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:- 04** **Date:-** / /2023

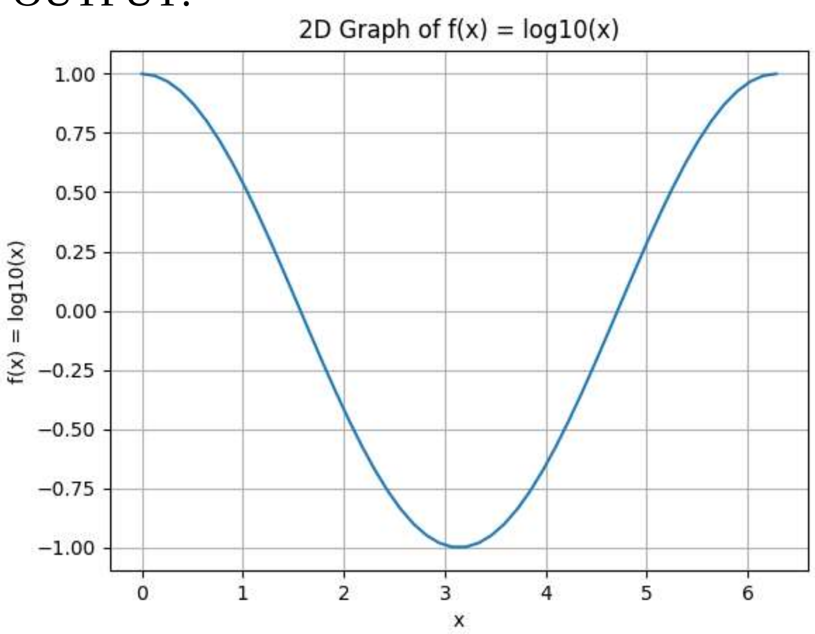
Title of the:- Practical 3 **Expt. No .** 3 **Class :-** S.Y.BCS

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:

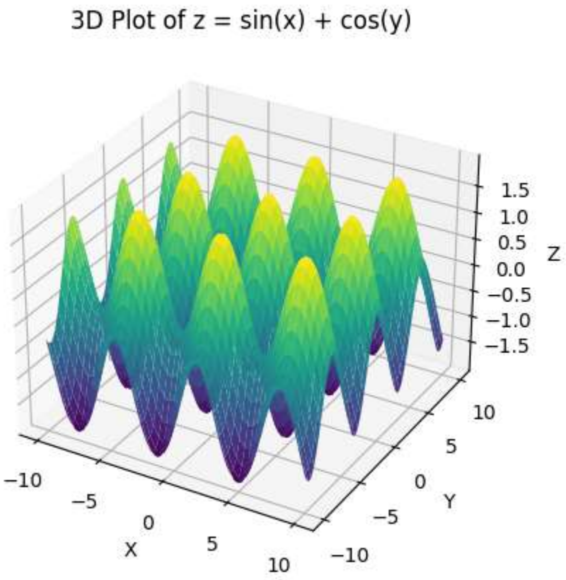


Q.2) Write a Python program to generate 3D plot of the functions z = sin x +cos y in -10 < x, y < 10.

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:



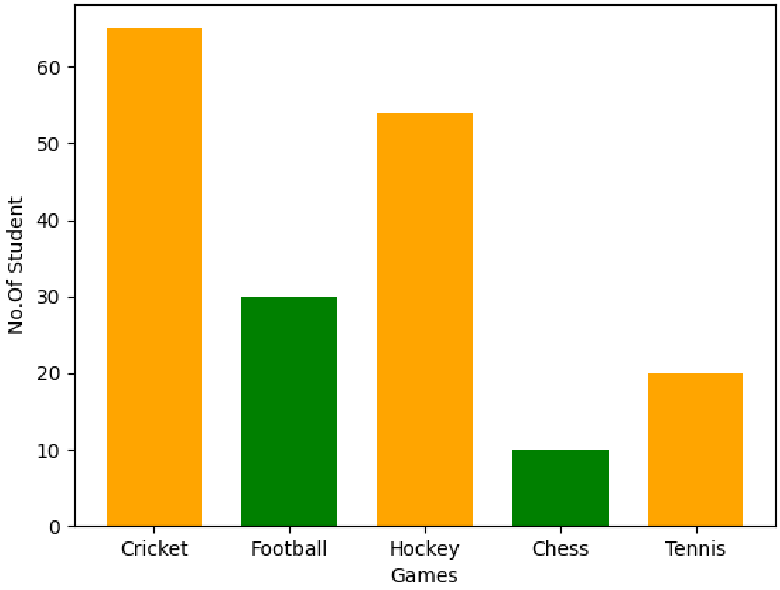
Q.3) Following is the information of student participating in various games in school. Represent it by a Bar graph with bar width 0.7 inches.

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]