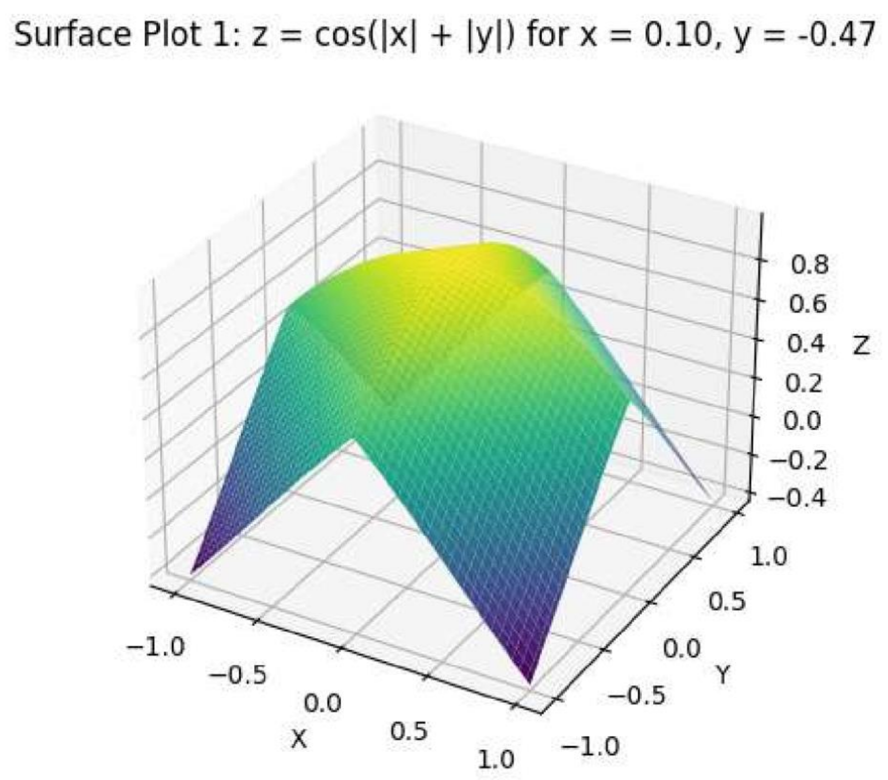
# Plot the surface plots for i in range(30): fig = plt.figure() ax = fig.add_subplot(111, projection='3d') ax.plot_surface(x, y, z, cmap='viridis') ax.set_xlabel('X') ax.set_ylabel('Y') ax.set_zlabel('Z') ax.set_title(f'Surface Plot {i+1}: z = cos(|x| + |y|) for x = {x_vals[i]:.2f}, y = {y_vals[i]:.2f}')
plt.show()

OUTPUT:

Q.3) Write a python program to plot 2D graph of the functions f(x) = log(x) + 5 and g(x) = log(x) – 5 in [0, 10] by setting different line width and different colors to the curve.

Syntax:

import numpy as np import
matplotlib.pyplot as plt #
Define the functions def f(x):
    return np.log(x) + 5 def
g(x):
    return np.log(x) - 5
Generate x values in the range [0, 10] x = np.linspace(0.01, 10, 100) # Compute y values for f(x) and g(x) y_f = f(x) y_g = g(x)



Create the plot plt.plot(x, y_f, label='f(x) = log(x) + 5', linewidth=2, color='blue') plt.plot(x, y_g, label='g(x) = log(x) - 5', linewidth=1, color='red')

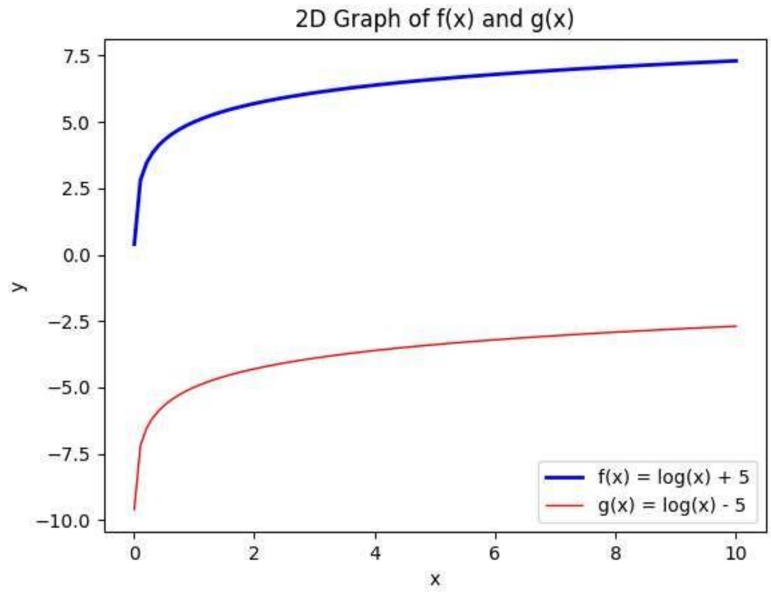
Add labels and legend

plt.xlabel('x')

plt.ylabel('y') plt.legend()

Set the title and show the plot plt.title('2D Graph of f(x) and g(x)') plt.show()

OUTPUT:



Q.4) Write a python program to rotate the segment by 90° having endpoints (0,0) and (4,4)

Syntax:

import math
# Define the endpoints of the line segment
x1, y1 = 0, 0 x2, y2 = 4, 4 # Perform the
rotation x1_rotated = -x1 y1_rotated = -y1
x2_rotated = -x2 y2_rotated = -y2
# Print the original and rotated endpoints print("Original Endpoint 1: ({}, {})".format(x1, y1)) print("Original Endpoint 2: ({}, {})".format(x2, y2)) print("Rotated Endpoint 1: ({}, {})".format(x1_rotated, y1_rotated)) print("Rotated Endpoint 2: ({}, {})".format(x2_rotated, y2_rotated))

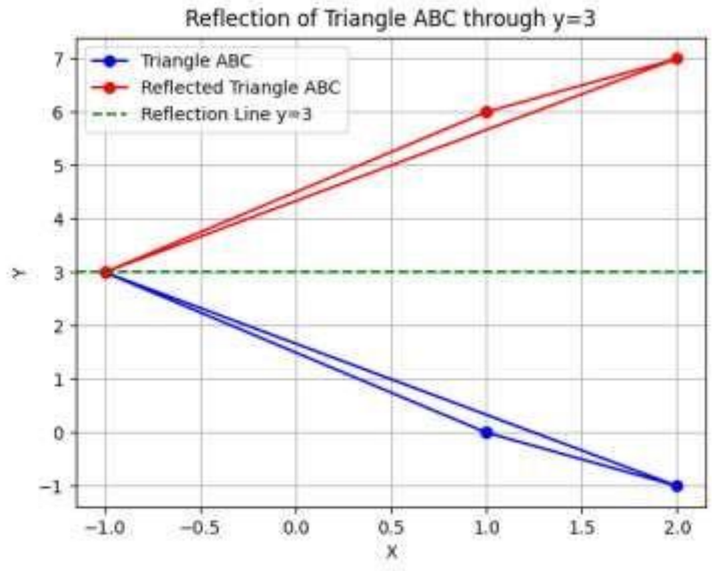
Output:

Original Endpoint 1: (0, 0)
Original Endpoint 2: (4, 4)
Rotated Endpoint 1: (0, 0)
Rotated Endpoint 2: (-4, -4)

Q.5) Write a Python program to Ref1ect the triangle ABC through the line y=3, where A[1, 0], B[2, -1] and C[-1, 3] Syntax:

```
import numpy as np import
matplotlib.pyplot as plt
# Define the coordinates of the triangle ABC
A = np.array([1, 0])
B = np.array([2, -1])
C = np.array([-1, 3])
# Define the equation of the reflection line y = 3 reflection_line
= 3
# Reflect the triangle ABC through the reflection line
A_reflected = np.array([A[0], 2*reflection_line - A[1]])
B_reflected = np.array([B[0], 2*reflection_line - B[1]])
C_reflected = np.array([C[0], 2*reflection_line - C[1]])
# Plot the original and reflected triangles
plt.plot([A[0], B[0], C[0], A[0]], [A[1], B[1], C[1], A[1]], 'bo-', label='Triangle
ABC')
plt.plot([A_reflected[0], B_reflected[0], C_reflected[0], A_reflected[0]],
[A_reflected[1], B_reflected[1], C_reflected[1], A_reflected[1]], 'ro-',
label='Reflected Triangle ABC') plt.axhline(y=reflection_line, color='g',
linestyle='--', label='Reflection Line y=3') plt.xlabel('X') plt.ylabel('Y')
plt.legend() plt.title('Reflection of Triangle ABC through y=3') plt.grid(True)
plt.show()
```

Output:



Q.6) Write a Python program to draw a polygon with vertices (0,0),(1,0),(2,2),(1,4). Also find area and perimeter of the polygon.

Synatx: import

matplotlib.pyplot as plt

Define the coordinates of the vertices of the polygon

vertices = [(0, 0), (1, 0), (2, 2), (1, 4)] # Extract the x

and y coordinates of the vertices x = [vertex[0] for

vertex in vertices] y = [vertex[1] for vertex in

vertices]

Plot the polygon plt.plot(x + [x[0]], y + [y[0]],

'bo-', label='Polygon') plt.xlabel('X') plt.ylabel('Y')

plt.legend() plt.title('Polygon with Vertices')

plt.grid(True) plt.show()

# Calculate the area of the polygon using shoelace formula
area = 0 for i in

range(len(vertices)):

    area += (x[i] * y[(i + 1) % len(vertices)]) - (x[(i + 1) % len(vertices)] * y[i])

area /= 2 area = abs(area)

Calculate the perimeter of the polygon

perimeter = 0 for i in

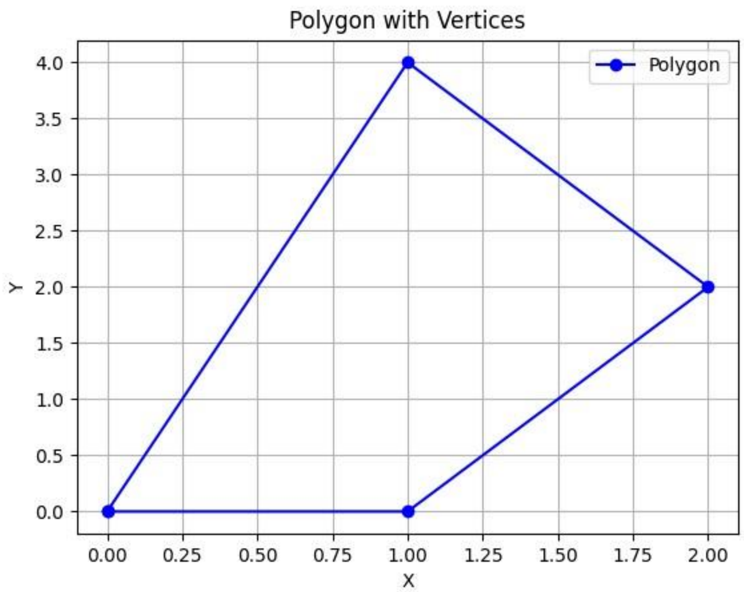
range(len(vertices)):

dx = x[(i + 1) % len(vertices)] - x[i] dy = y[(i + 1) % len(vertices)] - y[i] perimeter += ((dx ** 2) + (dy ** 2)) ** 0.5 # Print the area and perimeter of the polygon
print("Area of the Polygon:", area)
print("Perimeter of the Polygon:", perimeter)

OUTPUT:

Area of the Polygon: 4.0

Perimeter of the Polygon: 9.595241580617241



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

Max Z = x + 2y + z
Subjected to
X + 0.5y + 0.5z <= 1
1.5x + 2y + z >= 8
x > 0 , y > 0
Syntax:

from scipy.optimize import linprog #
Coefficients of the objective function c
= [1, 2, 1]
# Coefficients of the inequality constraints (LHS matrix)
A = [[1, 0.5, 0.5],
    [-1.5, -2, -1]]
# RHS values of the inequality constraints
b = [1, -8]

Bounds on the variables (x, y, z) bounds
= [(0, None), (0, None), (0, None)]
# Specify the inequality constraint directions (<=, >=)
# and the corresponding bound values (1, -1)
ineq_ops = ['<=', '>=']
# Solve the linear programming problem
res = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='simplex')
# Print the results
print("Optimization Result:")
print("Objective Value (Z):", res.fun)
print("Optimal Solution (x, y, z):", res.x)
OUTPUT:
Optimization Result:
Objective Value (Z): 4.0
Optimal Solution (x, y, z): [0. 2. 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 = 3x+5y + 4z subject
to
2x+ 3y <= 8
2y + 5z <= 10   3x
+ 2y + 4z <= 15
x>=0,y>=0,z>=0
Syntax: from pulp
import *
# Create a minimization problem
prob = LpProblem("Minimization Problem", LpMinimize)
# Define 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 +
5*y + 4*z, "Z" # Define the constraints prob
+= 2*x + 3*y <= 8, "Constraint 1" prob +=
2*y + 5*z <= 10, "Constraint 2" prob += 3*x +
2*y + 4*z <= 15, "Constraint 3"
Solve the problem prob.solve()
Print the status of the problem print("Status:",
LpStatus[prob.status])
# Print the optimal solution
print("Optimal Solution:")
print("x =", value(x))

```
print("y =", value(y)) print("z
=", value(z))
# Print the optimal objective value
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) Rotation about origin through an angle 48 degree
(II) Scaling in X – coordinate by 2 factor
(III) Reflection through the line y = 2x - 3
(IV) Shearing in X Direction by 7 units

Syntax:
```
import
numpy as np
# Define the initial point point
= np.array([-2, 4])
# Transformation 1: Rotation about origin through an angle of 48 degrees angle
= np.deg2rad(48)
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                           [np.sin(angle), np.cos(angle)]]) rotated_point
= np.dot(rotation_matrix, point)
# Transformation 2: Scaling in X-coordinate by a factor of 2 scaling_factor
= np.array([[2, 0],
                          [0, 1]])
scaled_point = np.dot(scaling_factor, rotated_point) #
Transformation 3: Reflection through the line y = 2x - 3
reflection_matrix = np.array([[1, -4],
                             [-4, 1]])
reflected_point = np.dot(reflection_matrix, scaled_point) #
Transformation 4: Shearing in X-direction by 7 units
shearing_factor = np.array([[1, 7],
                           [0, 1]])
sheared_point = np.dot(shearing_factor, reflected_point)
# Print the results print("Initial Point:",
point) print("Rotated Point:",
rotated_point) print("Scaled Point:",
scaled_point) print("Reflected Point:",
```

reflected_point) print("Sheared Point:",
sheared_point)
OUTPUT:
Initial Point: [-2 4]
Rotated Point: [-4.31084051 1.19023277]
Scaled Point: [-8.62168103 1.19023277]
Reflected Point: [-13.38261213 35.67695689]
Sheared Point: [236.35608611 35.67695689]

Q.10) Find Combined transformation of the line segment between the points A[4,-1] and B[3,0] for the following sequence :
First rotation about origin through an angle pi; followed by scaling in x coordinate by 3 units. Followed by reflection through the line y = x; Syntax: import numpy as np

```
# Define the initial points A and B
A = np.array([4, -1])
B = np.array([3, 0])
# Transformation 1: Rotation about origin through an angle pi (180 degrees)
angle = np.pi
rotation_matrix = np.array([[np.cos(angle), -np.sin(angle)],
                            [np.sin(angle), np.cos(angle)]])
rotated_A = np.dot(rotation_matrix, A) rotated_B
= np.dot(rotation_matrix, B)
# Transformation 2: Scaling in x-coordinate by 3 units scaling_factor
= np.array([[3, 0],
            [0, 1]])
scaled_A = np.dot(scaling_factor, rotated_A)
scaled_B = np.dot(scaling_factor, rotated_B) #
Transformation 3: Reflection through the line y = x
reflection_matrix = np.array([[0, 1],
                              [1, 0]])
reflected_A = np.dot(reflection_matrix, scaled_A) reflected_B
= np.dot(reflection_matrix, scaled_B)
# Print the results print("Initial
Points A and B:") print("A =",
A) print("B =", B)
print("Combined Transformed Points A and B:")
print("A' =", reflected_A)
print("B' =", reflected_B)
```

OUTPUT:
Initial Point: [-2 4]
Rotated Point: [-4.31084051 1.19023277]
Scaled Point: [-8.62168103 1.19023277]

Reflected Point: [-13.38261213 35.67695689]
Sheared Point: [236.35608611 35.67695689]

SLIP-20

Q.1) Write a Python program to plot 2D graph of the function f(x)= sin (x) and g(x) = cos (x) in [-2*pi,2*pi] Syntax: import numpy as np import matplotlib.pyplot as plt # Define the range of x values x = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

Compute the y values for f(x) = sin(x) and g(x) = cos(x)

f_x = np.sin(x) g_x = np.cos(x)

Create a figure and axis fig, ax = plt.subplots() # Plot f(x) = sin(x)

ax.plot(x, f_x, label='f(x) = sin(x)')

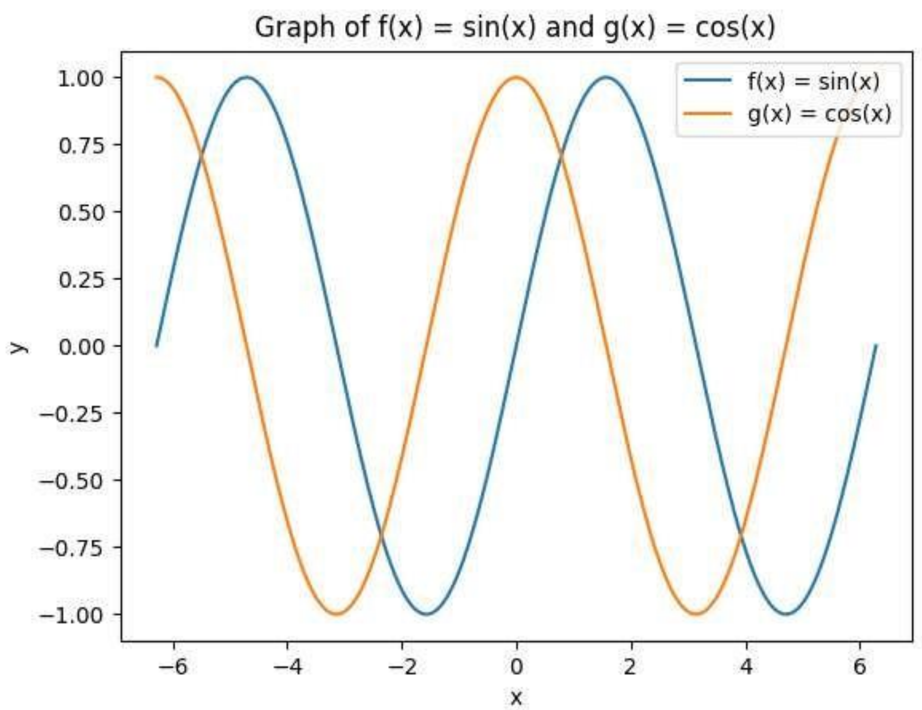
Plot g(x) = cos(x) ax.plot(x, g_x, label='g(x) = cos(x)') # Set the title and labels for x and y axes

ax.set_title('Graph of f(x) = sin(x) and g(x) = cos(x)')

ax.set_xlabel('x') ax.set_ylabel('y') # Add a legend

ax.legend() # Show the plot plt.show()

OUTPUT:

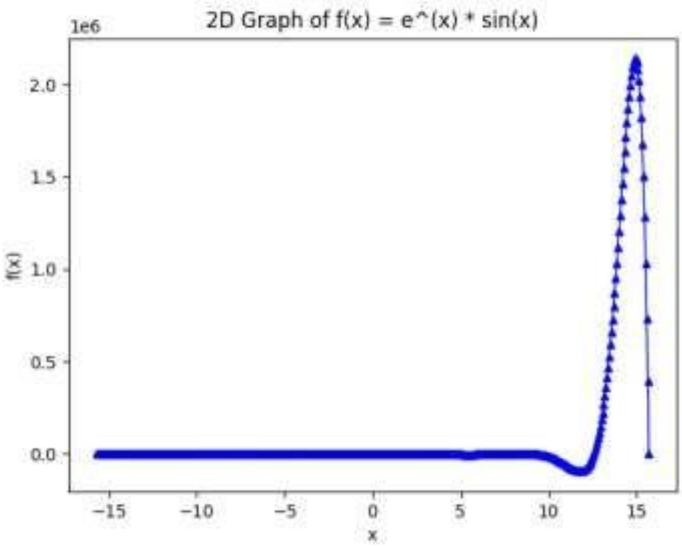


Q.2) Write n Python program to plot the 2D graph of the function f(x)=e(x)sin(x) in [-5*pi,5*pi] with blue points line with upward point line with upward pointing triangle.

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt #
Define the function f(x) def
f(x):
    return np.exp(x) * np.sin(x)
# Generate x values in the range [-5*pi, 5*pi]
x = np.linspace(-5*np.pi, 5*np.pi, 500) #
Calculate y values using the function f(x) y =
f(x)
# Plot the graph with blue points and a line with upward pointing triangles
plt.plot(x, y, 'b^-', linewidth=1, markersize=4)
# Set x and y axis labels
plt.xlabel('x') plt.ylabel('f(x)')
# Set the title of the graph plt.title('2D
Graph of f(x) = e^(x) * sin(x)')
# Show the graph plt.show()
```

OUTPUT:



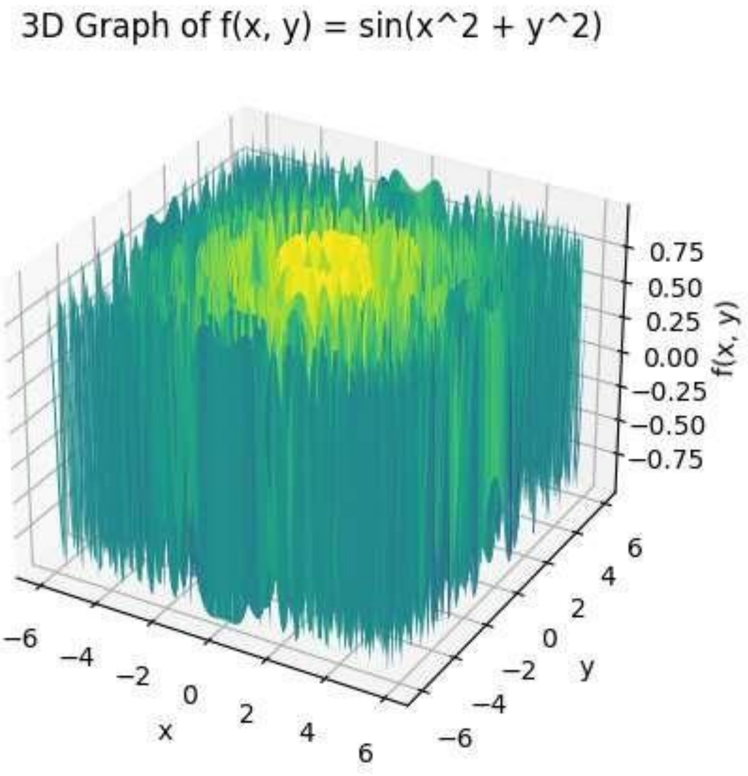
Q.3) Write a Python program to plot the 3D graph of the function f(x) = sin(x^2 + y^2), -6< x,y < 6.

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
# Define the function f(x, y) def
f(x, y):
    return np.sin(x**2 + y**2)
# Generate x, y values in the range -6 to 6 with a step of 0.1 x
= np.arange(-6, 6, 0.1)
y = np.arange(-6, 6, 0.1) # Create a
meshgrid from x, y values
X, Y = np.meshgrid(x, y)
# Calculate z values using the function f(x, y)
Z = f(X, Y) # Create a 3D figure fig =
plt.figure() ax = fig.add_subplot(111,
projection='3d')
# Plot the 3D surface ax.plot_surface(X,
Y, Z, cmap='viridis')
# Set x, y, z axis labels ax.set_xlabel('x')
ax.set_ylabel('y') ax.set_zlabel('f(x, y)') # Set the
title of the graph ax.set_title('3D Graph of f(x, y)
= sin(x^2 + y^2)')
```

Show the graph

OUTPUT:



plt.show()

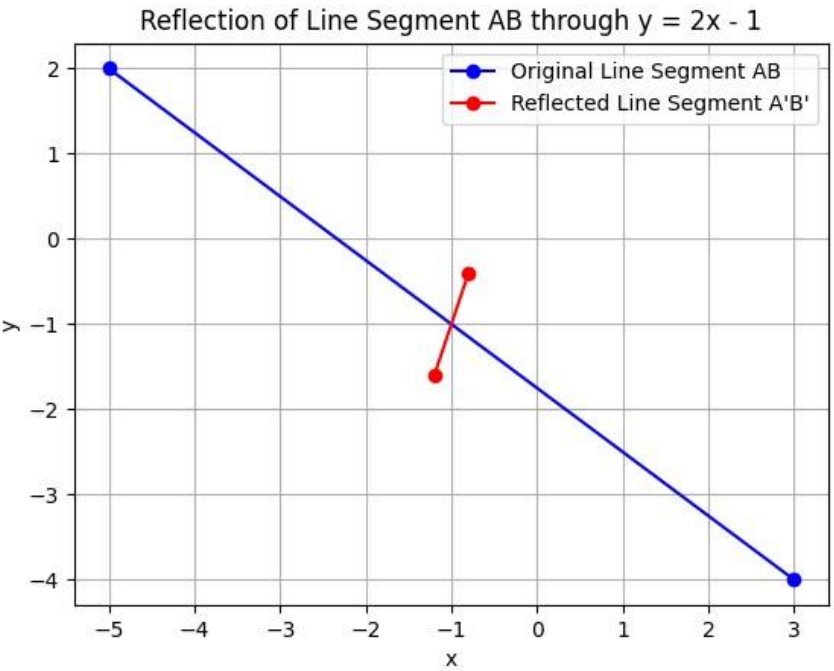
Q.4) Write a python program to reflect the line segment joining the points A[-5, 2], B[3, -4] through the line y = 2x - 1.

Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the line segment endpoints A and B
A = np.array([-5, 2])
B = np.array([3, -4])
# Define the reflection line y = 2x - 1 m = 2  #
slope of the reflection line c = -1  # y-intercept
of the reflection line # Calculate the midpoint of
the line segment AB midpoint = (A + B) / 2
# Calculate the direction vector of the reflection line direction
= np.array([1, m])
# Calculate the projection of the midpoint onto the reflection line
projection = (2 * midpoint.dot(direction) - 2 * c * direction) / (1 + m**2) #
Calculate the reflected point of A with respect to the reflection line
reflected_A = midpoint + (projection - midpoint)
# Calculate the reflected point of B with respect to the reflection line
reflected_B = midpoint - (projection - midpoint)
```

# Plot the original line segment AB and the reflected line segment A'B' plt.plot([A[0], B[0]], [A[1], B[1]], 'bo-', label='Original Line Segment AB') plt.plot([reflected_A[0], reflected_B[0]], [reflected_A[1], reflected_B[1]], 'ro-', label='Reflected Line Segment A\'B\'')
plt.xlabel('x')
plt.ylabel('y') plt.legend()
plt.title('Reflection of Line Segment AB through y = 2x - 1')
plt.grid() plt.show()

Output:



Q.5) Write a Python program lo find the area and perimeter of a polygon with vertices (0, 0), (-2, 0), (5,5), (1, -1)

Syntax:

import math
# Define the vertices of the polygon
vertices = [(0, 0), (-2, 0), (5, 5), (1, -1)]
# Calculate the area of the polygon using Shoelace formula def calculate_area(vertices):
area = 0 for i in range(len(vertices)): x1, y1 = vertices[i]

```
        x2, y2 = vertices[(i + 1) % len(vertices)]
area += (x1 * y2 - x2 * y1)     return
abs(area) / 2
# Calculate the perimeter of the polygon def
calculate_perimeter(vertices):
   perimeter = 0     for i in
range(len(vertices)):
        x1, y1 = vertices[i]
        x2, y2 = vertices[(i + 1) % len(vertices)]
        perimeter += math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
   return perimeter
# Call the functions to calculate area and perimeter area
= calculate_area(vertices)
perimeter = calculate_perimeter(vertices)
# Print the results
print("Area of the polygon: ", area) print("Perimeter
of the polygon: ", perimeter) OUTPUT:
Area of the polygon:  10.0
Perimeter of the polygon:  19.2276413803437
```

Q.6) Write a. Python program to plot the 3D graph of the function f(x, y) = sin x + cos y,   x, y belongs [-2*pi,2*pi] using wireframe plot.

```
import numpy as np import

matplotlib.pyplot as plt from

mpl_toolkits.mplot3d import Axes3D #

Define the range of x and y values x =

np.linspace(-2 * np.pi, 2 * np.pi, 100) y =

np.linspace(-2 * np.pi, 2 * np.pi, 100)

# Create a meshgrid from x and y

X, Y = np.meshgrid(x, y)

# Calculate the Z values using the function f(x, y) = sin(x) + cos(y)

Z = np.sin(X) + np.cos(Y) # Create a 3D

plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

# Create a wireframe plot

ax.plot_wireframe(X, Y, Z) #

Set labels and title
```

ax.set_xlabel('X')

ax.set_ylabel('Y')

ax.set_zlabel('Z')

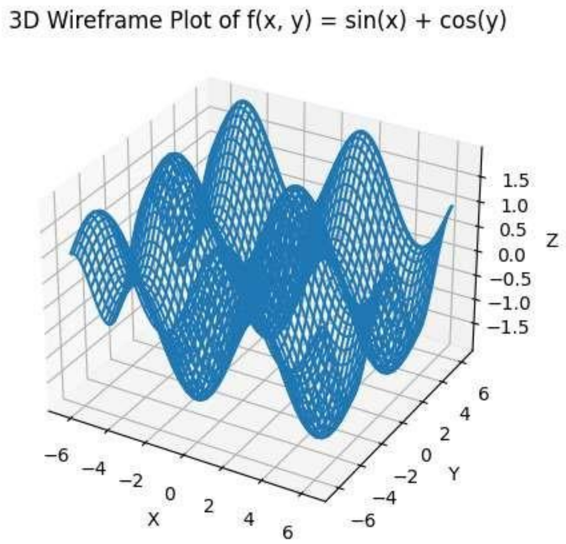
ax.set_title('3D Wireframe

Plot of f(x, y) = sin(x) +

cos(y)')

Show the plot plt.show()

OUTPUT:



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

Max Z = 3.5x +2 y

Subjected to x + y

>= 5 x >= 4 y <=

2 x > 0 , y > 0

Syntax:

import numpy as np from

scipy.optimize import linprog #

Coefficients of the objective function c

= [-3.5, -2]

Coefficients of the inequality constraints

A = [[-1, -1], [-1, 0], [0, 1]]
b = [-5, -4, 2]

Bounds on the variables

x_bounds = (0, None)

y_bounds = (0, None)

Solve the linear programming problem result = linprog(c, A_ub=A,

b_ub=b, bounds=[x_bounds, y_bounds]) if result.success:

print("Optimal solution found:")

print("x =", result.x[0]) print("y =",

result.x[1]) print("Maximum value of Z

=", -result.fun) else:

print("Optimal solution not found.")

OUTPUT:

Optimal solution not found.

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 - y >= 1 x
+ y >= 2
x>= 0, y>= 0

Syntax:

```
from pulp import *
# Create a LP Minimization problem problem
= LpProblem("LPP", LpMinimize)
# Define the decision variables x =
LpVariable("x", lowBound=0)  # x >= 0 y =
LpVariable("y", lowBound=0)  # y >= 0
# Define the objective function
problem += x + y #
Define the constraints
problem += x - y >= 1
problem += x + y >= 2
# Solve the problem using the simplex method status
= problem.solve()
# Check if the problem has an optimal solution
if status == LpStatusOptimal:     # Print the
```

optimal solution   print("Optimal Solution:")
print("x =", value(x))   print("y =", value(y))
print("Z =", value(problem.objective)) else:
    print("No Optimal Solution")
OUTPUT:
Optimal Solution:
x = 2.0 y = 0.0 Z
= 2.0

Q.9) Apply Python. Program in each of the following transformation on the point P[3,-2]

(I) Scaling in y direction by 4 unit.
(II) Reflection through y axis.
(III) Rotation about origin by angle 45° .
(IV) Reflection through the line y = x.

Syntax: import
numpy as np #
Given point P
P = np.array([3, -2])
# Transformation I: Scaling in y direction by 4 units
scaling_matrix = np.array([[1, 0], [0, 4]]) P_scaled_y =
np.dot(scaling_matrix, P) print("After Scaling in y
direction by 4 units:") print("Point P_scaled_y:",
P_scaled_y) # Transformation II: Reflection through y
axis reflection_y_axis_matrix = np.array([[-1, 0], [0, 1]])
P_reflected_y_axis = np.dot(reflection_y_axis_matrix, P)
print("After Reflection through y axis:")
print("Point P_reflected_y_axis:", P_reflected_y_axis) #
Transformation III: Rotation about origin by angle 45 degrees
angle_rad = np.deg2rad(45) # Convert angle to radians
rotation_matrix = np.array([[np.cos(angle_rad), -np.sin(angle_rad)],
[np.sin(angle_rad), np.cos(angle_rad)]]) P_rotated
= np.dot(rotation_matrix, P)
print("After Rotation about origin by angle 45 degrees:") print("Point
P_rotated:", P_rotated)
# Transformation IV: Reflection through the line y = x
reflection_line_matrix = np.array([[0, 1], [1, 0]])
P_reflected_line = np.dot(reflection_line_matrix, P)
print("After Reflection through the line y = x:") print("Point
P_reflected_line:", P_reflected_line)

OUTPUT:
Point P_scaled_y: [3 -8]
After Reflection through y axis:

Point P_reflected_y_axis: [-3 -2]
After Rotation about origin by angle 45 degrees:
Point P_rotated: [3.53553391 0.70710678] After
Reflection through the line y = x:
Point P_reflected_line: [-2 3]

Q.10) Apply the following transformation on the point P[3, -2] (I)
Shearing in x direction by -2 units.
(II) Scaling in X and y direction by -2 and 2 units respectively
(III) Reflection through x axis. (IV) Reflection through the
line y = -x Syntax:

```
import numpy as np #
Given point P
P = np.array([3, -2])
# Transformation I: Shearing in x direction by -2 units
shearing_x_matrix = np.array([[1, -2], [0, 1]])
P_sheared_x = np.dot(shearing_x_matrix, P) print("After
Shearing in x direction by -2 units:") print("Point
P_sheared_x:", P_sheared_x)
# Transformation II: Scaling in x and y direction by -2 and 2 units respectively
scaling_matrix = np.array([[-2, 0], [0, 2]]) P_scaled_xy =
np.dot(scaling_matrix, P)
print("After Scaling in x and y direction by -2 and 2 units respectively:")
print("Point P_scaled_xy:", P_scaled_xy) # Transformation III: Reflection
through x axis reflection_x_axis_matrix = np.array([[1, 0], [0, -1]])
P_reflected_x_axis = np.dot(reflection_x_axis_matrix, P) print("After
Reflection through x axis:") print("Point P_reflected_x_axis:",
P_reflected_x_axis) # Transformation IV: Reflection through the line y = -x
reflection_line_matrix = np.array([[0, -1], [-1, 0]]) P_reflected_line =
np.dot(reflection_line_matrix, P) print("After Reflection through the line y = -
x:") print("Point P_reflected_line:", P_reflected_line)
```

OUTPUT:
Line segment after applying the sequence of transformations:
After Scaling in y direction by 4 units: Point
P_scaled_y: [3 -8]
After Reflection through y axis:
Point P_reflected_y_axis: [-3 -2]
After Rotation about origin by angle 45 degrees:
Point P_rotated: [3.53553391 0.70710678] After
Reflection through the line y = x:
Point P_reflected_line: [-2 3]
PS E:\Python 2nd Sem Practical> python -u "e:\Python 2nd Sem
Practical\tempCodeRunnerFile.py"

After Shearing in x direction by -2 units:
Point P_sheared_x: [7 -2]
After Scaling in x and y direction by -2 and 2 units respectively:
Point P_scaled_xy: [-6 -4] After Reflection through x axis:
Point P_reflected_x_axis: [3 2]
After Reflection through the line y = -x:
Point P_reflected_line: [2 -3]

SLIP-21

Q.1) Plot the graph of f(x) = x**4 in [0, 5] with red dashed line with circle markers.

Syntax: import numpy as np

import matplotlib.pyplot as plt #

Define the function f(x) = x**4

def f(x):

    return x**4

Generate x values in the interval [0, 5] x

= np.linspace(0, 5, 100)

Generate y values using the function f(x) y

= f(x)

Plot the graph with red dashed line and circle markers plt.plot(x,

y, 'r--o', markersize=6)

Set x-axis label

plt.xlabel('x') # Set y-axis

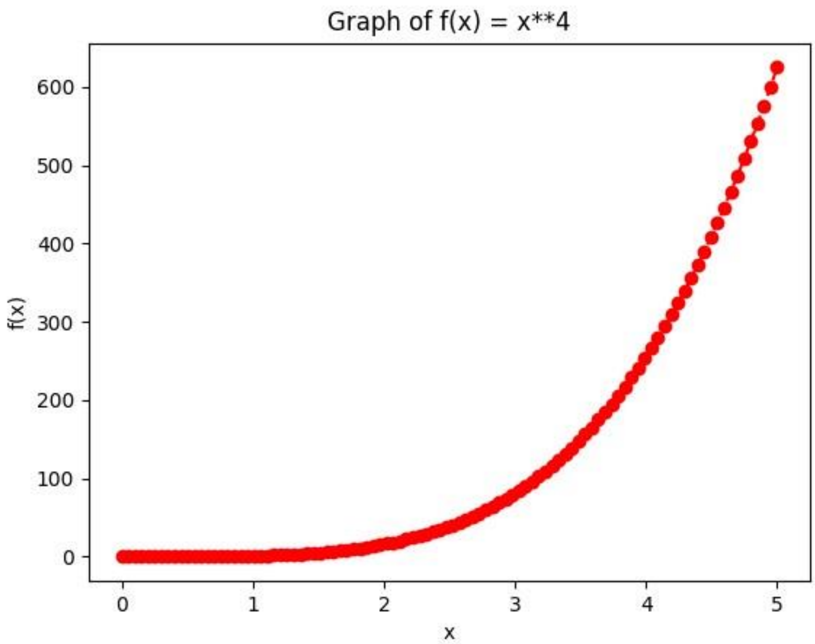
label plt.ylabel('f(x)') # Set

title plt.title('Graph of f(x) =

x**4')

Show the plot plt.show()

OUTPUT:



Q.2) Write a Python program to plot the 3D graph of the function f(x) = sin(x^2 + y^2), -6< x,y < 6.

Syntax:

```
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
# Define the function f(x, y) def
f(x, y):
   return np.sin(x**2 + y**2)
# Generate x, y values in the range -6 to 6 with a step of 0.1
x = np.arange(-6, 6, 0.1) y = np.arange(-6, 6, 0.1)
# Create a meshgrid from x, y values
X, Y = np.meshgrid(x, y)
# Calculate z values using the function f(x, y)
Z = f(X, Y)
# Create a 3D figure
fig = plt.figure() ax =
fig.add_subplot(111, projection='3d')
# Plot the 3D surface ax.plot_surface(X,
Y, Z, cmap='viridis')
```

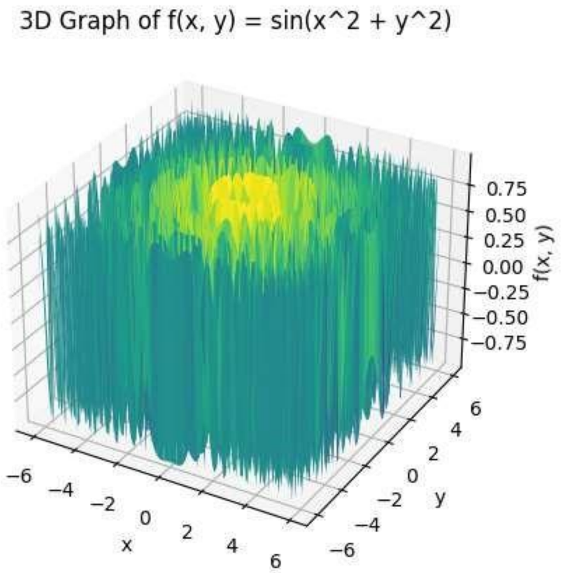
Set x, y, z axis labels ax.set_xlabel('x')

ax.set_ylabel('y') ax.set_zlabel('f(x, y)') # Set the

title of the graph ax.set_title('3D Graph of f(x, y)

= sin(x^2 + y^2)')

#



Show the graph plt.show()

OUTPUT:

Q.3) Write a Python program to plot the 3D graph of the function f(x) = e(x^2+y^2) for x, y belongs [0, 2*pi] using wireframe.

Syntax:

import numpy as np import

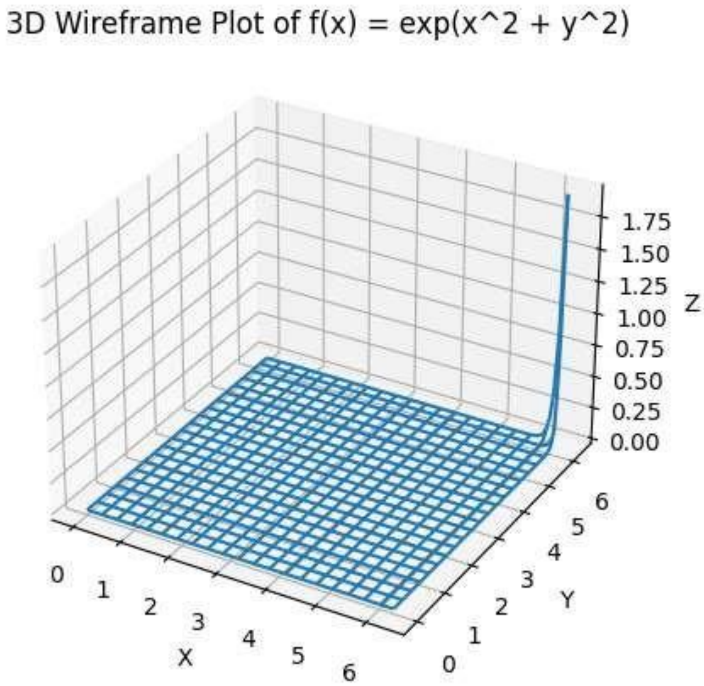
matplotlib.pyplot as plt from

mpl_toolkits.mplot3d import Axes3D

Define the function f(x, y)

```
def f(x, y):
    return np.exp(x**2 + y**2) #
Generate x, y values x =
np.linspace(0, 2*np.pi, 100) y =
np.linspace(0, 2*np.pi, 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')
# Create a wireframe plot ax.plot_wireframe(X,
Y, Z, rstride=5, cstride=5)
# Set axis labels ax.set_xlabel('X') ax.set_ylabel('Y')
ax.set_zlabel('Z') ax.set_title('3D Wireframe Plot of f(x)
= exp(x^2 + y^2)')
# Show the plot plt.show()
```

OUTPUT:



Q.4) if the line segment joining the points A[2,5] and [4,-13] is transformed to 2
3 the using python the line segment A'B' by the transformation matrix [T] =
4 1

find the slope and midpoint of the transformed line.

Syntax:

```
import numpy as np
# Define the original line segment points A and B
A = np.array([2, 5])
B = np.array([4, -13])
# Define the transformation matrix [T]
T = np.array([[2, 3], [4, 1]])
# Apply the transformation matrix [T] to points A and B
A_transformed = np.dot(T, A)
B_transformed = np.dot(T, B)
# Calculate the slope of the transformed line
slope_transformed = (B_transformed[1] - A_transformed[1]) /
(B_transformed[0] - A_transformed[0]) #
Calculate the midpoint of the transformed line
midpoint_transformed = (A_transformed + B_transformed) / 2 #
Print the slope and midpoint of the transformed line print("Slope
of the transformed line: ", slope_transformed) print("Midpoint of
the transformed line: ", midpoint_transformed)
```

Output:

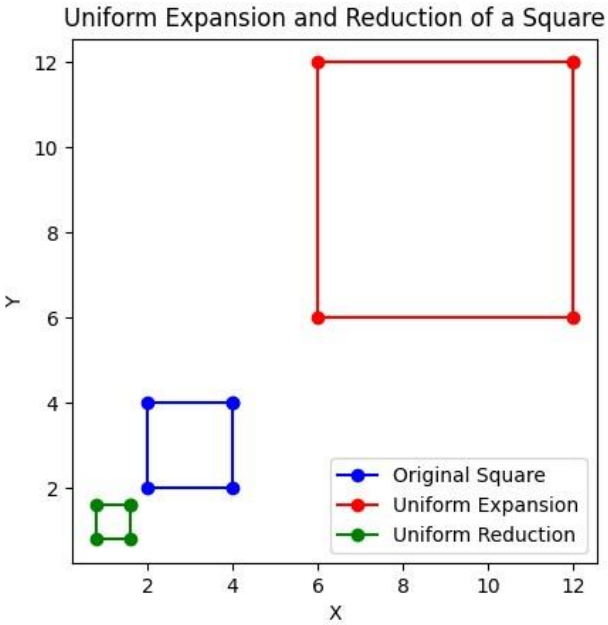
```
Slope of the transformed line:  0.2
Midpoint of the transformed line:  [-6.  8.]
```

Q.5) Write a python program to plot square with vertices at [4, 4] [2, 4], [2, 2], [4, 2] and find its uniform expansion by factor 3, uniform reduction by factor 0.4. Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the vertices of the original square vertices =
np.array([[4, 4], [2, 4], [2, 2], [4, 2], [4, 4]]) # Create
a figure and axis fig, ax = plt.subplots() # Plot the
original square
ax.plot(vertices[:, 0], vertices[:, 1], 'b-o', label='Original Square')
# Define the uniform expansion and reduction factors
expansion_factor = 3 reduction_factor = 0.4 #
Perform uniform expansion
expanded_vertices = vertices * expansion_factor
# Perform uniform reduction reduced_vertices
= vertices * reduction_factor # Plot the
expanded and reduced squares
```

ax.plot(expanded_vertices[:, 0], expanded_vertices[:, 1], 'r-o', label='Uniform Expansion')
ax.plot(reduced_vertices[:, 0], reduced_vertices[:, 1], 'g-o', label='Uniform Reduction')
# Set axis labels and title
ax.set_xlabel('X') ax.set_ylabel('Y')
ax.set_title('Uniform Expansion and Reduction of a Square')
# Set legend ax.legend()
# Set aspect ratio to 'equal' for a square plot
ax.set_aspect('equal') # Show the plot
plt.show()

OUTPUT:



Q.6) write a Python program to find the equation of the transformed line if shearing is applied on the line 2x + y = 3 in x and y direction by 2 and -3 units respectively.

import numpy as np

Define the original line equation original_line = np.array([2, 1, -3])

Coefficients of x, y, and constant term

Define the shear transformation matrices in x and y directions shear_matrix_x

= np.array([[1, 2, 0],

[0, 1, 0],

[0, 0, 1]])

shear_matrix_y = np.array([[1, 0, 0],

[-3, 1, 0],

[0, 0, 1]])

Apply shear transformations to the original line transformed_line_x = np.dot(shear_matrix_x, original_line) transformed_line_y = np.dot(shear_matrix_y, original_line)

Extract the coefficients of x, y, and constant term from the transformed lines a_x, b_x, c_x = transformed_line_x a_y, b_y, c_y = transformed_line_y

Print the equations of the transformed lines print("Equation of the transformed line after x-direction shear: {}x + {}y = {}".format(a_x, b_x, c_x)) print("Equation of the transformed line after y-direction shear: {}x + {}y = {}".format(a_y, b_y, c_y))

OUTPUT:

Equation of the transformed line after x-direction shear: 4x + 1y = -3

Equation of the transformed line after y-direction shear: 2x + -5y = -3

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

Max Z = 4x + 2y Subjected to
x + y <= 5 x - y >= 2 y <= 2 x > 0 , y > 0 Syntax:

from pulp import *

Create a maximization problem prob = LpProblem("Maximize Z", LpMaximize)

Define the decision variables x = LpVariable("x", lowBound=0, cat='Continuous') # x >= 0 y = LpVariable("y", lowBound=0, cat='Continuous') # y >= 0

Define the objective function

prob += 4 * x + 2 * y, "Z" # Define the constraints prob += x + y <= 5, "Constraint 1" prob += x - y >= 2, "Constraint 2" prob += y

<= 2, "Constraint 3" # Solve the

problem prob.solve()

Print the solution status print("Solution Status:

{}".format(LpStatus[prob.status])) # Print the optimal

values of the decision variables print("Optimal

Solution:") print("x = {}".format(value(x))) print("y =

{}".format(value(y)))

# Print the optimal value of the objective function
print("Z = {}".format(value(prob.objective)))

OUTPUT:

Solution Status:
Optimal Optimal
Solution: x = 5.0 y = 0.0
Z = 20.0

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 = 2x+4y subject
to
2x + 2y >= 30 x
+ 2y = 26 x>= 0, y>=
0 Syntax:
from pulp import *
# Create a maximization problem
prob = LpProblem("Minimize Z", LpMinimize)
# Define the decision variables x = LpVariable("x",
lowBound=0, cat='Continuous') # x >= 0 y = LpVariable("y",
lowBound=0, cat='Continuous') # y >= 0
# Define the objective function
obj_func = 2 * x + 4 * y prob
+= obj_func # Define the
constraints constr1 = 2 * x + 2
* y >= 30 constr2 = x + 2 * y
== 26 prob += constr1 prob
+= constr2
Solve the problem using the simplex method solver
= getSolver('PULP_CBC_CMD')
solver.actualSolve(prob) #
Print the solution status

print("Solution Status: {}".format(LpStatus[prob.status]))
# If the problem has an optimal solution, print the optimal values of the decision variables and the objective function if prob.status == LpStatusOptimal:
print("Optimal Solution:")
print("x = {}".format(value(x)))
print("y = {}".format(value(y)))
print("Z = {}".format(value(obj_func)))

OUTPUT:
Solution Status: Optimal
Optimal Solution: x =
4.0 y = 11.0 Z = 52.0

Q.9) Apply Python. Program in each of the following transformation on the point P[-2,4]
(I) Reflection through line 3x + 4y = 5 (II)
Scaling in X coordinate by factor 6.
(III) Scaling in Y coordinate by factor 4.1 (IV)
Reflection through the line y = 2x + 3 Syntax:
P = [-2, 4]
print("Original Point P: {}".format(P))
A = 3
B = 4
C = -5
# Compute the reflected point
Px_reflect = P[0] - 2 * (A * P[0] + B * P[1] + C) / (A**2 + B**2)
Py_reflect = P[1] - 2 * (A * P[1] - B * P[0] + C) / (A**2 + B**2)
P_reflect = [Px_reflect, Py_reflect] print("Reflection through line 3x + 4y = 5: {}".format(P_reflect)) # Transformation (II):
Scaling in X coordinate by factor 6 scale_factor_x = 6
Px_scaled_x = P[0] * scale_factor_x
Py_scaled_x = P[1]
P_scaled_x = [Px_scaled_x, Py_scaled_x] print("Scaling in X coordinate by factor 6: {}".format(P_scaled_x)) # Transformation (III): Scaling in Y coordinate by factor 4.1 scale_factor_y = 4.1
Px_scaled_y = P[0]
Py_scaled_y = P[1] * scale_factor_y
P_scaled_y = [Px_scaled_y, Py_scaled_y]
print("Scaling in Y coordinate by factor 4.1: {}".format(P_scaled_y))
A = 2
B = -1
C = -3
# Compute the reflected point

Px_reflect_y = P[0]
Py_reflect_y = P[1] - 2 * (A * P[0] + B * P[1] + C) / (A**2 + B**2) P_reflect_y = [Px_reflect_y, Py_reflect_y]
print("Reflection through line y = 2x + 3: {}".format(P_reflect_y))

OUTPUT:
Original Point P: [-2, 4]
Reflection through line 3x + 4y = 5: [-2.4, 2.8]
Scaling in X coordinate by factor 6: [-12, 4]
Scaling in Y coordinate by factor 4.1: [-2, 16.4]
Reflection through line y = 2x + 3: [-2, 8.4]

Q.10) Apply the following transformation on the point P[-2,4] (I) Shearing in Y direction by 7 units.
(II) Scaling in X and Y direction by 4 and 7 units respectively (III) Rotation about origin by an angle 48 degree. (IV) Reflection through the line y = x
Syntax: import math # Point P P = [-2, 4] 121
# Transformation (I): Shearing in Y direction by 7 units shear_factor_y =Px_shear_y = P[0] Py_shear_y = P[1] + shear_factor_y * P[0] P_shear_y = [Px_shear_y, Py_shear_y] print("Shearing in Y direction by 7 units: {}".format(P_shear_y)) # Transformation (II): Scaling in X and Y direction by 4 and 7 units respectively **Error! Bookmark not defined.**
scale_factor_x = 4 scale_factor_y = **Error! Bookmark not defined.**
Px_scaled_xy = P[0] * scale_factorPy_scaled_xy = P[1] * scale_factor_y **Error! Bookmark not defined.**

print("Original Point P: {}".format(P))
P_scaled_xy = [Px_scaled_xy, Py_scaled_xy] print("Scaling in X and Y direction by 4 and 7 units respectively: {}".format(P_scaled_xy))
# Transformation (III): Rotation about origin by an angle of 48 degrees
angle_degrees = 48
angle_radians = math.radians(angle_degrees)
Px_rotate = P[0] * math.cos(angle_radians) - P[1] * math.sin(angle_radians)
Py_rotate = P[0] * math.sin(angle_radians) + P[1] * math.cos(angle_radians)
P_rotate = [Px_rotate, Py_rotate]
print("Rotation about origin by an angle of 48 degrees: {}".format(P_rotate))
# Transformation (IV): Reflection through the line y = x
Px_reflect = P[1]
Py_reflect = P[0]
P_reflect = [Px_reflect, Py_reflect]
print("Reflection through the line y = x: {}".format(P_reflect))

OUTPUT:

Original Point P: [-2, 4]
Shearing in Y direction by 7 units: [-2, -10]
Scaling in X and Y direction by 4 and 7 units respectively: [-8, 28]
Rotation about origin by an angle of 48 degrees: [-4.3108405146272935, 1.1902327744806445]
Reflection through the line y = x: [4, -2]

SLIP – 22

Q.1) Write a python program to draw 2D plot y = log(x^2) + sin(x) with suitable label in the x axis , y axis and a tittle in [-5*pi,5*pi] Syntax: import numpy as np import matplotlib.pyplot as plt

Generate x values from -5*pi to 5*pi x =

np.linspace(-5 * np.pi, 5 * np.pi, 500) #

Compute y values using the given function y

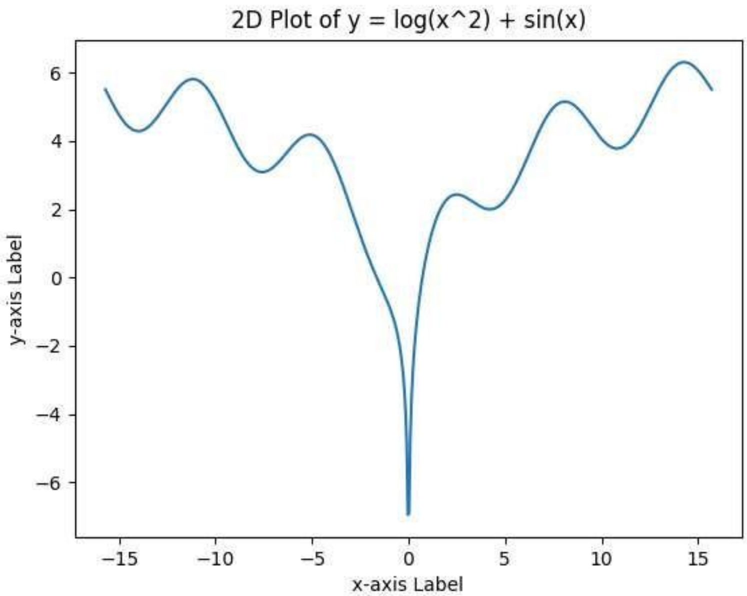
= np.log(x**2) + np.sin(x)

Create the plot plt.plot(x, y) plt.xlabel('x-axis Label')

Label for x-axis plt.ylabel('y-axis Label') # Label for y-axis

plt.title('2D Plot of y = log(x^2) + sin(x)') # Title of the plot

Show the plot plt.show()



OUTPUT:

Q.2) Write a python program to plot 3D dimensional Contour plot of parabola z = x^2 + y^2 , -6 < x,y < 6 Syntax:

import numpy as np import

matplotlib.pyplot as plt from

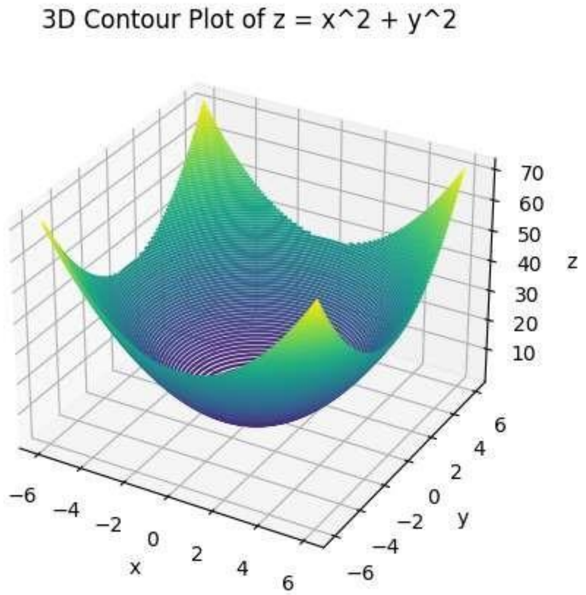
mpl_toolkits import mplot3d #

Generate x and y values x =

np.linspace(-6, 6, 100) y =

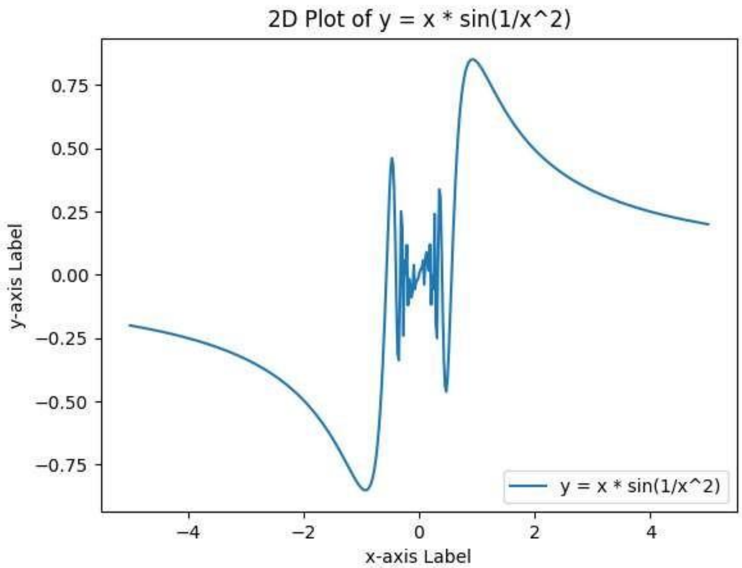
np.linspace(-6, 6, 100) # Create a
grid of x and y values
X, Y = np.meshgrid(x, y)
# Compute z values using the given function
Z = X**2 + Y**2
# Create a 3D contour plot fig =
plt.figure() ax = fig.add_subplot(111,
projection='3d') ax.contour3D(X, Y, Z,
100, cmap='viridis')
# Set labels and title ax.set_xlabel('x')
ax.set_ylabel('y') ax.set_zlabel('z')
ax.set_title('3D Contour Plot of z = x^2 + y^2')
# Show the plot plt.show()

OUTPUT:



Q.3) Write a python program to draw 2D plot y = x sin(1/x^2) in [-5,5] with suitable label in the x axis, y axis a title and location of legend to lower right corner. Syntax:

import numpy as np import
matplotlib.pyplot as plt #
Generate x values from -5 to 5 x
= np.linspace(-5, 5, 500)
Compute y values using the given function y = x *
np.sin(1 / x**2) # Create the plot plt.plot(x, y, label='y = x
* sin(1/x^2)') plt.xlabel('x-axis Label') # Label for x-axis
plt.ylabel('y-axis Label') # Label for y-axis plt.title('2D
Plot of y = x * sin(1/x^2)') # Title of the plot
plt.legend(loc='lower right') # Legend in lower-right corner
Show the plot plt.show()



OUTPUT:

Q.4) Write n python program to find the angle at each vertices of the triangle ABC where A[0,0] B[2,2] and C[0,2].

Syntax:

```
import numpy as np
# Given coordinates of points A, B, and C
A = np.array([0, 0])
B = np.array([2, 2])
C = np.array([0, 2])
# Compute the sides of the triangle
AB = np.linalg.norm(B - A)  # Length of side AB
BC = np.linalg.norm(C - B)  # Length of side BC
AC = np.linalg.norm(C - A)  # Length of side AC # Compute the
angles using the Law of Cosines angle_A = np.arccos((AB**2 +
AC**2 - BC**2) / (2 * AB * AC)) angle_B = np.arccos((-AB**2 +
AC**2 + BC**2) / (2 * AC * BC)) angle_C = np.pi - angle_A -
angle_B # Convert angles from radians to degrees angle_A_deg =
np.degrees(angle_A) angle_B_deg = np.degrees(angle_B)
angle_C_deg = np.degrees(angle_C) # Print the angles in degrees
print("Angle at vertex A: {:.2f} degrees".format(angle_A_deg))
print("Angle at vertex B: {:.2f} degrees".format(angle_B_deg))
print("Angle at vertex C: {:.2f} degrees".format(angle_C_deg))
```

Output:

Angle at vertex A: 45.00 degrees
Angle at vertex B: 90.00 degrees
Angle at vertex C: 45.00 degrees

Q.5) Write a Python program to Reflect the Point P[3,6] through the line x - 2y + 4 = 0. Syntax:

```
import numpy as np #
Given point P
P = np.array([3, 6]) # Given line parameters
a = 1  # Coefficient of x in the line equation
b = -2  # Coefficient of y in the line equation
c = 4  # Constant term in the line equation
# Compute the perpendicular distance from point P to the line dist = abs(a *
P[0] + b * P[1] + c) / np.sqrt(a**2 + b**2) # Compute the reflected point
using the formula reflected_x = P[0] - 2 * a * dist / (a**2 + b**2) reflected_y
= P[1] - 2 * b * dist / (a**2 + b**2) # Create the reflected point as a NumPy
array reflected_P = np.array([reflected_x, reflected_y]) # Print the coordinates
of the reflected point print("Reflected Point: ({:.2f},
{:.2f})".format(reflected_P[0], reflected_P[1]))
```

OUTPUT:
Reflected Point: (2.11, 7.79)

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

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

Max Z = x +
y Subjected to
x + y <= 11 x
>= 6 y >= 6

x > 0, y > 0

Syntax:

```
from pulp import * # Create the problem prob =

LpProblem("Maximization Problem", LpMaximize)

# Define the variables x =

LpVariable("x", lowBound=0) y =

LpVariable("y", lowBound=0) #

Define the objective function prob

+= x + y, "Objective Function"

# Define the constraints prob += x + y

<= 11, "Constraint 1" prob += x >= 6,

"Constraint 2" prob += y >= 6,

"Constraint 3" # Solve the problem

prob.solve() # Print the results

print("Status: ", LpStatus[prob.status])

print("Optimal Solution:") print("x = ",

value(x)) print("y = ", value(y))

print("Max Z = ",

value(prob.objective)) OUTPUT:
```

Status: Infeasible

Optimal Solution:

x = 5.0

y = 6.0

Max Z = 11.0

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-5z-4w >= -20
-8x-3y+3z+2w <= 20
-3x -2y + 4z + w <= 10 x
>= 0,y>= 0,z>= 0,w>= 0

Syntax:

```
#BY using Pulp Module  from pulp import LpProblem, LpMinimize,

LpVariable, lpSum, LpStatus, value

# Create the LPP problem
problem = LpProblem("LPP", LpMinimize)
# Define the variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) z =
LpVariable("z", lowBound=0) w =
LpVariable("w", lowBound=0) #
Define the objective function
objective = 4 * x + y + 3 * z + 5 * w
problem += objective # Define the constraints
constraint1 = 4 * x + 6 * y - 5 * z - 4 * w >= -20
constraint2 = -8 * x - 3 * y + 3 * z + 2 * w <= 20
constraint3 = -3*x -2*y + 4*z + w  <= 10
problem += constraint1 problem += constraint2
problem += constraint3
# Solve the LPP problem using the simplex method
problem.solve()
# Check if the optimization was successful if
LpStatus[problem.status] == "Optimal":
   print("Optimal solution found:")
   print("x =", value(x))     print("y =", value(y))
print("z =", value(z))     print("w =", value(w))
print("Minimum value of Z =", value(objective))
else:     print("Optimization failed.")
```

OUTPUT :
Optimal solution found:
x = 0.0 y = 0.0 z = 0.0
w = 0.0
Minimum value of Z = 0.0

Q.9) Write the python program for each of the following (I)Rotate the point (1,1) about (1,4) through angle pi / 2
(II) Find Distance between two points (0,0) and (1,0)
(III) Find the shearing of the point (3,4) in X direction by 3 units.
(IV) Represent two dimensional points using point function (-2,5)
Syntax: import math
# Rotate point (x, y) about (cx, cy) through angle theta (in radians) def rotate_point(x, y, cx, cy, theta):
dx = x - cx dy = y - cy cos_theta = math.cos(theta) sin_theta = math.sin(theta)
new_x = cx + dx * cos_theta - dy * sin_theta
new_y = cy + dx * sin_theta + dy * cos_theta
return new_x, new_y
Find distance between two points (x1, y1) and (x2, y2)
def distance_between_points(x1, y1, x2, y2): return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2) # Shear point (x, y) in X direction by shx units
def shear_point_x(x, y, shx):
new_x = x + shx * y
return new_x, y
# Define a class for representing two-dimensional points
class Point: def __init__(self, x, y):
self.x = x
self.y = y
# Rotate point (1,1) about (1,4) through angle pi/2 x1, y1 = rotate_point(1, 1, 1, 4, math.pi / 2) print("Rotated point: ({}, {})".format(x1, y1)) # Find distance between two points (0,0) and (1,0) x2, y2 = 0, 0 x3, y3 = 1, 0 distance = distance_between_points(x2, y2, x3, y3) print("Distance between points: {}".format(distance)) # Find the shearing of the point (3,4) in X direction by 3 units x4, y4 = shear_point_x(3, 4, 3) print("Sheared point: ({}, {})".format(x4, y4)) # Represent two-dimensional points using Point class p = Point(-2, 5)
print("Point: ({}, {})".format(p.x, p.y))

OUTPUT:
Rotated point: (4.0, 4.0)
Distance between points: 1.0
Sheared point: (15, 4)
Point: (-2, 5)

Q.10) A company has 3 production facilities S1, S2, and S3 with production capacity of 7,9 and 18 units (in 100's) per week of a product, respectively. These units are to be shipped to 4 warehouse D1,D2,D3,D4 with requirement of 5,6,7 and 14 units (in 100's) per week, respectively. The transportation costs (in rupee) per units between factories to warehouse are given in the table below.

	D1	D2	D3	D4	Supply
S1	19	30	50	10	7
S2	70	30	40	60	9
S3	40	8	70	20	18
Demand	5	8	7	14	34

Write a python program to solve transportation problem for minimizing the costs of whole operation.

Syntax: from pulp
import *

```
# Define the factories, warehouses, and their respective capacities
factories = ['S1', 'S2', 'S3'] warehouses = ['D1', 'D2', 'D3', 'D4']
capacities = {'S1': 7, 'S2': 9, 'S3': 18}
requirements = {'D1': 5, 'D2': 6, 'D3': 7, 'D4': 14}
# Define the transportation costs between factories and warehouses
transportation_costs = {
   ('S1', 'D1'): 19, ('S1', 'D2'): 30, ('S1', 'D3'): 50, ('S1', 'D4'): 10,
   ('S2', 'D1'): 70, ('S2', 'D2'): 30, ('S2', 'D3'): 40, ('S2', 'D4'): 60,
('S3', 'D1'): 40, ('S3', 'D2'): 8, ('S3', 'D3'): 70, ('S3', 'D4'): 20 }
# Crete a binary variable to represent the shipment decision
shipment = LpVariable.dicts('Shipment', (factories, warehouses), lowBound=0,
cat='Integer')
# Create the LP problem
prob = LpProblem('Transportation Problem', LpMinimize)
# Define the objective function
prob += lpSum(shipment[f][w] * transportation_costs[f, w] for f in factories for
w in warehouses)
# Add the capacity constraints for factories for
f in factories:
   prob += lpSum(shipment[f][w] for w in warehouses) <= capacities[f]
# Add the demand constraints for warehouses for w in warehouses:
prob += lpSum(shipment[f][w] for f in factories) >= requirements[w]
# Solve the LP problem prob.solve()
# Print the optimal solution
print('Optimal Solution:')
for f in factories:     for w
in warehouses:
      if shipment[f][w].varValue > 0:
         print('Ship {} units from {} to {}'.format(shipment[f][w].varValue, f, w))
```

```
# Print the total cost of the optimal solution
print('Total Cost: Rs. {}'.format(value(prob.objective)))
```

OUTPUT:

```
Optimal Solution:
Ship 5.0 units from S1 to D1
Ship 2.0 units from S1 to D4
Ship 7.0 units from S2 to D3
Ship 6.0 units from S3 to D2
Ship 12.0 units from S3 to D4
Total Cost: Rs. 683.0
```

SLIP-23

Q.1) Write a python program to plot the graph of sinx, and cos x in [0,pi] in one figure with 2 *1 subplots.

Syntax: import numpy as np

import matplotlib.pyplot as plt

Generate x values from 0 to pi with 100 data points

x = np.linspace(0, np.pi, 100) # Calculate sin(x) and

cos(x) values sin_x = np.sin(x) cos_x = np.cos(x)

Create a figure with 2x1 subplots fig,

axs = plt.subplots(2, 1, figsize=(8, 6))

Plot sin(x) in the first subplot

axs[0].plot(x, sin_x, label='sin(x)')

axs[0].set_title('sin(x)') axs[0].legend()

Plot cos(x) in the second subplot

axs[1].plot(x, cos_x, label='cos(x)')

axs[1].set_title('cos(x)') axs[1].legend()

Add overall title to the figure

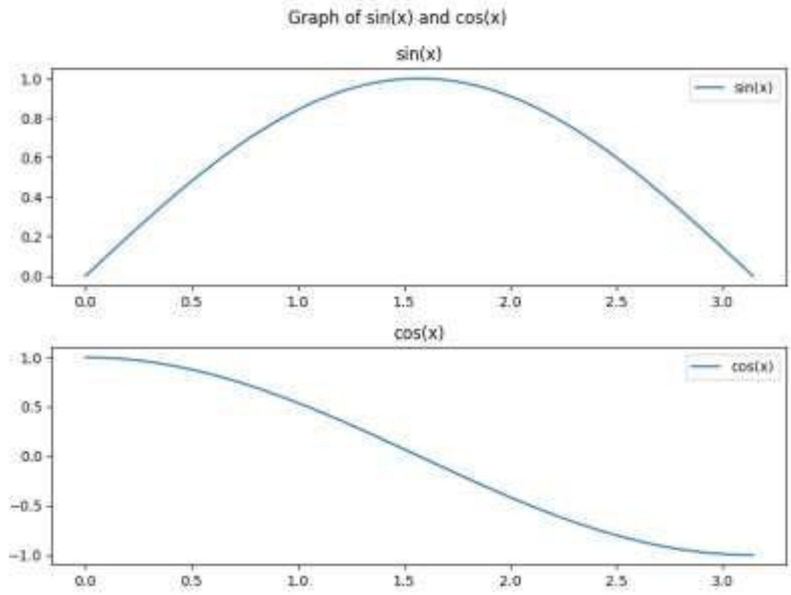
fig.suptitle('Graph of sin(x) and cos(x)') #

Adjust spacing between subplots

plt.tight_layout()

Show the plot plt.show()

OUTPUT:



Q.2) Write a python program to plot 30 Surface Plot of the function z = cos(|x| +|y|) in -1 < x,y1 < 1.

Syntax:

import numpy as np import

matplotlib.pyplot as plt

Generate 30 random x, y pairs within the range -1 < x, y < 1

np.random.seed(0) x_vals = np.random.uniform(-1, 1,

size=30) y_vals = np.random.uniform(-1, 1, size=30)

Create a 2D grid of x, y values x, y = np.meshgrid(np.linspace(-

1, 1, 100), np.linspace(-1, 1, 100))

Compute the z values for each x, y pair z

= np.cos(np.abs(x) + np.abs(y))

Plot the surface plots for i in range(30):

fig = plt.figure()   ax =

fig.add_subplot(111, projection='3d')

ax.plot_surface(x, y, z, cmap='viridis')

ax.set_xlabel('X')   ax.set_ylabel('Y')

ax.set_zlabel('Z')   ax.set_title(f'Surface

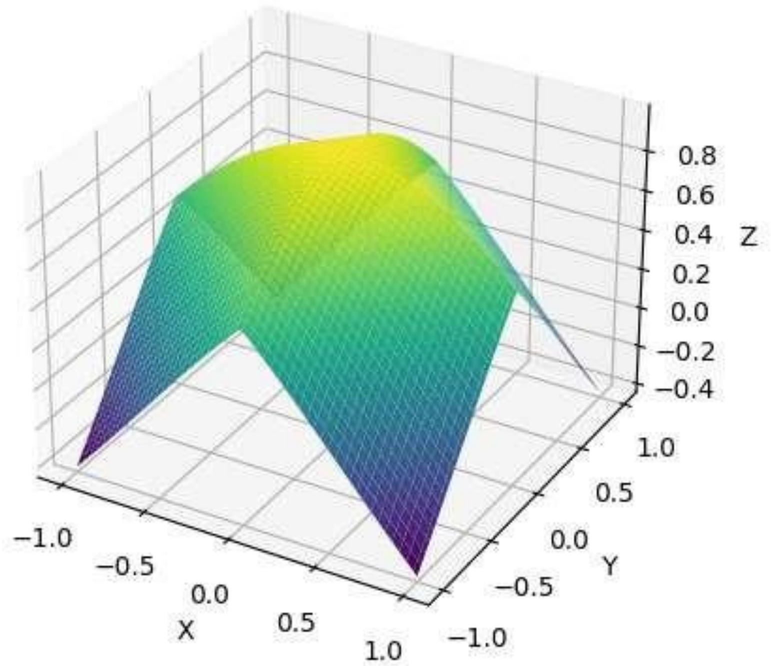
Plot {i+1}: z = cos(|x| + |y|) for x =

{x_vals[i]:.2f}, y =

{y_vals[i]:.2f}')

plt.show()

OUTPUT: Surface Plot 1: z = cos(|x| + |y|) for x = 0.10, y = -0.47



Q.3) Write a python program to Plot the graph of the following function in the given interval

i) f(x) = x^3 in [0,5] ii)

f(x) x ^2 in [-2,2]

Syntax:

import numpy as np import

matplotlib.pyplot as plt

Define the functions

def f1(x):

return x**3 def

f2(x): return

x**2

Generate x values for the intervals

x1 = np.linspace(0, 5, 100) x2 =

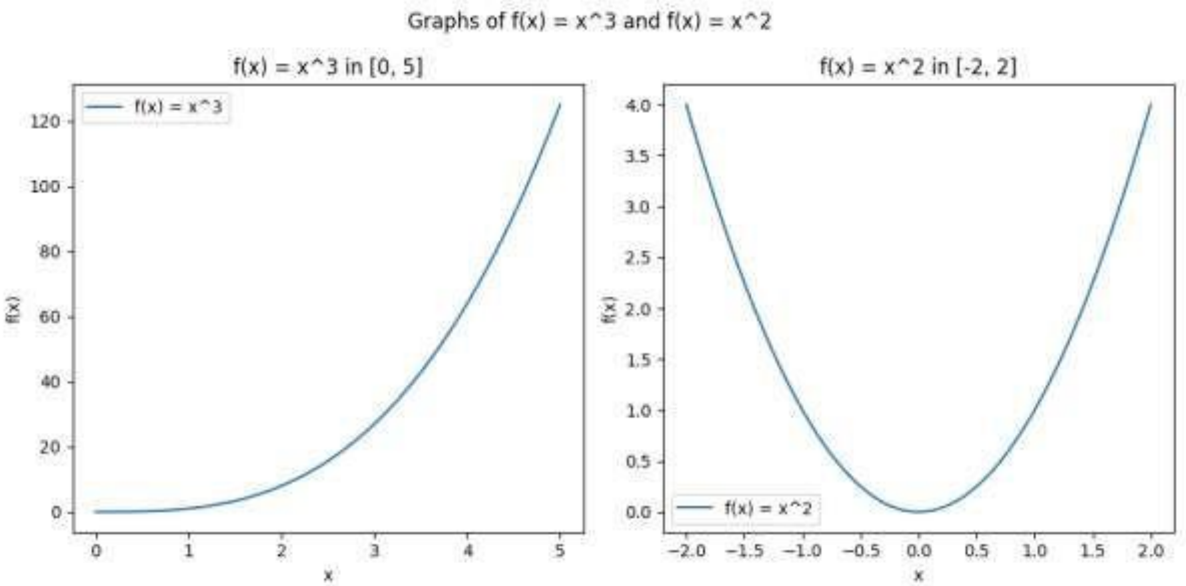
np.linspace(-2, 2, 100) # Calculate y

values for the functions y1 = f1(x1)

y2 = f2(x2)

```
# Create a figure with two subplots side by side
fig, axs = plt.subplots(1, 2, figsize=(10, 5)) #
Plot f(x) = x^3 in the first subplot
axs[0].plot(x1, y1, label='f(x) = x^3')
axs[0].set_xlabel('x') axs[0].set_ylabel('f(x)')
axs[0].set_title('f(x) = x^3 in [0, 5]')
axs[0].legend()
# Plot f(x) = x^2 in the second subplot
axs[1].plot(x2, y2, label='f(x) = x^2')
axs[1].set_xlabel('x')
axs[1].set_ylabel('f(x)') axs[1].set_title('f(x)
= x^2 in [-2, 2]') axs[1].legend()
# Add overall title to the figure
fig.suptitle('Graphs of f(x) = x^3 and f(x) = x^2')
# Adjust spacing between subplots
plt.tight_layout() # Show the plot
plt.show()
```

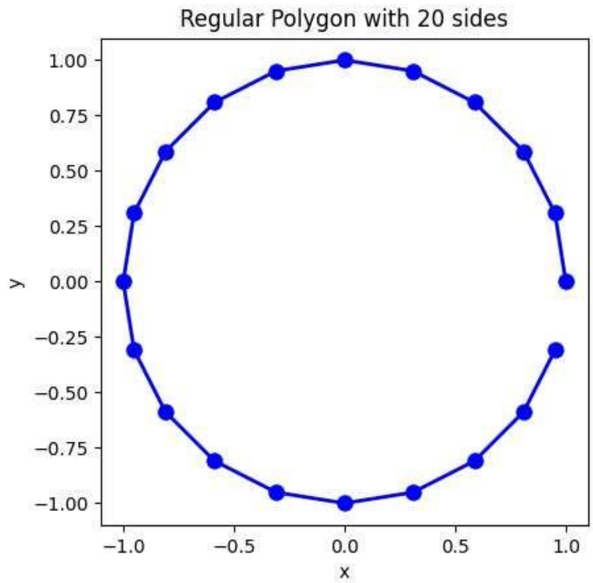
OUTPUT:



Q.4) Write a python program to draw regular polygon with 20 sides and radius 1 centered at (0,0) Syntax:

```
import numpy as np import
matplotlib.pyplot as plt #
```

Number of sides of the polygon n
= 20
Radius of the polygon radius
= 1
Generate angles for the vertices of the polygon angles
= np.linspace(0, 2 * np.pi, n + 1)[:-1]
# Calculate x and y coordinates for the vertices of the polygon
x = radius * np.cos(angles) y = radius * np.sin(angles)
# Create a figure fig, ax =
plt.subplots() # Plot the
regular polygon
ax.plot(x, y, 'b-o', linewidth=2, markersize=8) ax.set_aspect('equal',
'box')
ax.set_title(f'Regular Polygon with {n} sides')
ax.set_xlabel('x')
ax.set_ylabel('y')
#
Show
the
plot



plt.show() Output:

Q.5) Write a Python program to draw a polygon with vertices

(0,0),(1,0),(2,2),(1,4). Also find area of polygon.

Syntax: import matplotlib.pyplot as plt #

Define the vertices of the polygon vertices

= [(0, 0), (1, 0), (2, 2), (1, 4)] # Extract x

and y coordinates of the vertices x =

[vertex[0] for vertex in vertices] y =

[vertex[1] for vertex in vertices]

Create a figure fig, ax = plt.subplots() # Plot the polygon ax.plot(x + [x[0]], y + [y[0]], 'b-o', linewidth=2, markersize=8) # Connect last vertex to first vertex ax.set_aspect('equal', 'box') ax.set_title('Polygon') ax.set_xlabel('x')

ax.set_ylabel('y')

Calculate the area of the polygon using Shoelace formula

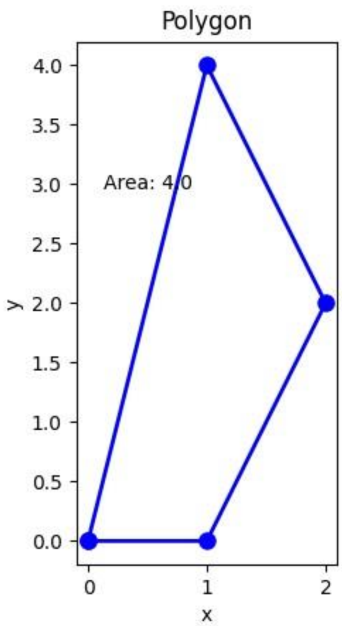
area = 0 for i in range(len(vertices)):

area += x[i] * y[(i + 1) % len(vertices)] - y[i] * x[(i + 1) % len(vertices)] area *= 0.5

Display the area of the polygon ax.text(0.5, 3,

f'Area: {area}', ha='center', va='center')

Show the plot



plt.show() Output:

Q.6) Write a Python program to find area and perimeter of triangle ABC where A[0, 1], B[-5,0] and C[-3,3].

Synatx:

```
import math

# Define the coordinates of the vertices A, B, and C

A = [0, 1]

B = [-5, 0]

C = [-3, 3]
# Calculate the lengths of the sides AB, BC, and AC using Euclidean distance formula

AB = math.sqrt((B[0] - A[0])**2 + (B[1] - A[1])**2)

BC = math.sqrt((C[0] - B[0])**2 + (C[1] - B[1])**2)

AC = math.sqrt((C[0] - A[0])**2 + (C[1] - A[1])**2)

# Calculate the perimeter of the triangle perimeter

= AB + BC + AC

# Calculate the area of the triangle using Heron's formula

s = perimeter / 2  # Semi-perimeter of the triangle area =

math.sqrt(s * (s - AB) * (s - BC) * (s - AC))

# Print the calculated area and perimeter print("Area

of triangle ABC:", area) print("Perimeter of triangle

ABC:", perimeter)
```

OUTPUT:

Area of triangle ABC: 6.500000000000002

Perimeter of triangle ABC: 12.310122064520764

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

Max Z = 3x + 5y + 4z
Subjected to
2x + 3y <= 8
2x + 5y <= 10
3x+2y+4z <= 15
x , y,z > 0
Syntax:

```
import numpy as np
from scipy.optimize import linprog
# Define the coefficients of the objective function
c = [-3, -5, -4]  # Coefficients of x, y, z in the objective function
# Define the coefficients of the inequality constraints


A = [
   [2, 3, 0],  # Coefficients of x, y, z in the first inequality constraint
   [2, 5, 0],  # Coefficients of x, y, z in the second inequality constraint
[3, 2, 4]   # Coefficients of x, y, z in the third inequality constraint
]
b = [8, 10, 15]  # Right-hand side values of the inequality constraints
# Define the bounds for the variables
x_bounds = (0, None)  # x >= 0
y_bounds = (0, None)  # y >= 0
z_bounds = (0, None)  # z >= 0 # Solve
the linear programming problem
res = linprog(c, A_ub=A, b_ub=b, bounds=[x_bounds, y_bounds,
z_bounds], method='simplex') # Extract the results x = res.x[0]  # Value of x
that maximizes the objective function y = res.x[1]  # Value of y that
maximizes the objective function z = res.x[2]  # Value of z that
maximizes the objective function max_z = -res.fun  # Maximum value
of the objective function (negation
due to maximization) # Print the
results print("Optimal
solution:")
print("x =", x)
print("y =", y)
print("z =", z)
print("Max Z =", max_z)
```

OUTPUT:

Optimal solution:
x = 0.0 y = 2.0 z
= 2.75
Max Z = 21.0

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 = 3x+5y + 4z
subject to 2x+ 2y
<= 12
2x+ 2y <= 10
5x + 2y <= 10
x>=0,y>=0,z>=0

Syntax: from pulp
import *
Create a minimization problem prob = LpProblem("LPP",
LpMinimize) # Define the decision variables x =
LpVariable('x', lowBound=0, cat='Continuous') # x >= 0 y =
LpVariable('y', lowBound=0, cat='Continuous') # y >= 0 z =
LpVariable('z', lowBound=0, cat='Continuous') # z >= 0
Define the objective function
prob += 3*x + 5*y + 4*z # Define
the inequality constraints prob +=
2*x + 2*y <= 12 prob += 2*x +
2*y <= 10 prob += 5*x + 2*y <=
10
# Solve the linear programming problem
prob.solve(PULP_CBC_CMD(msg=False))
# Extract the results optimal_solution = [] if
LpStatus[prob.status] == 'Optimal':
optimal_solution.append(('x', value(x)))
optimal_solution.append(('y', value(y)))
optimal_solution.append(('z', value(z)))
optimal_solution.append(('Min Z', value(prob.objective)))
else:
   print("No optimal solution found.")
# Print the results print("Optimal
solution:") for variable, value in
optimal_solution:
   print(variable, "=", value)

OUTPUT:
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 = (3, -1)

(I) Reflection through X axis
(II) Rotation about origin through an angle 30 degree
(III) Scaling in Y Coordinate by factor 8 (IV) Shearing in X Direction by 2 units Syntax:

import math # Initial point point =
(3, -1) print("Initial point:", point)
# Reflection through X axis
reflection_x = (point[0], -point[1])
print("Reflection through X axis:", reflection_x)
# Rotation about origin by 30 degrees angle
= math.radians(30)
rotation = (point[0] * math.cos(angle) - point[1] * math.sin(angle), point[0] * math.sin(angle) + point[1] * math.cos(angle)) print("Rotation about origin by 30 degrees:", rotation)
# Scaling in Y coordinate by factor 8 scaling_y
= (point[0], point[1] * 8)
print("Scaling in Y coordinate by factor 8:", scaling_y)
# Shearing in X direction by 2 units shearing_x
= (point[0] + 2 * point[1], point[1])
print("Shearing in X direction by 2 units:", shearing_x)
OUTPUT:
Initial point: (3, -1)
Reflection through X axis: (3, 1)
Rotation about origin by 30 degrees: (2.0497560061708553, 1.6960434741550814)
Scaling in Y coordinate by factor 8: (3, -8)
Shearing in X direction by 2 units: (1, -1)

Q.10) Write a python program lo apply the following transformation on the point = (-2, 4)

(I) Reflection through y = x + 2
(II) Scaling in Y Coordinate by factor 2
(III) Shearing in X direction by 4units
(IV) Rotation about origin through an angle 60 degree

Syntax:

```
import math # Initial point point = (-
2, 4) print("Initial point:", point) #
Reflection through y = x + 2
reflection = (point[1] - 2, point[0] - 2)
print("Reflection through y = x + 2:", reflection)
# Scaling in Y coordinate by factor 2 scaling_y
= (point[0], point[1] * 2)
print("Scaling in Y coordinate by factor 2:", scaling_y)
# Shearing in X direction by 4 units shearing_x =
(point[0] + 4 * point[1], point[1]) print("Shearing in X
direction by 4 units:", shearing_x)
# Rotation about origin by 60 degrees angle
= math.radians(60)
rotation = (point[0] * math.cos(angle) - point[1] * math.sin(angle), point[0] *
math.sin(angle) + point[1] * math.cos(angle)) print("Rotation about origin by 60
degrees:", rotation)
```

OUTPUT:

```
Initial point: (-2, 4)
Reflection through y = x + 2: (2, -4)
Scaling in Y coordinate by factor 2: (-2, 8)
Shearing in X direction by 4 units: (14, 4)
Rotation about origin by 60 degrees: (-1.9641016151377544,
2.098076211353316)
```

SLIP-24

Q.1) Write the python program to plot 3D graph of the function f(x) = e(-x^2) in [-5,5] with green dashed points line with upward pointing triangle.

Syntax: import numpy as np

import matplotlib.pyplot as plt

Define the function def

f(x):

    return np.exp(-x**2)

Generate x values in the range [-5, 5] x

= np.linspace(-5, 5, 100)

Calculate y values using the function y

= f(x)

Create a 3D plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

Plot the points with green dashed lines and upward pointing triangles as

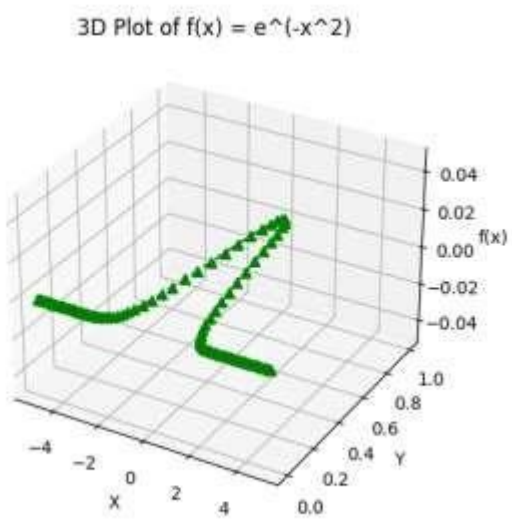
markers ax.plot(x, y, 'g--', marker='^', markersize=6)

Set labels and title ax.set_xlabel('X')

ax.set_ylabel('Y') ax.set_zlabel('f(x)')

ax.set_title('3D Plot of f(x) = e^(-x^2)')

Show the plot plt.show()



OUTPUT:

Q.2) Write the python program to plot graph of the function f(x) = log(3x^2) in [1,10] with black dashed points Syntax:

import numpy as np import

matplotlib.pyplot as plt #

Define the function def f(x):

   return np.log(3 * x**2)

Generate x values in the range [1, 10] x

= np.linspace(1, 10, 100)

Calculate y values using the function y =

f(x) # Create a plot plt.plot(x, y, 'k--',

marker='o', markersize=6)

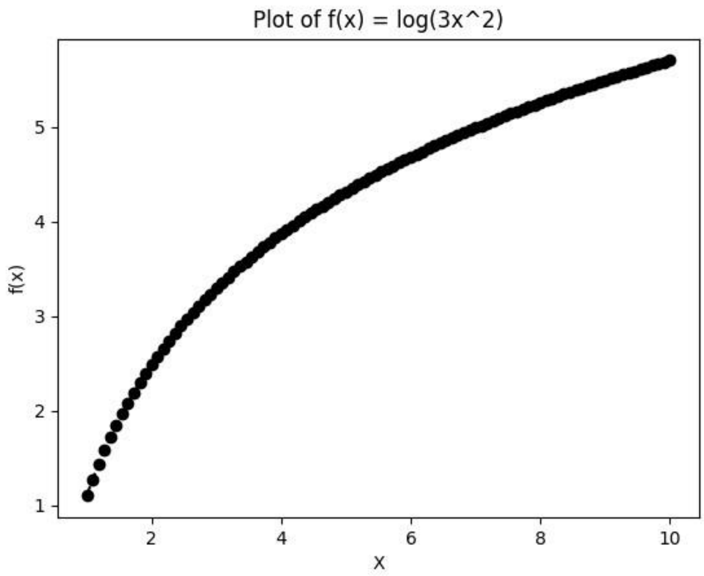
Set labels and title

plt.xlabel('X') plt.ylabel('f(x)')

plt.title('Plot of f(x) = log(3x^2)')

Show the plot plt.show()

OUTPUT:



Q.3) Write the python program to plot the graph of the function using def ()

$$f(x) = \{ \begin{array}{l} x^2 + 4, if - 10 < x < 5 \\ 3x + 9, if \quad 5 < x \geq 0 \end{array}$$

Syntax:

import numpy as np import

matplotlib.pyplot as plt def

f(x):

"""Function to define f(x)."""

if -10 < x < 5:

return x**2 + 4

elif 5 <= x:

return 3*x + 9

else:

return None # Generate x values x = np.linspace(-11, 11, 500) #

Generate 500 points between -11 and 11

Calculate y values using f(x) y

= np.array([f(xi) for xi in x]) #

Create the plot plt.plot(x, y,

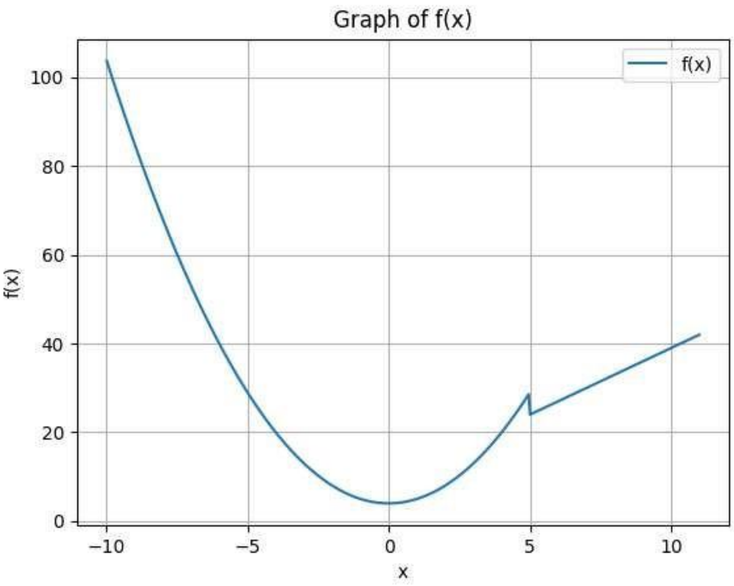
label='f(x)') plt.xlabel('x')

plt.ylabel('f(x)') plt.title('Graph

of f(x)') plt.legend()

plt.grid(True) plt.show()

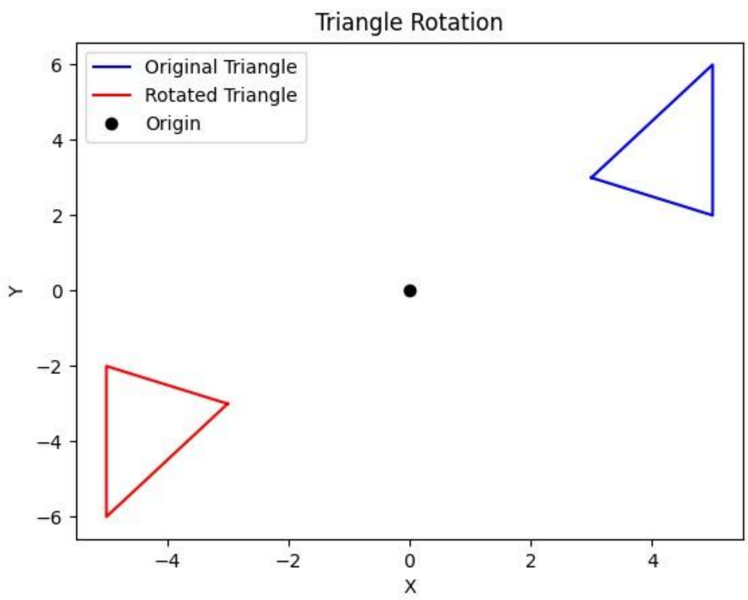
OUTPUT:



Q.4) Write the python program to plot triangle with vertices [3,3],[5,6],[5,2] and its rotation about the origin by angle –pi radians Syntax:

```
import numpy as np
import matplotlib.pyplot as plt
# Define the vertices of the original triangle
v1 = np.array([3, 3]) v2 = np.array([5, 6])
v3 = np.array([5, 2])
# Calculate the rotation matrix theta = -
np.pi  # Angle of rotation in radians R =
np.array([[np.cos(theta), -np.sin(theta)],
[np.sin(theta), np.cos(theta)]])
# Apply the rotation matrix to each vertex
v1_rotated = np.dot(R, v1) v2_rotated =
np.dot(R, v2) v3_rotated = np.dot(R, v3)
# Create a plot plt.figure()
plt.plot([v1[0], v2[0], v3[0], v1[0]], [v1[1], v2[1], v3[1], v1[1]], 'b-',
label='Original Triangle')
plt.plot([v1_rotated[0], v2_rotated[0], v3_rotated[0], v1_rotated[0]],
[v1_rotated[1], v2_rotated[1], v3_rotated[1], v1_rotated[1]], 'r-', label='Rotated
Triangle')
plt.plot(0, 0, 'ko', label='Origin')
plt.xlabel('X')
plt.ylabel('Y')
```

plt.title('Triangle Rotation')
plt.legend() # Show the
plot plt.show() Output:



Q.5) Write a python to generate vector x in the interval [-22,22] using numpy package with 80 subinterval

Syntax:

import numpy as np
# Generate vector x with 80 subintervals
n_subintervals = 80 lower_bound = -22
upper_bound = 22
x = np.linspace(lower_bound, upper_bound, n_subintervals+1)
# Print the generated vector x
print("Vector x:", x)
OUTPUT:

```
Vector x: [-22.   -21.45 -20.9  -20.35 -19.8  -19.25 -18.7  -18.15 -17.6  -17.05
 -16.5  -15.95 -15.4  -14.85 -14.3  -13.75 -13.2  -12.65 -12.1  -11.55
 -11.   -10.45  -9.9   -9.35  -8.8   -8.25  -7.7   -7.15  -6.6   -6.05
  -5.5   -4.95  -4.4   -3.85  -3.3   -2.75  -2.2   -1.65  -1.1   -0.55
   0.     0.55   1.1    1.65   2.2    2.75   3.3    3.85   4.4    4.95
   5.5    6.05   6.6    7.15   7.7    8.25   8.8    9.35   9.9   10.45
  11.    11.55  12.1   12.65  13.2   13.75  14.3   14.85  15.4   15.95
  16.5   17.05  17.6   18.15  18.7   19.25  19.8   20.35  20.9   21.45
  22.  ]
```

Q.6) Write a Python program to draw a polygon with vertices (0,0) ,(1,0) , (2,2) ,(1,4) also find area and perimeter of the polygon.

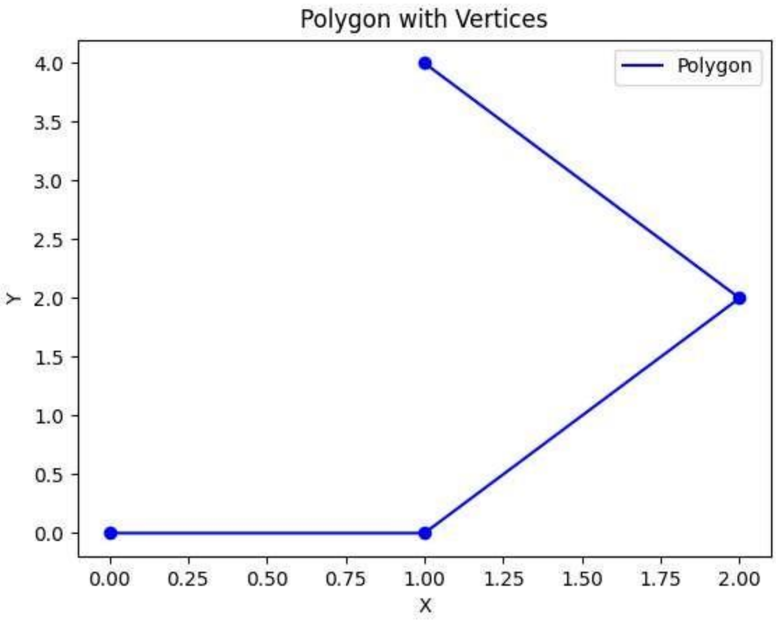
Syntax:

```
import numpy as np import matplotlib.pyplot
as plt # Define the vertices of the polygon
vertices = np.array([[0, 0], [1, 0], [2, 2], [1,
4]]) # Extract x and y coordinates of the
vertices x = vertices[:, 0] y = vertices[:, 1] #
Plot the polygon plt.plot(x, y, 'b-',
label='Polygon') plt.plot(x, y, 'bo')
plt.xlabel('X') plt.ylabel('Y')
plt.title('Polygon with Vertices')
plt.legend()
# Calculate the area of the polygon using shoelace formula def
calculate_area(vertices):
   x = vertices[:, 0]    y = vertices[:, 1]    return 0.5 * np.abs(np.dot(x,
np.roll(y, 1)) - np.dot(y, np.roll(x, 1))) area = calculate_area(vertices) #
Calculate the perimeter of the polygon perimeter =
np.sum(np.sqrt(np.sum(np.diff(vertices, axis=0)**2, axis=1)))
# Print the calculated area and perimeter
print("Area of the polygon:", area) print("Perimeter
of the polygon:", perimeter)
# Show the plot plt.show()
```

OUTPUT:

Area of the polygon: 4.0

Perimeter of the polygon: 5.47213595499958



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

Max Z = 3.5x +

2y Subjected to x

+ y >= 5 x >=4

y<=5 x >= 0,y>=

0.

Syntax:

from pulp import * # Create the LP problem

problem = LpProblem("Maximize Z",

LpMaximize)

Define the decision variables x =

LpVariable('x', lowBound=0) # x >= 0 y =

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

Define the objective function problem +=

3.5 * x + 2 * y # Define the constraints

problem += x + y >= 5 problem += x >= 4

problem += y <= 5 # Solve the LP

problem status = problem.solve() # Check

the solution status if status == 1:

```
# Print the optimal solution
print("Optimal solution:")
print(f"x = {value(x)}")    print(f"y
= {value(y)}")    print(f"Z =
{value(problem.objective)}") else:
    print("No feasible solution found.")
OUTPUT:
No feasible solution found.
```

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: Optimal

Status: Infeasible

Q.9) Write a python program lo apply the following transformation on the point = (3, -1)
(I) Reflection through X axis
(II) Reflection through the line y = x.
(III) Scaling in X Coordinate by factor 2
(IV) Scaling in Y Coordinate by factor 1.5
Sy import numpy as np

Define the point point

= np.array([3, -1])

Transformation 1: Reflection through X axis reflection_x =

np.array([[1, 0], [0, -1]]) point_reflection_x = np.dot(reflection_x,

point) print("After reflection through X axis:", point_reflection_x)

Transformation 2: Reflection through the line y = x reflection_yx

= np.array([[0, 1], [1, 0]]) point_reflection_yx =

np.dot(reflection_yx, point) print("After reflection through the line

y = x:", point_reflection_yx) # Transformation 3: Scaling in X

Coordinate by factor 2 scaling_x = np.array([[2, 0], [0, 1]])

point_scaling_x = np.dot(scaling_x, point) print("After scaling in X

Coordinate by factor 2:", point_scaling_x) # Transformation 4:

Scaling in Y Coordinate by factor 1.5 scaling_y = np.array([[1, 0],

[0, 1.5]]) point_scaling_y = np.dot(scaling_y, point) print("After

scaling in Y Coordinate by factor 1.5:", point_scaling_y)ntax:

OUTPUT:
After reflection through X axis: [3 1]
After reflection through the line y = x: [-1 3]
After scaling in X Coordinate by factor 2: [6 -1]
After scaling in Y Coordinate by factor 1.5: [3. -1.5]

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

(I) Reflection Through the line y = x
(II) Scaling in X-Coordinate by factor 3
(III) Shearing in Y – Direction by 4.5 unit (IV) Rotation about origin by an angle pi.

Syntax:

```
import numpy as np #
Define the points A and B
A = np.array([4, -1])
B = np.array([3, 0])
# Transformation 1: Reflection through the line y = x
reflection_yx = np.array([[0, 1], [1, 0]]) A_reflection_yx
= np.dot(reflection_yx, A)
B_reflection_yx = np.dot(reflection_yx, B)
# Transformation 2: Scaling in X-Coordinate by factor 3
scaling_x = np.array([[3, 0], [0, 1]])
A_scaling_x = np.dot(scaling_x, A_reflection_yx)
B_scaling_x = np.dot(scaling_x, B_reflection_yx)
# Transformation 3: Shearing in Y-Direction by 4.5 units
shearing_y = np.array([[1, 0], [0, 1]]) shearing_y[0,
1] = 4.5
A_shearing_y = np.dot(shearing_y, A_scaling_x)
B_shearing_y = np.dot(shearing_y, B_scaling_x) #
Transformation 4: Rotation about origin by an angle pi
rotation_pi = np.array([[-1, 0], [0, -1]]) A_rotation_pi =
np.dot(rotation_pi, A_shearing_y)
B_rotation_pi = np.dot(rotation_pi, B_shearing_y)
# Print the transformed points print("After
Reflection through the line y = x:") print("A:",
A_reflection_yx) print("B:", B_reflection_yx)
print("\nAfter Scaling in X-Coordinate by factor 3:")
print("A:", A_scaling_x) print("B:", B_scaling_x)
print("\nAfter Shearing in Y-Direction by 4.5 units:")
print("A:", A_shearing_y) print("B:", B_shearing_y)
```

```
print("\nAfter Rotation about origin by an angle pi:")
print("A:", A_rotation_pi) print("B:", B_rotation_pi)
```

OUTPUT:

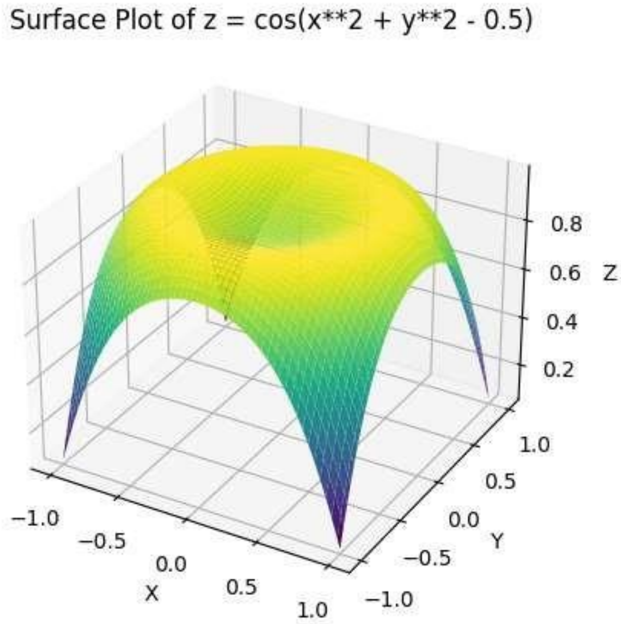
```
After Reflection through the line y = x:
A: [-1  4]
B: [0 3]
After Scaling in X-Coordinate by factor 3:
A: [-3  4]
B: [0 3]
After Shearing in Y-Direction by 4.5 units:
A: [13  4]
B: [12  3]
After Rotation about origin by an angle pi:
A: [-13  -4]
B: [-12  -3]
```

SLIP-25

Q.1) Using Python plot the surface plot of function z = cos (x**2 + y**2 - 0.5) in the interval from -1 < x,y < 1.

Syntax:
```
import numpy as np import
matplotlib.pyplot as plt from
mpl_toolkits.mplot3d import Axes3D
# Define the function def
func(x, y):
   return np.cos(x**2 + y**2 - 0.5)
# Generate x, y values in the interval from -1 to 1
x = np.linspace(-1, 1, 100) y = np.linspace(-1, 1,
100)
X, Y = np.meshgrid(x, y)  # Create a grid of x, y values
Z = func(X, Y)  # Compute z values using the function
# Create a 3D plot fig = plt.figure() ax =
fig.add_subplot(111, projection='3d') ax.plot_surface(X,
Y, Z, cmap='viridis')  # Plot the surface ax.set_xlabel('X')
ax.set_ylabel('Y') ax.set_zlabel('Z')
ax.set_title('Surface Plot of z = cos(x**2 + y**2 - 0.5)') plt.show()
# Show the plot
```

OUTPUT:



Q.2) Write n Python program to generate 3D plot of the function z = six (x) + cos(y) in -10 < x, y < 10.

Syntax:

import numpy as np import

matplotlib.pyplot as plt from

mpl_toolkits.mplot3d import Axes3D

Generate x, y values x =

np.linspace(-10, 10, 100) y =

np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)

Calculate z values

Z = np.sin(X) + np.cos(Y) # Create 3D

plot fig = plt.figure() ax =

fig.add_subplot(111, projection='3d')

Plot the surface ax.plot_surface(X, Y,

Z, cmap='viridis')

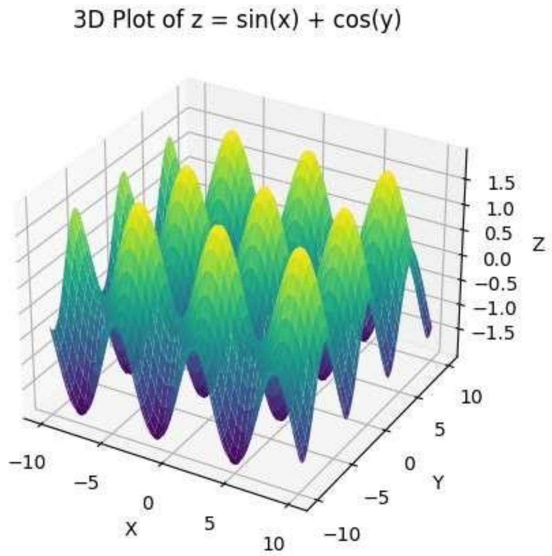
Set labels and title

ax.set_xlabel('X') ax.set_ylabel('Y')

ax.set_zlabel('Z') ax.set_title('3D Plot of z

= sin(x) + cos(y)')

Show the plot plt.show()



Q.3) Using Python plot the graph of function f(x) = sin-1(x) on the interval [-1,
OUTPUT:

1].

Syntax:

import numpy as np import

matplotlib.pyplot as plt #

Generate x values x =

np.linspace(-1, 1, 1000) #

Calculate f(x) values f_x =

np.arcsin(x) # Create the plot

plt.plot(x, f_x) plt.xlabel('x')

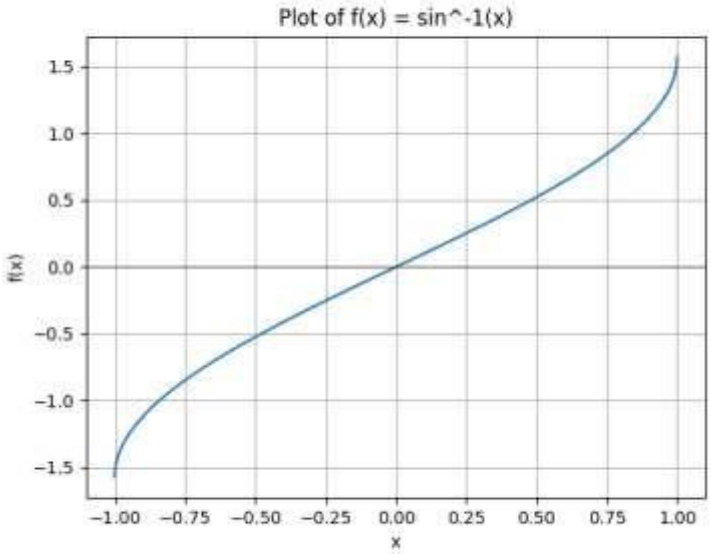
plt.ylabel('f(x)')

plt.title('Plot of f(x) = sin^-1(x)') plt.grid(True) plt.axhline(0,

color='black', lw=0.5) # Add horizontal grid line at y=0

Show the plot plt.show()

OUTPUT:



Q.4) Rotate the line segment by 180° having endpoints (1,0) and (2,-1).

Syntax:

```
import numpy as np
# Define the endpoints of the line segment
point1 = np.array([1, 0]) point2 =
np.array([2, -1])
# Define the rotation matrix for 180 degrees
R = np.array([[-1, 0],[0, -1]])
# Rotate the endpoints of the line segment
rotated_point1 = np.dot(R, point1)
rotated_point2 = np.dot(R, point2) # Print
the rotated coordinates print("Rotated
endpoint 1: ", rotated_point1) print("Rotated
endpoint 2: ", rotated_point2)
```

Output:

```
Rotated endpoint 1:  [-1  0]
Rotated endpoint 2:  [-2  1]
```

Q.5) Using sympy, declare the points P(5, 2), Q(5, -2), R(5, 0), check whether these points are collinear. Declare the ray passing through the points P and Q, find the length of this ray between P and Q. Also find slope of this ray. Syntax:

```
from sympy import Point, Line
# Declare the points
P  = Point(5, 2)
Q = Point(5, -2)
R = Point(5, 0)
```

Check if points are collinear collinear
= Point.is_collinear(P, Q, R) if
collinear:
    print("Points P, Q, and R are collinear.") else:
print("Points P, Q, and R are not collinear.") #
Declare the ray passing through points P and Q
ray_PQ = Line(P, Q)
# Calculate the length of the ray PQ length_PQ
= P.distance(Q)
print("Length of ray PQ:", length_PQ)
# Calculate the slope of the ray PQ if
ray_PQ.slope == None:
    print("Slope of ray PQ: Undefined (division by zero)") else:
    print("Slope of ray PQ:", ray_PQ.slope)
OUTPUT:

Points P, Q, and R are collinear.
Length of ray PQ: 4
Slope of ray PQ: 00

Q.6) Generate triangle with vertices (0, 0), (4, 0), (1, 4), check whether the triangle is Scalene triangle.

Synatx: import matplotlib.pyplot
as plt # Define the vertices of the
triangle vertices = [(0, 0), (4, 0),
(1, 4)]
Check if the triangle is a scalene triangle def is_scalene(vertices):   x1, y1
= vertices[0]   x2, y2 = vertices[1]   x3, y3 = vertices[2]   return (x1 != x2
and x1 != x3 and x2 != x3) and (y1 != y2 and y1 != y3 and y2 != y3) if
is_scalene(vertices):
    print("The triangle is a scalene triangle.") else:
    print("The triangle is not a scalene triangle.")
# Plot the triangle x = [point[0] for point in
vertices] + [vertices[0][0]] y = [point[1] for point in
vertices] + [vertices[0][1]] plt.plot(x, y, marker='o',

linestyle='-', color='blue') plt.xlabel('x')

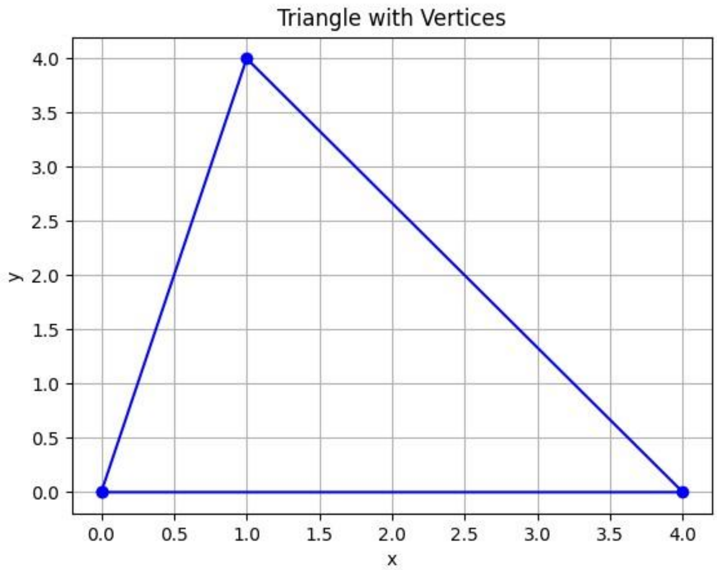
plt.ylabel('y') plt.title('Triangle with Vertices')

plt.grid(True) plt.show()

OUTPUT:

The triangle is not a scalene triangle.

Plot



Q.7)

Write a python program to display the following LPP:

Min Z = 4x+y+3z+5w

subject to

4x+6y-5z-4w >= 20

-8x-3y+3z+2w <= 20

-3x -2y + 4z + w <= 10 x

>= 0,y>= 0,z>= 0,w>= 0 Syntax:

from pulp import *

Create a minimization problem prob

= LpProblem("LPP", LpMinimize) #

Define the decision variables x =

LpVariable("x", lowBound=0) y =

LpVariable("y", lowBound=0) z =

LpVariable("z", lowBound=0) w =

LpVariable("w", lowBound=0) #

Define the objective function prob +=

4*x + y + 3*z + 5*w, "Z"

Define the constraints prob += 4*x

+ 6*y - 5*z - 4*w >= 20 prob += -8*x

- 3*y + 3*z + 2*w <= 20 prob += -

3*x - 2*y + 4*z + w <= 10 # Solve the

problem prob.solve()

Print the status of the problem print("Status:",

LpStatus[prob.status]) # Print the optimal values

of the decision variables print("Optimal values:")

print("x =", value(x)) print("y =", value(y))

print("z =", value(z)) print("w =", value(w))

Print the optimal value of the objective function print("Optimal

Z =", value(prob.objective))

OUTPUT:

Status: Optimal

Optimal values: x =

0.0 y = 3.3333333 z =

0.0 w = 0.0 Optimal Z

= 3.3333333

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 = 150x + 75y
subject to
4x+ 6y <= 24
5x + 3y <= 15 x >=
0,y>= 0 Syntax:

from pulp import *

Create a minimization problem prob =
LpProblem("LPP", LpMinimize) #
Define the decision variables x =
LpVariable("x", lowBound=0) y =
LpVariable("y", lowBound=0) # Define
the objective function prob += 150*x +
75*y, "Z" # Define the constraints prob
+= 4*x + 6*y <= 24 prob += 5*x + 3*y
<= 15
# Solve the problem using the simplex method
prob.solve(PULP_CBC_CMD(msg=False, mip=0))
# Print the status of the problem print("Status:",
LpStatus[prob.status])
# Print the optimal values of the decision variables
print("Optimal values:") print("x =", value(x))
print("y =", value(y))
# Print the optimal value of the objective function
print("Optimal Z =", value(prob.objective))

OUTPUT :
Status: Optimal Optimal
values:
x = 0.0 y = 0.0
Optimal Z = 0.0

Q.9) Write a python program to apply the following transformation on the point (-2,4) :
(I)Reflection through X-axis.
(II)Scaling in X-coordinate by factor 6.
(III) Shearing in X direction by 4 units.
(IV) Rotate about origin through an angle 30

Syntax: import
numpy as np
# Initial point
point = np.array([-2, 4])
# Transformation I: Reflection through X-axis
reflection_x = np.array([[1, 0],[0, -1]])
reflected_point = np.dot(reflection_x, point)
print("Reflection through X-axis:") print("Initial
point:", point)
print("Reflected point:", reflected_point)

```
# Transformation II: Scaling in X-coordinate by factor 6
scaling_x = np.array([[6, 0],[0, 1]]) scaled_point =
np.dot(scaling_x, point) print("\nScaling in X-
coordinate by factor 6:")
print("Initial point:", point) print("Scaled
point:", scaled_point)
# Transformation III: Shearing in X direction by 4 units
shearing_x = np.array([[1, 4],[0, 1]]) sheared_point =
np.dot(shearing_x, point) print("\nShearing in X
direction by 4 units:")
print("Initial point:", point)
print("Sheared point:", sheared_point)
# Transformation IV: Rotate about origin through an angle of 30 degrees angle
= np.deg2rad(30)
rotation   =       np.array([[np.cos(angle), -np.sin(angle)],[np.sin(angle),
np.cos(angle)]])
rotated_point = np.dot(rotation, point)
print("\nRotate about origin through an angle of 30 degrees:") print("Initial
point:", point)
print("Rotated point:", rotated_point)
```

OUTPUT:

```
Reflection through X-axis:
Initial point: [-2  4]
Reflected point: [-2 -4]
Scaling in X-coordinate by factor 6:
Initial point: [-2  4]
Scaled point: [-12   4]
Shearing in X direction by 4 units:
Initial point: [-2  4]
Sheared point: [14  4]
Rotate about origin through an angle of 30 degrees:
Initial point: [-2  4]
Rotated point: [-3.73205081  2.46410162]
```

Q.10) 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:

(I) Rotation about origin through an angle pi/6 (II)
Scaling in Y – Coordinate by -4 unit.

(III) Uniform scaling by -6.4 units (IV)
Shearing in Y direction by 5 units.
Syntax: import
numpy as np
# Initial points
A = np.array([3, 2])
B = np.array([2, -3])
# Transformation I: Rotation about origin through an angle pi/6 angle
= np.pi / 6
rotation = np.array([[np.cos(angle), -np.sin(angle)],[np.sin(angle),
np.cos(angle)]]) rotated_A = np.dot(rotation, A) rotated_B = np.dot(rotation, B)
print("Transformation I: Rotation about origin through an angle pi/6")
print("Rotated point A:", rotated_A) print("Rotated point B:",
rotated_B)
# Transformation II: Scaling in Y-coordinate by -4 units
scaling_y = np.array([[1, 0],[0, -4]]) scaled_A =
np.dot(scaling_y, rotated_A) scaled_B =
np.dot(scaling_y, rotated_B)
print("\nTransformation II: Scaling in Y-coordinate by -4 units")
print("Scaled point A:", scaled_A) print("Scaled
point B:", scaled_B)
# Transformation III: Uniform scaling by -6.4 units
uniform_scaling = np.array([[-6.4, 0],[0, -6.4]])
uniform_scaled_A = np.dot(uniform_scaling, scaled_A)
uniform_scaled_B = np.dot(uniform_scaling, scaled_B)
print("\nTransformation III: Uniform scaling by -6.4 units")
print("Uniform scaled point A:", uniform_scaled_A)
print("Uniform scaled point B:", uniform_scaled_B) #
Transformation IV: Shearing in Y direction by 5 units
shearing_y = np.array([[1, 5],[0, 1]]) sheared_A =
np.dot(shearing_y, uniform_scaled_A) sheared_B =
np.dot(shearing_y, uniform_scaled_B) print("\nTransformation
IV: Shearing in Y direction by 5 units") print("Sheared point
A:", sheared_A)
print("Sheared point B:", sheared_B)

OUTPUT:
Transformation I: Rotation about origin through an angle pi/6
Rotated point A: [1.59807621 3.23205081]
Rotated point B: [3.23205081 -1.59807621]
Transformation II: Scaling in Y-coordinate by -4 units
Scaled point A: [1.59807621 -12.92820323]
Scaled point B: [3.23205081 6.39230485]

Transformation III: Uniform scaling by -6.4 units
Uniform scaled point A: [-10.22768775 82.74050067]
Uniform scaled point B: [-20.68512517 -40.91075101]
Transformation IV: Shearing in Y direction by 5 units
Sheared point A: [403.47481562 82.74050067]
Sheared point B: [-225.23888022 -40.91075101]