Sahakar Maharshi Bhausaheb Santuji Thorat College Sangamner

DEPARTMENT OF COMPUTER SCIENCE

MATHEMATICS

Remark
Demonstrators Signature
Date :- / /2023

Name :- Prem Vijay Vajare **Batch No. :- D** **Roll No:- 75** **Date:-** / /2023

Title of the:- Practical 23 **Expt. No .** 23 **Class :-** S.Y.BCS

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.

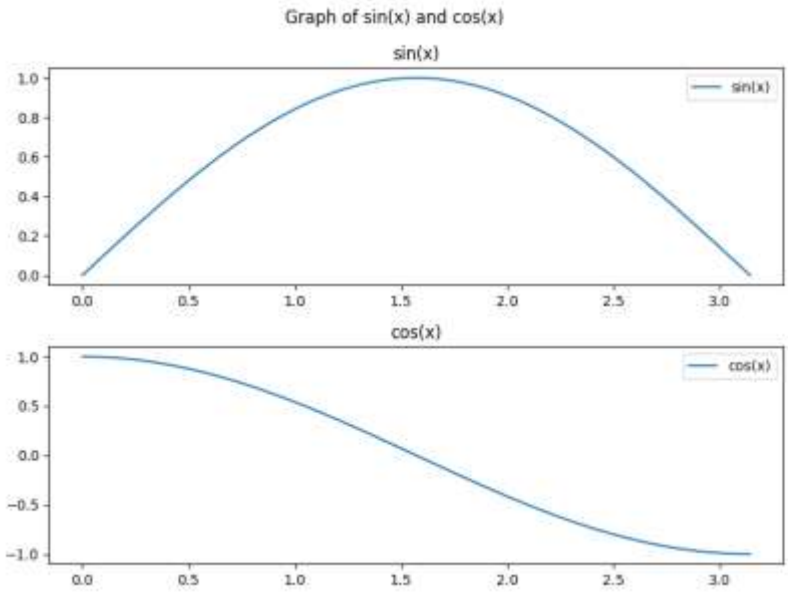
Syntax:

```
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:



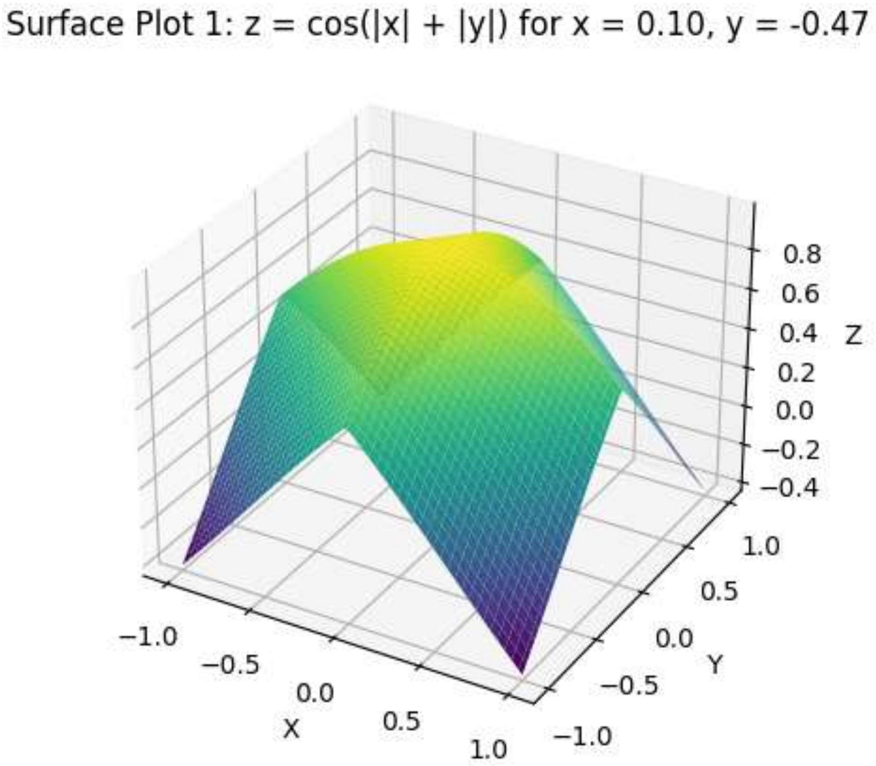
Q.2) Write a python program to plot 30 Surface Plot of the function z = cos(|x| +|y|) in -1 < x,y1 < 1.

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:



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]

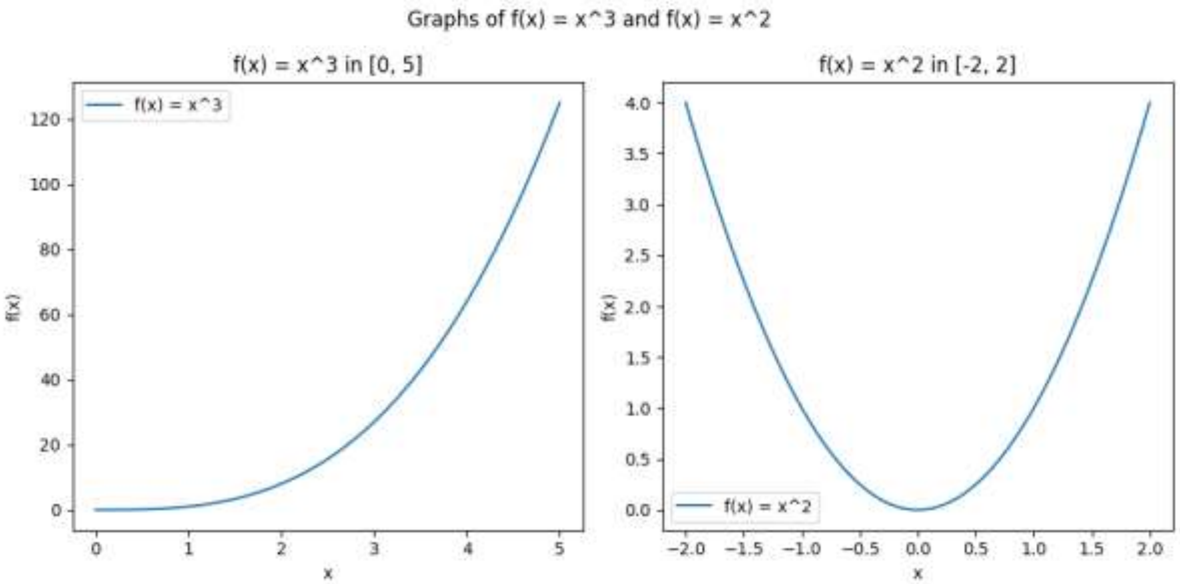
Syntax:

```
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:



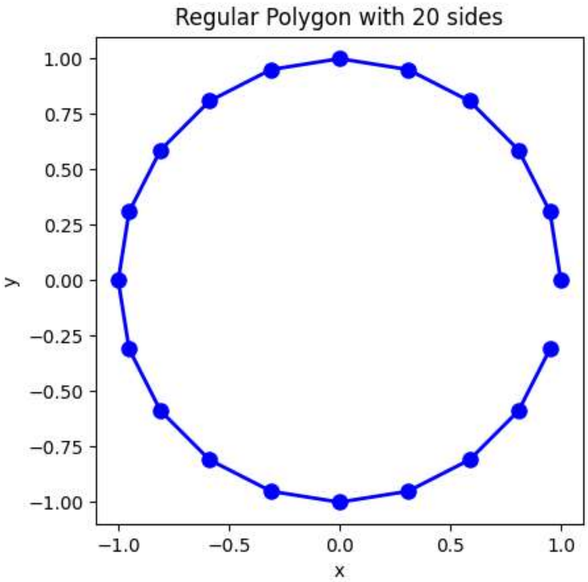
Q.4) Write a python program to draw regular polygon with 20 sides and radius 1 centered at (0,0)

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:



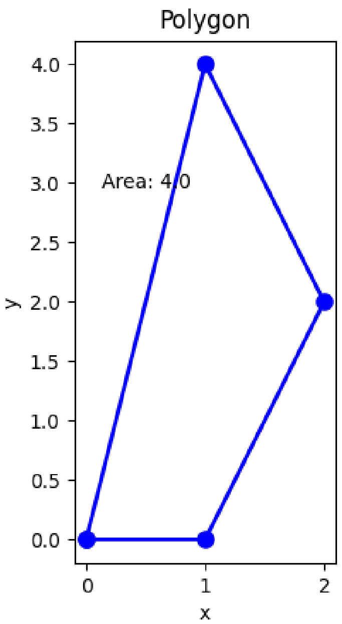
Q.5) Write a Python program to draw a polygon with vertices (0,0),(1,0),(2,2),(1,4). Also find area of polygon.

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)
```