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

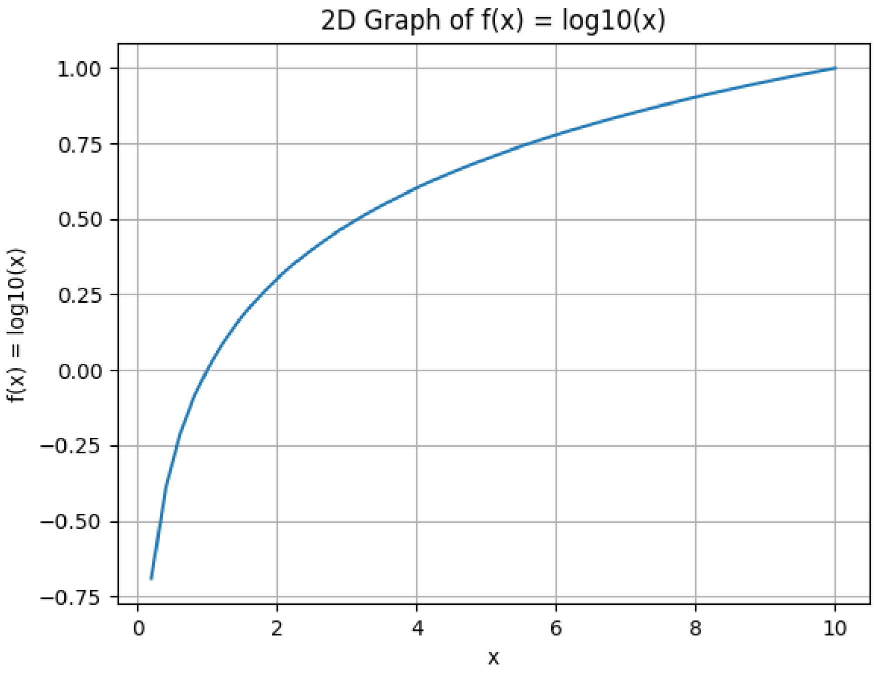
Title of the:- Practical 2 **Expt. No .** 2 **Class :-** S.Y.BCS

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:

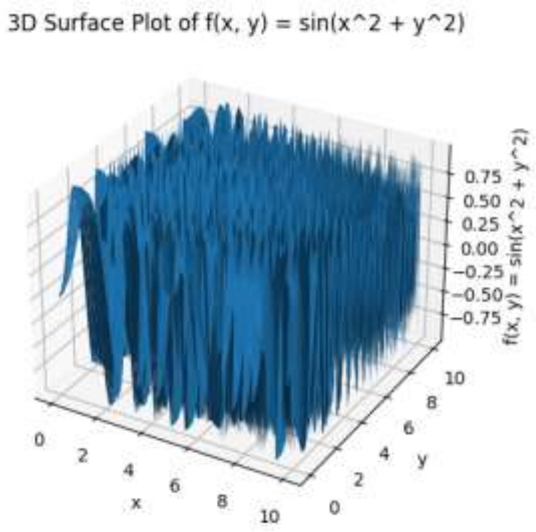


Q.2) Using python, generate 3D surface Plot for the function $f(x) = \sin(x^2 + y^2)$ in the interval [0,10]

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:



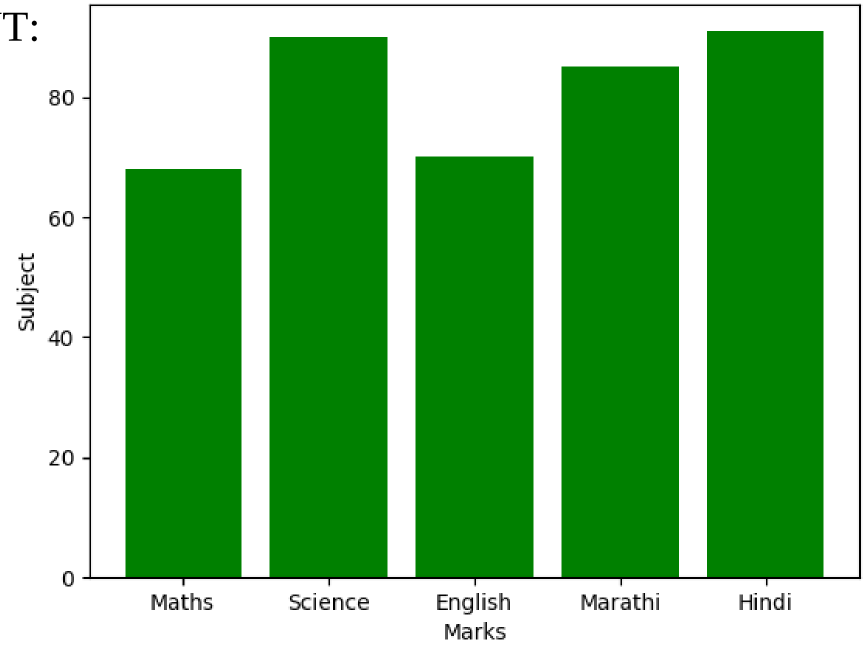
Q.3) Using python, represent the following information using a bar graph (in green color)

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
```

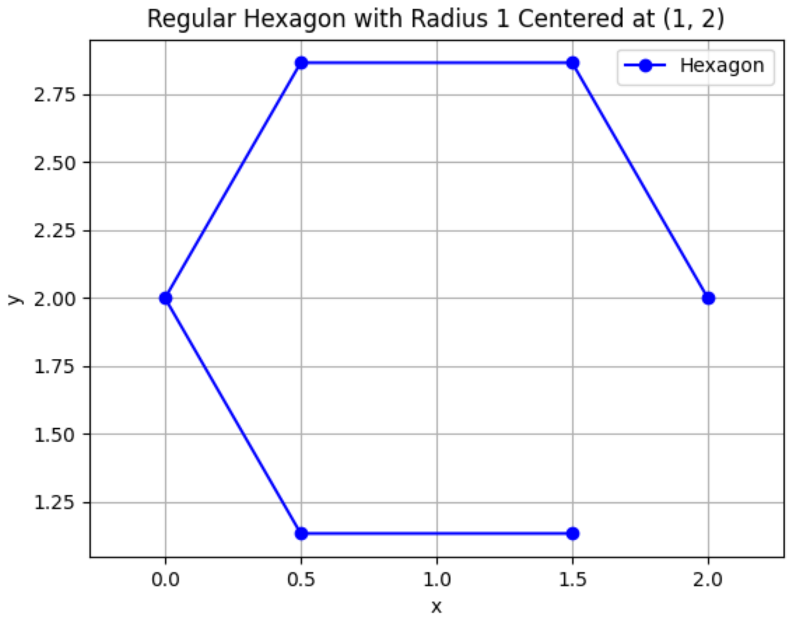
Calculate the perimeter of the hexagon

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]