Rao Bahadur Y. Mahabaleswarappa Engineering College

Department of Mathematics

LAB MANUAL

of

First Semester Engineering Mathematics

Academic Year: 2025-2026

Contents

Lab 1: 2D-Plots of Cartesian and Polar Curves

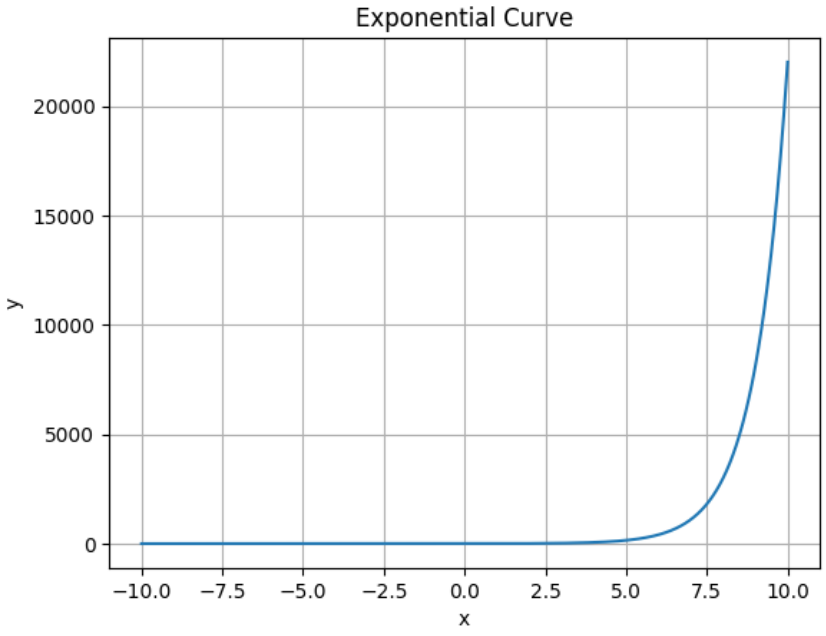
Aim: To plot Cartesian curves, implicit curves and polar functions

1.1 Write a program to plot the Exponential curve $y = e^x$

Code:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define range for x values
x = np.arange(-10, 10, 0.001)

# Define the exponential function y = e^x
y = np.exp(x)

# Plot the function
plt.plot(x, y)

# Add title and labels
plt.title(r"Exponential Curve")
plt.xlabel("x")
plt.ylabel("y")

# Add grid
plt.grid()

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

Output:

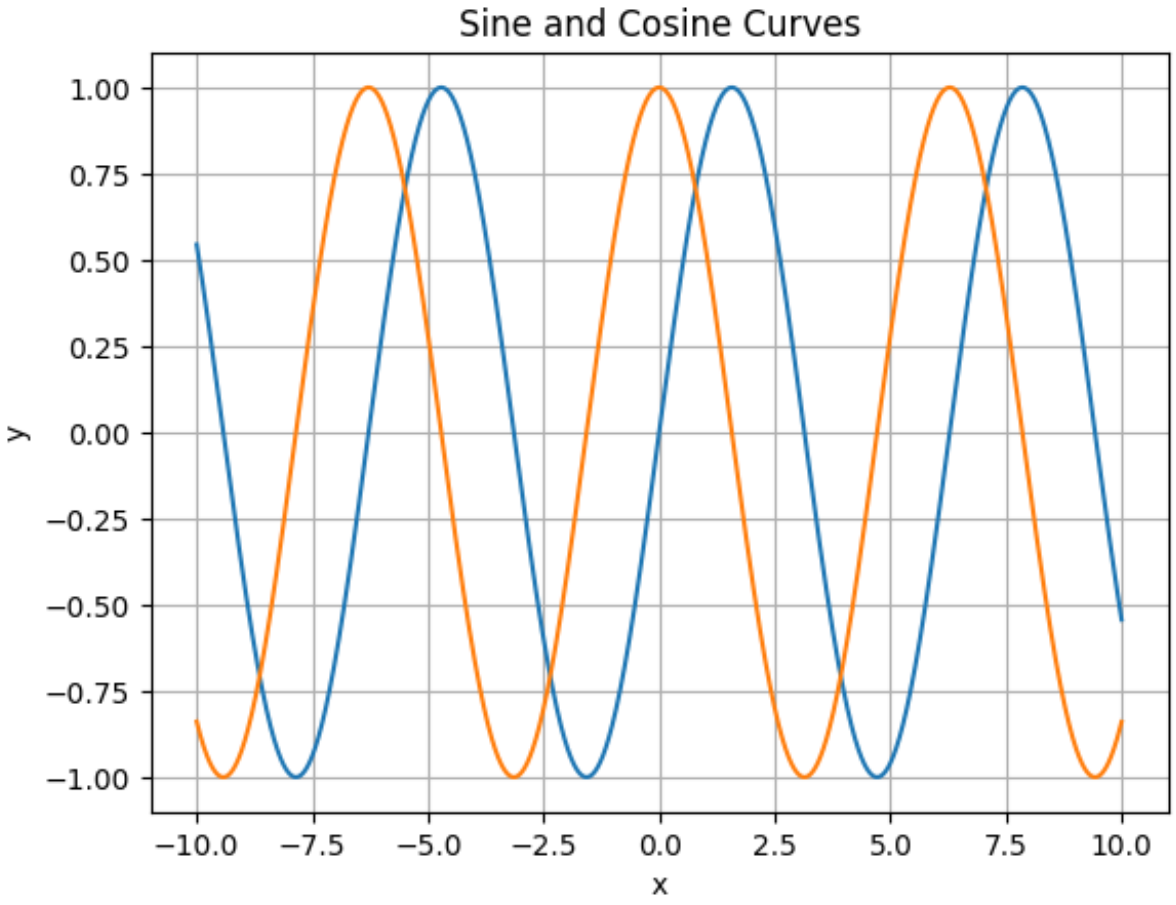


1.2 Write a program to plot Sine and Cosine curves

Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Define x values
x = np.arange(-10, 10, 0.001)

# Define functions
y1 = np.sin(x)
y2 = np.cos(x)

# Plot sine and cosine curves
plt.plot(x, y1)
plt.plot(x, y2)
plt.title("Sine and Cosine Curves")
plt.xlabel("x")
plt.ylabel("y")
plt.grid()
plt.show()
```

Output:

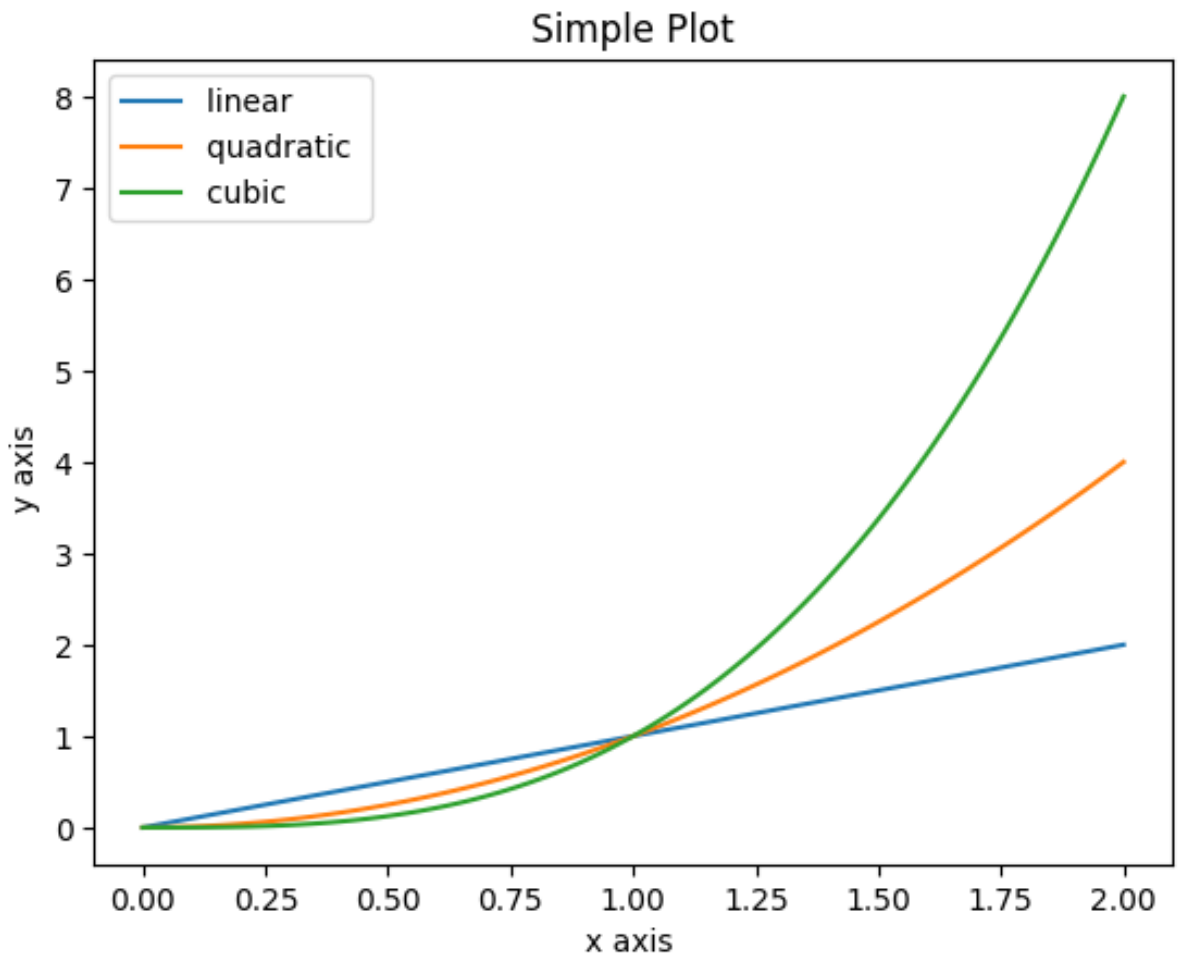


1.3 Write a program to plot Linear, Quadratic and Cubic curves

Code:

```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(0, 2, 100)

plt.plot(x, x, label ='linear ')
plt.plot(x, x**2, label ='quadratic ')
plt.plot(x, x ** 3, label ='cubic ')

plt.xlabel('x axis ')
plt.ylabel('y axis ')
plt.title(" Simple Plot ")
plt.legend()
plt.show()
```

Output:



1.4:Write a program to plot a circle given by the equation

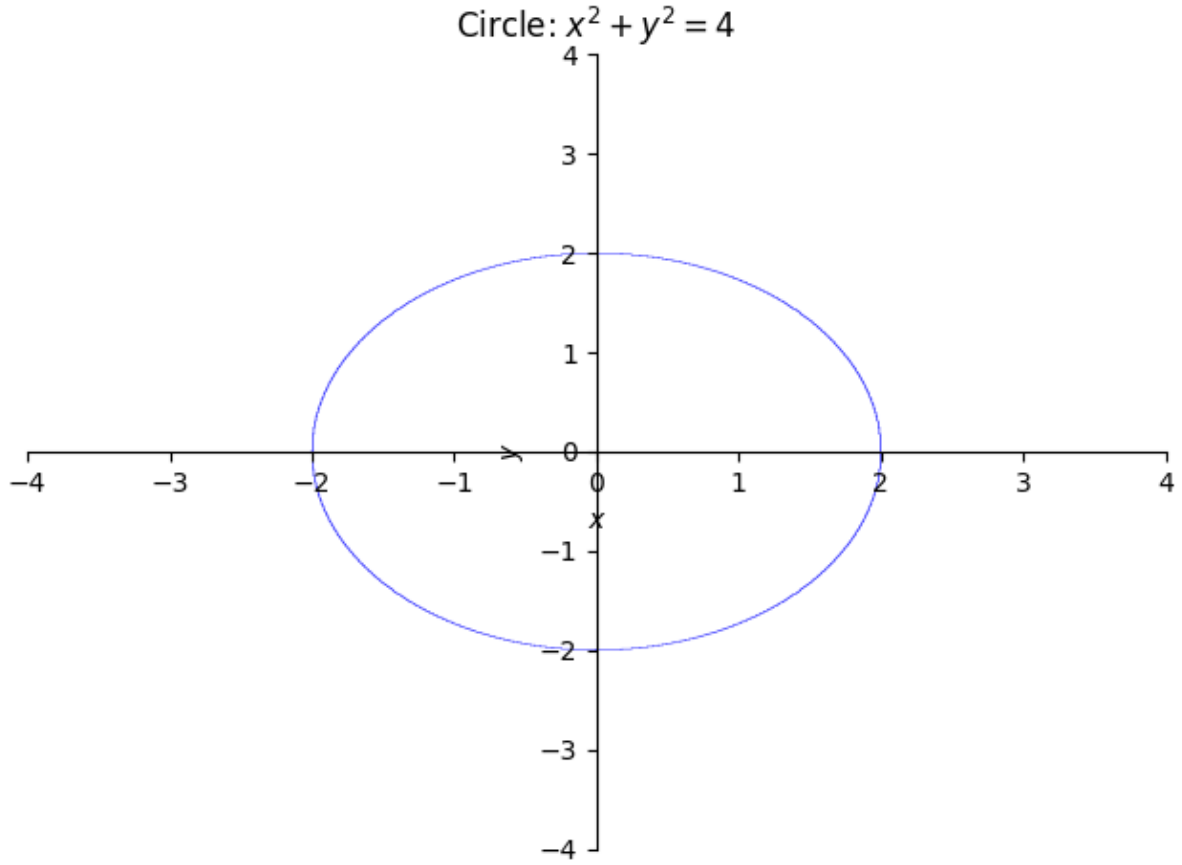
$$x^2 + y^2 = 4$$

(i.e., a circle with centre $(0, 0)$ and radius $r = 2$)

Code:

```
from sympy import *

# Define symbols
x, y = symbols('x y')

# Plot the circle x^2 + y^2 = 4
p1 = plot_implicit(
    Eq(x**2 + y**2, 4),
    (x, -4, 4),(y, -4, 4),
    title='Circle: $x^2 + y^2 = 4$'
    )
```

Output:



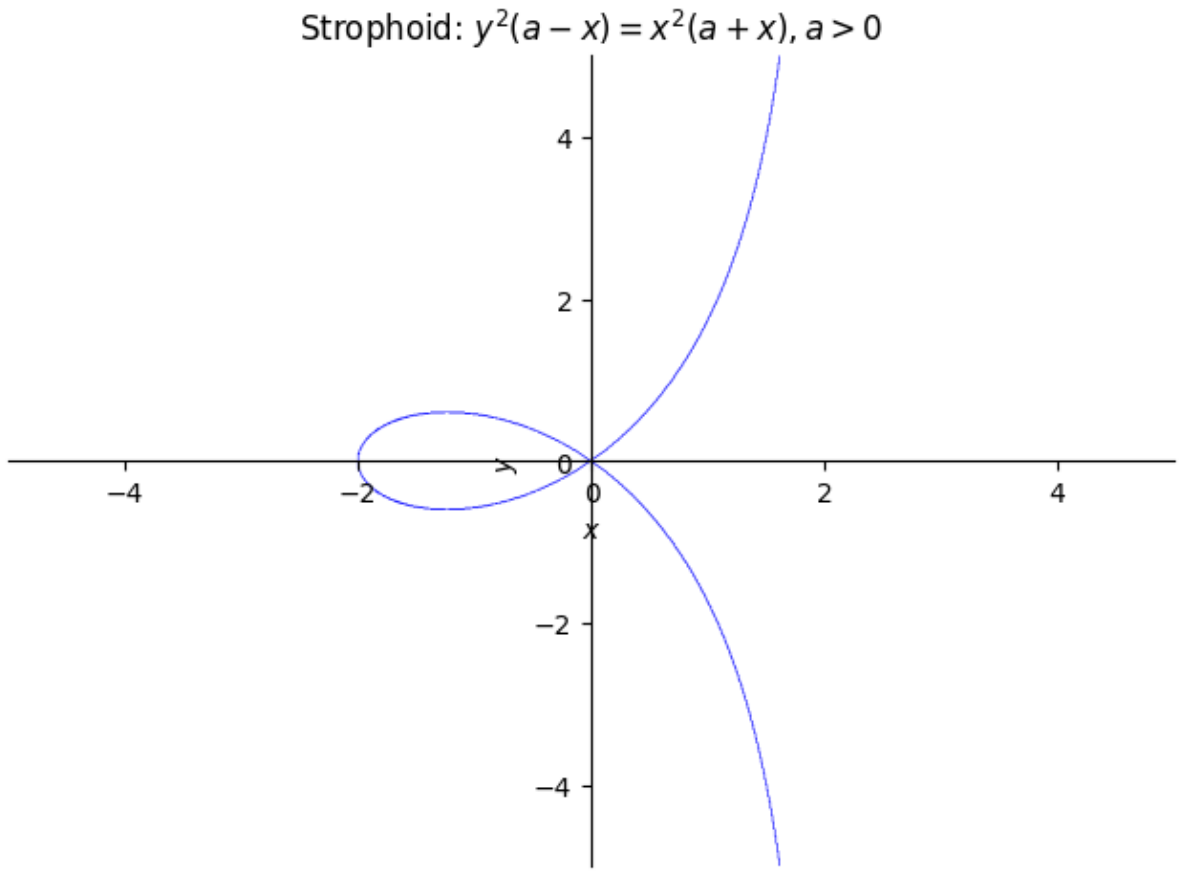
1.5: Write a program to plot the strophoid given by the equation

$$y^2(a-x) = x^2(a+x), \quad a > 0$$

Code:

```
from sympy import *

# Define symbols
x, y = symbols('x y')
a = 2  # parameter a>0

# Plot the strophoid: y^2*(a-x) = x^2*(a+x)
p1 = plot_implicit(
    Eq(y**2 * (a - x), x**2 * (a + x)),
    (x, -5, 5), (y, -5, 5),
    title='Strophoid: $y^2 (a-x) = x^2 (a+x), a>0$'
)
```

Output:



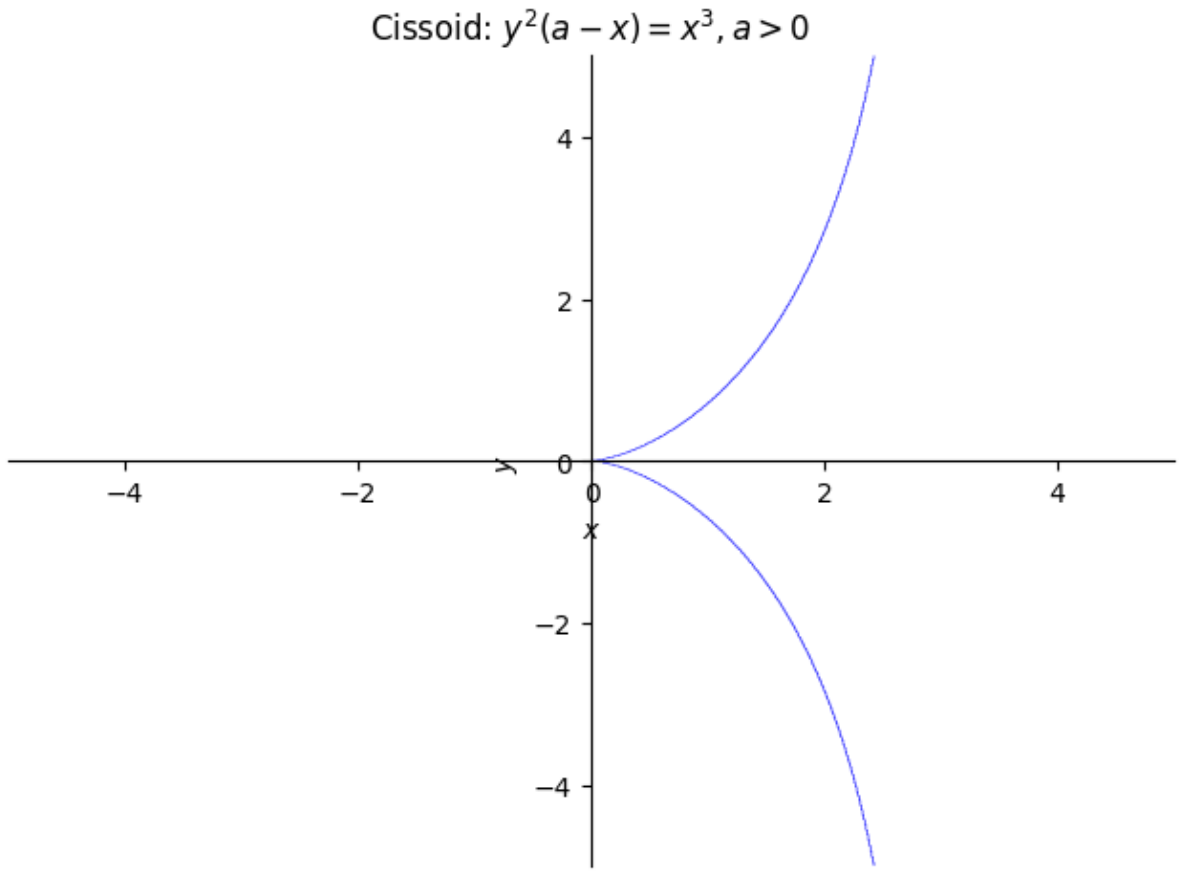
1.6: Write a program to plot the cissoid given by the equation

$$y^2(a-x) = x^3, \quad a > 0$$

Code:

```
from sympy import *

# Define symbols
x, y = symbols('x y')
a = 3  # parameter a>0

# Plot the cissoid: y^2 * (a - x) = x^3
p1 = plot_implicit(
    Eq(y**2 * (a - x), x**3),
    (x, -5, 5),(y, -5, 5),
    title='Cissoid: $y^2 (a-x) = x^3, a>0$',
)
```

Output:



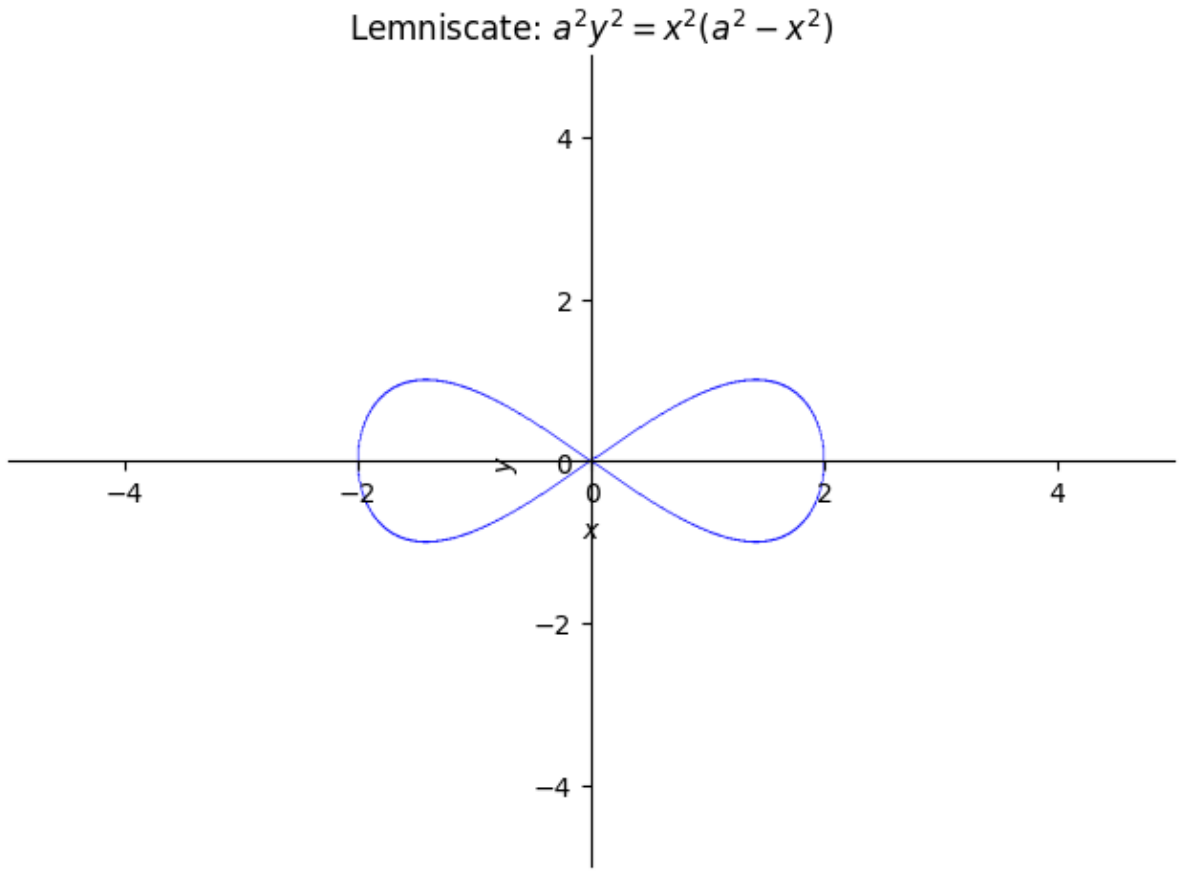
1.7: Write a program to plot the lemniscate given by the equation

$$a^2y^2 = x^2(a^2 - x^2)$$

Code:

```
from sympy import *

# Define symbols
x, y = symbols('x y')
a = 2  # parameter a>0

# Plot the lemniscate: a^2*y^2 = x^2*(a^2 - x^2)
p1 = plot_implicit(
    Eq(a**2 * y**2, x**2 * (a**2 - x**2)),
    (x, -5, 5), (y, -5, 5),
    title='Lemniscate: $a^2 y^2 = x^2 (a^2 - x^2)$',
   )
```

Output:



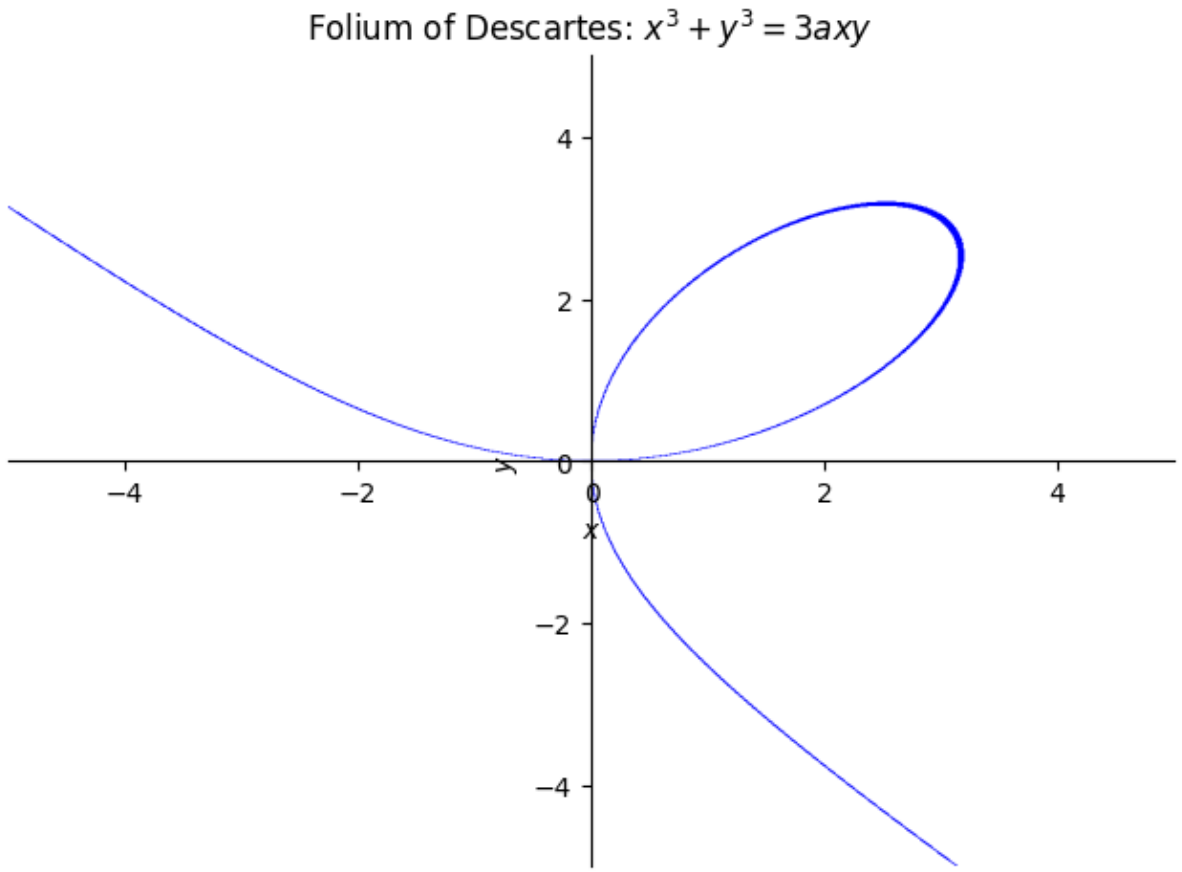
1.8: Write a program to plot the Folium of Descartes given by the equation

$$x^3 + y^3 = 3axy$$

Code:

```
from sympy import *

# Define symbols
x, y = symbols('x y')
a = 2  # parameter a>0

# Plot Folium of Descartes: x^3 + y^3 = 3*a*x*y
p1 = plot_implicit(
    Eq(x**3 + y**3, 3*a*x*y),
    (x, -5, 5), (y, -5, 5),
    title='Folium of Descartes: $x^3 + y^3 = 3 a x y$',
   )
```

Output:



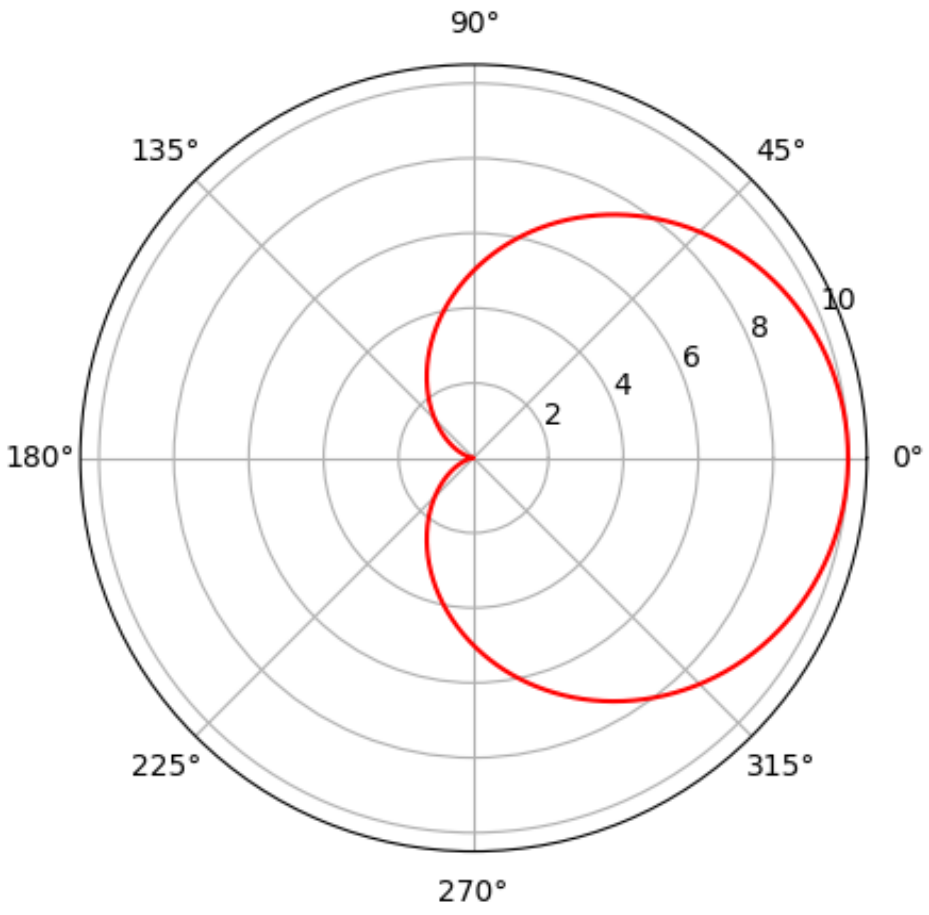
1.9: Write a program to plot the cardioid given by the polar equation

$$r = 5(1 + \cos\theta)$$

Code:

```
from pylab import *
import numpy as np

# Define theta
theta = linspace(0, 2*np.pi, 1000)

# Define r for cardioid: r = 5 + 5*cos(theta)
r1 = 5 + 5 * cos(theta)

# Polar plot
polar(theta, r1, 'r')  # 'r' specifies red color

# Show the plot
show()
```

Output:



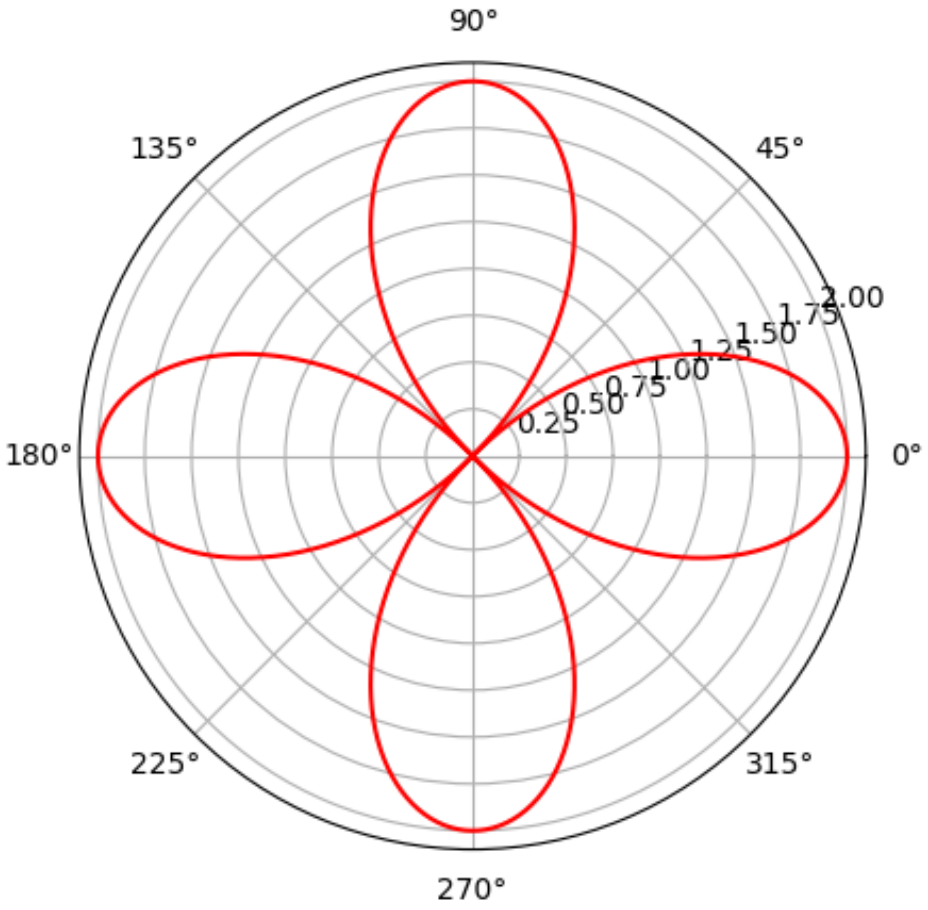
1.10: Write a program to plot the four-leaved rose given by the polar equation

$$r = 2\,|\cos 2\theta|$$

Code:

```
from pylab import *
import numpy as np

# Define theta
theta = linspace(0, 2*pi, 1000)

# Define r for four-leaved rose
r = 2 * abs(cos(2 * theta))

# Polar plot
polar(theta, r, 'r')  # 'r' for red color

# Show the plot
show()
```

Output:



Lab 2: Finding angle between two polar curves

Aim: To find angle between two polar curves.

2.1: Write a program Find the angle between the curves

$$r_1 = 4(1 + \cos t), \quad r_2 = 5(1 - \cos t)$$

Code:

```
from sympy import *

# Define symbols
r, t = symbols('r t')

# Define the polar curves
r1 = 4*(1 + cos(t))
r2 = 5*(1 - cos(t))

# Compute derivatives with respect to t
dr1 = diff(r1, t)
dr2 = diff(r2, t)

# Tangent slopes in polar form: dr/dtheta
t1 = r1 / dr1
t2 = r2 / dr2

# Solve r1 - r2 = 0 for points of intersection
q = solve(r1 - r2, t)

# Evaluate slopes at intersection (choose second solution)
w1 = t1.subs(t, float(q[1]))
w2 = t2.subs(t, float(q[1]))

# Angle between curves
y1 = atan(w1)
y2 = atan(w2)
w = abs(y1 - y2)

# Display result
print('Angle between curves in radians is %0.3f' % w)
```

Output:

```
Angle between curves in radians is 1.571
```

Exercise: Write a program Find the angle between the curves

$$r_1 = 4\cos t \quad \text{and} \quad r_2 = 5\sin t$$

Lab 3: Finding Radius of curvature

Aim: To find radius of curvature

3.1: Write a program to Find the radius of curvature of the curve $r = 4(1+\cos t)$ at $t = \frac{\pi}{2}$.

Code:

```
from sympy import *

# Define symbols
t = symbols('t')

# Define polar curve
r = 4 * (1 + cos(t))

# First derivative dr/dt
r1 = diff(r,t)

# Second derivative d^2r/dt^2
r2 = diff(r1, t)

# Radius of curvature formula in polar coordinates
rho = (r**2 + r1**2)**(3/2) / (r**2 + 2*r1**2 - r*r2)

# Evaluate at t = pi/2
rho_val = rho.subs(t, pi/2)

print('The radius of curvature is %0.4f units' % rho_val)
```

Output:

```
The radius of curvature is 3.7712 units
```

3.2: Find the radius of curvature of the curve

$$x = a\cos t, \quad y = a\sin t$$

as a function of t **Code:**

```
from sympy import *

# Define symbols
t, a = symbols('t a')

# Parametric equations
x = a * cos(t)
y = a * sin(t)

# First derivatives
dx_dt = diff(x, t)
dy_dt = diff(y, t)

# Total derivative dy/dx
dydx = simplify(dy_dt / dx_dt)

# Second derivative
d2ydx2 = simplify(diff(dydx, t) / dx_dt)

# Formula for radius of curvature
rho = simplify((1 + dydx**2)**(3/2) / d2ydx2)
print('Radius of curvature (general form):')
display(rho)

# Substitution at t = pi/2 and a = 5
rho_val = rho.subs({t: pi/2, a: 5})
print('\nRadius of curvature at t = pi/2 and a = 5:')
display(simplify(rho_val))

# Curvature = 1 / radius of curvature
curvature = 1 / rho_val
print('\nCurvature at (a=5, t=pi/2) =', float(curvature))
```

Output: Radius of Curvature (General Form):

$$-a\left(\frac{1}{\sin^2(t)}\right)^{1.5}\sin^3(t)$$

Radius of Curvature at $t = \frac{\pi}{2}$, $a = 5$:

$$-5$$

Curvature at $(a = 5, t = \frac{\pi}{2}) = -0.2$:

Exercise 1: Write a program to Find the radius of curvature of the curve $r = a\sin(nt)$ **at** $t = \pi/2$ **with** $n = 1$**.**

Exercise 2: Find the radius of curvature of the curve

$$x = a\cos^{\frac{3}{2}}(t), \quad y = a\sin^{\frac{3}{2}}(t).$$

as a function of t

Lab 4: Expansion of Taylor's and Maclaurin's series

Aim: To compute and visualize the Taylor and Maclaurin series expansions of a given function.

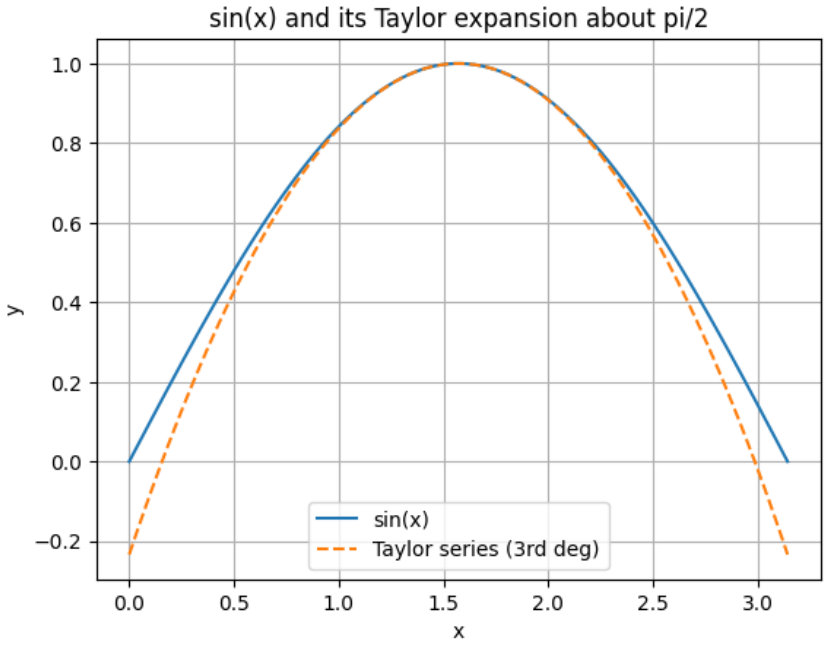
4.1: Write a program to expand $\sin(x)$ as a Taylor series about $x = \frac{\pi}{2}$ up to the 3rd-degree term. Also, find the value of $\sin(100°)$.

Code:

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Define the variable
x = sp.symbols('x')

# Function
f = sp.sin(x)

# Expansion point
a = sp.pi/2

# Taylor series up to 3rd degree term
taylor_series = f.series(x, a, 4).removeO() #upto 3rd deg
print("Taylor series of sin(x) about x = pi/2 up to 3rd degree
   term:")
print(taylor_series)

# Plot the function and its Taylor series
x_vals = np.linspace(0, np.pi, 100)
f_vals = np.sin(x_vals)
taylor_func = sp.lambdify(x, taylor_series)
taylor_vals = taylor_func(x_vals)

plt.plot(x_vals, f_vals, label='sin(x)')
plt.plot(x_vals, taylor_vals,'--',label='Taylor series(3rd deg)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('sin(x) and its Taylor expansion about pi/2')
plt.legend()
plt.grid()
plt.show()

# Evaluate sin(100  )
deg = 100
rad = deg*np.pi/180
sin_val = np.sin(rad)
print(f"sin({deg}  ) =", sin_val)
```

Output

```
Taylor series of sin(x) about x = pi/2 up to 3rd degree term:
1 - (x - pi/2)**2/2
```




```
sin(100°) = 0.984807753012208
```

4.2: Write a program to expand $\sin(x)$ as a Taylor series about $x = \frac{\pi}{2}$ up to the 3rd-degree term. Also, find the value of $\sin(100°)$.

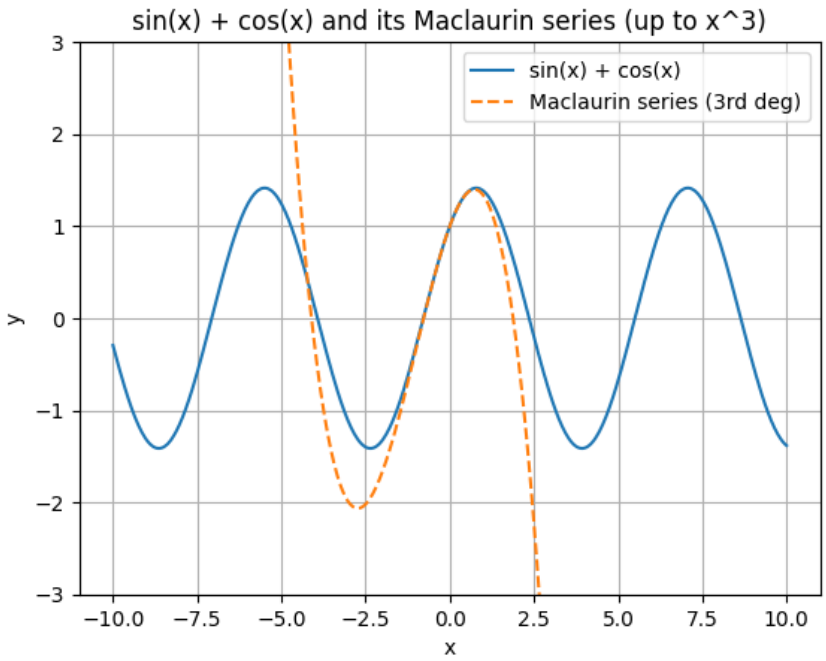
Code:

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Define the variable
x = sp.symbols('x')

# Function
f = sp.sin(x) + sp.cos(x)

# Maclaurin series up to 3rd-degree term (x^3)
maclaurin_series = f.series(x, 0, 4).removeO()  # up to 3rd-deg
print("Maclaurin series of sin(x)+cos(x) up to 3rd-degree term:")
print(maclaurin_series)

# Plot the function and its Maclaurin series
x_vals = np.linspace(-10, 10, 400)
f_vals = np.sin(x_vals) + np.cos(x_vals)
maclaurin_func = sp.lambdify(x, maclaurin_series)
maclaurin_vals = maclaurin_func(x_vals)
```

```
21
22 plt.plot(x_vals, f_vals, label='sin(x) + cos(x)')
23 plt.plot(x_vals, maclaurin_vals, '--', label='Maclaurin series')
24 plt.xlabel('x')
25 plt.ylabel('y')
26 plt.ylim([-3,3])
27 plt.title('sin(x) + cos(x) and its Maclaurin series (up to x^3)')
28 plt.grid()
29 plt.legend()
30 plt.show()
31
32 # Evaluate sin(10) + cos(10) in radians
33 value_series = maclaurin_func(10*np.pi/180)
34 print("Approximate sin(10)+cos(10) using series =", simplify(
     value_series))
```

Output

```
Maclaurin series of sin(x)+cos(x) up to 3rd-degree term:
-x**3/6 - x**2/2 + x + 1
```




```
Approximate sin(10)+cos(10) using series = 1.15841595805440
```

Lab 5: Finding partial derivatives and Jacobian Functions of Several Variables

Aim: To find partial derivatives and Jacobian of functions of several variables.

5.1 Write a program to prove that mixed partial derivatives

$$u_{xy} = u_{yx} \quad \text{for} \quad u = e^x \left(x\cos(y) - y\sin(y)\right).$$

Code:

```
from sympy import *

# Define variables
x, y = symbols('x y')

# Define the function
u = exp(x) *( x*cos(y) - y*sin(y))

# Partial derivatives
dux = diff(u, x)
duy = diff(u, y)

# Mixed partials
duxy = diff(dux, y)
duyx = diff(duy, x)

# Check equality
if duxy == duyx:
    print("Mixed partial derivatives are equal")
else:
    print("Mixed partial derivatives are not equal")
```

Output:

```
Mixed partial derivatives are equal
```

5.2: Write a program to prove that if

$$u = \tan^{-1}\left(\frac{y}{x}\right)$$

then $u_{xx} + u_{yy} = 0$.

Code:

```
from sympy import *

# Define variables
x, y = symbols('x y')

# Define the function
u = atan(y/x)


# First derivatives
dux = diff(u, x)
duy = diff(u, y)

# Second derivatives
uxx = diff(dux, x)    # or diff(u, x, 2)
uyy = diff(duy, y)    # or diff(u, y, 2)

# Sum of second derivatives
w = uxx + uyy

# Simplify result
w1 = simplify(w)

print("Ans:", w1)
```

Output:

```
Ans: 0
```

5.3: Write a program to compute the Jacobian determinant of the transformation

$$u = \frac{xy}{z}, \quad v = \frac{yz}{x}, \quad w = \frac{zx}{y}.$$

Code:

```
from sympy import *

# Define variables
x, y, z = symbols('x y z')

# Define functions
u = x*y/z
v = y*z/x
w = z*x/y

# Partial derivatives
dux = diff(u, x); duy = diff(u, y); duz = diff(u, z)
dvx = diff(v, x); dvy = diff(v, y); dvz = diff(v, z)
dwx = diff(w, x); dwy = diff(w, y); dwz = diff(w, z)

# Construct Jacobian matrix
J = Matrix([[dux, duy, duz],
            [dvx, dvy, dvz],
            [dwx, dwy, dwz]])

print("The Jacobian matrix is:\n")
display(J)

# Determinant of Jacobian
Jac = J.det()
print("\nJ =", Jac)
```

Output:

$$J = \begin{bmatrix} \frac{y}{z} & \frac{x}{z} & -\frac{xy}{z^2} \\ -\frac{yz}{x^2} & \frac{z}{x} & \frac{y}{x} \\ \frac{z}{y} & -\frac{zx}{y^2} & \frac{x}{y} \end{bmatrix}$$

$$J = 4$$

5.4 Write a program to compute the Jacobian matrix and its determinant for

$$u = x + 3y^2 - z^3, \quad v = 4x^2yz, \quad w = 2z^2 - xy$$

and evaluate the determinant at $(x, y, z) = (1, -1, 0)$**.**

Code:

```
from sympy import *

# Define variables
x, y, z = symbols('x y z')

# Define functions
u = x + 3*y**2 - z**3
v = 4*x**2*y*z
w = 2*z**2 - x*y

# Partial derivatives
dux, duy, duz = diff(u, x), diff(u, y), diff(u, z)
dvx, dvy, dvz = diff(v, x), diff(v, y), diff(v, z)
dwx, dwy, dwz = diff(w, x), diff(w, y), diff(w, z)

# Jacobian matrix
J = Matrix([
    [dux, duy, duz],
    [dvx, dvy, dvz],
    [dwx, dwy, dwz]
])

print("The Jacobian matrix is:\n")
display(J)

# Determinant of Jacobian
Jac = J.det()
print("\nJ =")
display(Jac)

# Substituting values (x=1, y=-1, z=0)
J1 = J.subs({x:1, y:-1, z:0})
print("\nJ at (1, -1, 0):\n")
display(J1)

Jac1 = J1.det()
print("\nDeterminant of J at (1, -1, 0):")
display(Jac1)
```

Output:

$$J = \begin{bmatrix} 1 & 6y & -3z^2 \\ 8xyz & 4x^2z & 4x^2y \\ -y & -x & 4z \end{bmatrix}$$

$$J = 32x^3yz + 24xyz^3 + 24x^2y^2z - 32x^2z^2$$

At $(x, y, z) = (1, -1, 0)$:

$$J = \begin{bmatrix} 1 & -6 & 0 \\ 0 & 0 & -4 \\ 1 & -1 & 0 \end{bmatrix}$$

$$\det(J) = 20$$

.

.

Exercise 1:If $u = \log(\frac{x^2+y^2}{x+y})$, **write a program show that** $xu_x + yu_y = 1$.

Exercise 2:Write a program to find $\dfrac{\partial(u,v,w)}{\partial(x,y,z)}$ **for**

$$u = x + 3y^2 - z^3, \quad v = 4x^2yz, \quad w = 2z^2 - xy \text{ at the point } (-2,-1,1).$$

Lab 6: Solution of ordinary differential equations.

Aim: To solve first-order and higher-order ordinary differential equations (ODEs).

6.1 Write a program to Solve the differential equation

$$\frac{dy}{dx} + \tan(x) - y^3 \sec x = 0$$

Code:

```
from sympy import *

# Define variable and function
x = symbols('x')
y = Function('y')(x)

y1= Derivative (y,x)

# Define the differential equation: dy/dx+y*tan(x)-y**3*sec(x)=0
z = Eq(y1 + y * tan(x) - y**3 * sec(x), 0)

# Solve the differential equation
sol = dsolve(z, y)

# Display the solution
display(sol)
```

Output:

$$y(x) = -\sqrt{\frac{\cos(x)}{C_1 - 2\sin(x)}}, \quad y(x) = \sqrt{\frac{\cos(x)}{C_1 - 2\sin(x)}}$$

Exercise 1:Write a program to Solve the differential equation

$$x^3\frac{dy}{dx} - x^2y + y^4\cos(x) = 0$$

Lab 7: Plotting solutions of ODE.

Aim: To represent the solution graphically.

7.1 A culture initially has P_0 number of bacteria. At $t = 1$ hour, the number of bacteria is measured to be $\frac{3}{2}P_0$. If the rate of growth is proportional to the number of bacteria $P(t)$ present at time t, determine the time necessary for the number of bacteria to triple.

The differential equation is:

$$\frac{dP}{dt} = kP, \quad P(1) = \frac{3}{2}P_0.$$

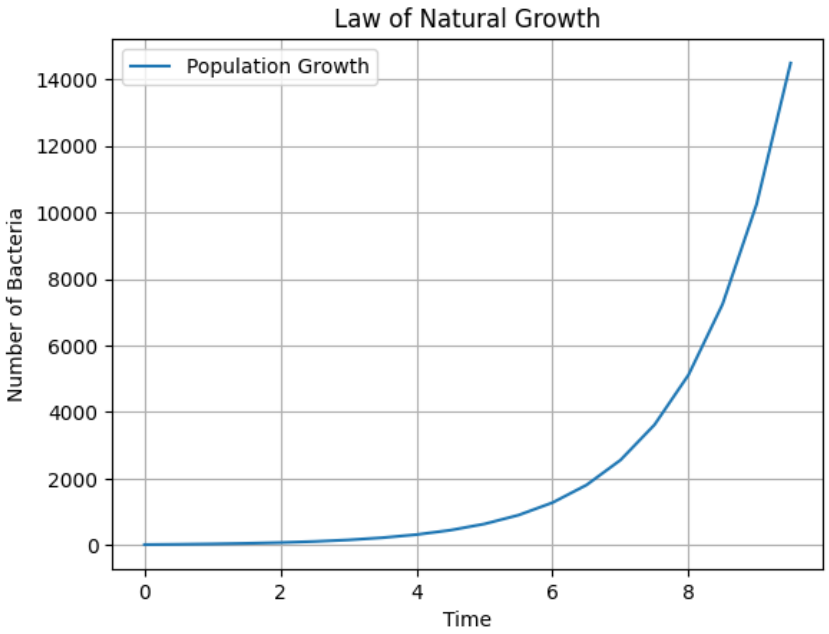
The solution is:

$$y = P_0 e^{(k)t}, \quad y_0 = 20, k = \log 2.$$

Code:

```
import numpy as np
import matplotlib.pyplot as plt

# Time range
t = np.arange(0, 10, 0.5)

# Initial population
P0 = 20

# Growth rate (example: k = ln(2)     0.693 for doubling)
k = np.log(2)  # you can change as needed

# Population at time t
y = P0 * np.exp(k * t)

# Plot
plt.plot(t, y, label='Population Growth')
plt.xlabel('Time')
plt.ylabel('Number of Bacteria')
plt.title('Law of Natural Growth')
plt.grid()
plt.legend()
plt.show()
```

Output:



Lab 7.2: The solution of the mathematical representation of Newton's Law of Cooling is given by:

$$T(t) = t_2 + (t_1 - t_2)e^{-kt},$$

where

- $T(t)$ = temperature of the body at time t,
- T_1 = initial temperature of the body,
- T_2 = surrounding temperature,
- k = thermal conductivity of the material.

A body initially at $T_1 = 100°\text{C}$ cools to $T = 75°\text{C}$ in 10 minutes, where the surrounding temperature is $T_2 = 20°\text{C}$.

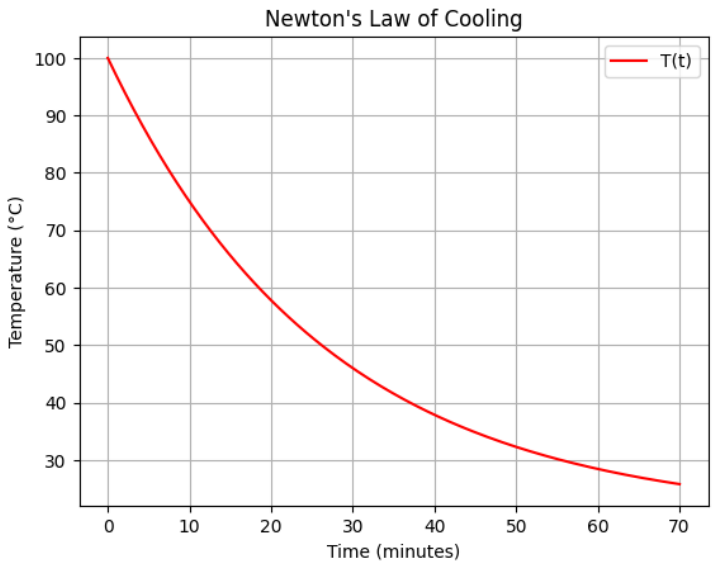
1. Determine the temperature of the body after 30 minutes.
2. Plot the cooling curve of the body using the given data.

Code:

```
import numpy as np
from sympy import symbols, Function, exp, log, lambdify
import matplotlib.pyplot as plt

# Given data
t1 = 100        # Initial temperature ( C )
t2 = 20         # Surrounding temperature ( C )

# one reading t =10 minute temp is 75 degree
t = 10     # Time of measurement (minutes)
T = 75     # Temperature at t_meas

```

```
# Calculate Thermal conductivity of material
k_val = (1/t) * log((t1 - t2)/(T - t2))
print("Thermal conductivity=", k_val)

# Define symbolic variables
t,k = symbols('t k')
T = Function('T')(t)

# Newton's Law of Cooling solution
T = t2 + (t1 - t2) * exp(-k*t)
print("Symbolic solution T(t) =", T)

# Substitute numerical k value for plotting
T_num = T.subs(k, k_val)

# Convert to numerical function
T_func = lambdify(t, T_num)

# Time array for plotting
t_vals = np.linspace(0, 70, 200)

# Plot the cooling curve
plt.plot(t_vals, T_func(t_vals), color='r', label='T(t)')
plt.xlabel('Time (minutes)')
plt.ylabel('Temperature ( C )')
plt.title("Newton's Law of Cooling")
plt.grid(True)
plt.legend()
plt.show()

# Compute temperature at t = 30 minutes
T_30 = T_func(30)
print("Temperature at t = 30 minutes:", T_30, " C ")
```

Output:



Lab 8: Finding rank, reduced echelon form, solving system of linear equations using Gauss elimination method

Aim:

1. To write a Python program to determine the **rank of a given matrix** .
2. To write a Python program to compute the **Reduced Row Echelon Form (RREF)** of a given matrix .
3. To write a Python program to **solve a system of linear equations** using the **Gauss Elimination method**.

8.1: Write a Python program to determine the **rank of the given matrix**:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 1 \end{bmatrix}$$

Code:

```
import sympy as sp

# Input matrix
A = sp.Matrix([[1, 2, 3],
               [2, 4, 6],
               [1, 1, 1]])

# Compute rank
rank_A = A.rank()

print("Matrix A:")
sp.pprint(A)
print("\nRank of A:", rank_A)
```

Output:

```
Matrix A:
[1  2  3]
[2  4  6]
[1  1  1]

Rank of A: 2
```

8.2: Write a Python program to compute the **Reduced Row Echelon Form (RREF)** of the following matrix:

$$A = \begin{bmatrix} 1 & 2 & -1 & -4 \\ 2 & 3 & -1 & -11 \\ -2 & 0 & -3 & 22 \end{bmatrix}$$

Code:

```python
import sympy as sp

# Input matrix
A = sp.Matrix([[1, 2, -1, -4],
               [2, 3, -1, -11],
               [-2, 0, -3, 22]])

# Compute RREF
rref_A, pivots = A.rref()

print("Matrix A:")
sp.pprint(A)
print("\nRREF of A:")
sp.pprint(rref_A)
print("\nPivot Columns:", pivots)
```

Output:

```
Matrix A:
[ 1   2  -1  -4]
[ 2   3  -1 -11]
[-2   0  -3  22]

RREF of A:
[1  0  0  8]
[0  1  0  1]
[0  0  1  -2]

Pivot Columns: (0, 1, 2)
```

8.3: Solve System using Gauss Elimination

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

Code:

```python
import numpy as np

# Coefficient matrix
A = np.array([[2, 1, -1],
              [-3, -1, 2],
              [-2, 1, 2]], dtype=float)

# Constants vector
b = np.array([8, -11, -3], dtype=float)

n = len(b)

print("Original system:")
print("A =")
```

```
print(A)
print("b =", b)
print()

# Forward elimination
for i in range(n):
    print(f"Step {i+1}: Making diagonal element at position ({i},{i}) = 1")

    # Store the pivot value before modifying the row
    pivot = A[i, i]

    # Make the diagonal element 1
    A[i] = A[i] / pivot
    b[i] = b[i] / pivot

    print(f"After making pivot 1:")
    print("A =")
    print(A)
    print("b =", b)

    # Eliminate below
    for j in range(i+1, n):
        factor = A[j, i]
        A[j] = A[j] - factor * A[i]
        b[j] = b[j] - factor * b[i]

    print(f"After elimination below row {i}:")
    print("A =")
    print(A)
    print("b =", b)
    print()

# Back substitution
x = np.zeros(n)
for i in range(n-1, -1, -1):
    x[i] = b[i] - np.dot(A[i, i+1:], x[i+1:])
    print(f"x[{i}] = {x[i]}")

print("\nFinal solution:")
print("x =", x)

# Verification
print("\nVerification (A*x should equal b):")
print("A*x =", np.dot(A, x))
print("Original b =", b)
```

Output

```
Original system:
```

```
A =
[[ 2.  1. -1.]
 [-3. -1.  2.]
 [-2.  1.  2.]]
b = [  8. -11.  -3.]

Step 1: Making diagonal element at position (0,0) = 1
After making pivot 1:
A =
[[ 1.   0.5 -0.5]
 [-3.  -1.   2. ]
 [-2.   1.   2. ]]
b = [  4. -11.  -3.]
After elimination below row 0:
A =
[[ 1.   0.5 -0.5]
 [ 0.   0.5  0.5]
 [ 0.   2.   1. ]]
b = [4. 1. 5.]

Step 2: Making diagonal element at position (1,1) = 1
After making pivot 1:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [ 0.   2.   1. ]]
b = [4. 2. 5.]
After elimination below row 1:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [ 0.   0.  -1. ]]
b = [4. 2. 1.]

Step 3: Making diagonal element at position (2,2) = 1
After making pivot 1:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [-0.  -0.   1. ]]
b = [ 4.  2. -1.]
After elimination below row 2:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [-0.  -0.   1. ]]
b = [ 4.  2. -1.]
```

```
x[2] = -1.0
x[1] = 3.0
x[0] = 2.0

Final solution:
x = [ 2.  3. -1.]

Verification (A*x should equal b):
A*x = [ 4.  2. -1.]
Original b = [ 4.  2. -1.]
```

Lab 9: Solving system of linear equations using Gauss-Seidel method

Aim: To write a Python program to **solve a system of linear equations** by Gauss-Seidel method

9.1: Solve the following system of linear equations using the **Gauss-Seidel Iteration Method**:

$$\begin{aligned} 20x + y - 2z &= 17 \\ 3x + 20y - z &= -18 \\ 2x - 3y + 20z &= 25 \end{aligned}$$

Code:

```
# Defining equations in diagonally dominant form
f1 = lambda x, y, z: (17 - y + 2*z) / 20
f2 = lambda x, y, z: (-18 - 3*x + z) / 20
f3 = lambda x, y, z: (25 - 2*x + 3*y) / 20

# Initial guesses
x0 = 0
y0 = 0
z0 = 0

# Read tolerable error
e = float(input('Enter tolerable error: '))

print('\nCount \tx\t\ty\t\tz\n')
count = 1
condition = True

# Iterative process
while condition:
    x1 = f1(x0, y0, z0)
    y1 = f2(x1, y0, z0)
    z1 = f3(x1, y1, z0)

    print('%d\t%0.4f\t%0.4f\t%0.4f' % (count, x1, y1, z1))

    e1 = abs(x0 - x1)
    e2 = abs(y0 - y1)
    e3 = abs(z0 - z1)

    x0 = x1
    y0 = y1
    z0 = z1

    count += 1
    condition = e1 > e or e2 > e or e3 > e
```

```
36
37 print('\nSolution: x = %0.3f, y = %0.3f, z = %0.3f\n' % (x1, y1,
       z1))
```

Output:

```
Enter tolerable error: 0.01

Count  x  y  z

1 0.8500 -1.0275 1.0109
2 1.0025 -0.9998 0.9998
3 1.0000 -1.0000 1.0000

Solution: x = 1.000, y = -1.000, z = 1.000
```

9.2: Solve the following system of linear equations using the **Gauss-Seidel Iteration Method**:

$$\begin{aligned} 3x - y + 2z &= 1 \\ x + 2y - z &= 3 \\ 2x - 2y + 6z &= 2 \end{aligned}$$

Code:

```
1  # Defining equations in diagonally dominant form
2  f1 = lambda x, y, z: (1 + y - 2*z) / 3
3  f2 = lambda x, y, z: (3 - x + z) / 2
4  f3 = lambda x, y, z: (2 - 2*x + 2*y) / 6
5
6  # Initial setup
7  x0, y0, z0 = 0, 0, 0
8
9  # Reading tolerable error
10 e = float(input('Enter tolerable error: '))
11
12 print('\nIteration\t x\t\t y\t\t z\n')
13
14 for i in range(1, 25):
15     x1 = f1(x0, y0, z0)
16     y1 = f2(x1, y0, z0)
17     z1 = f3(x1, y1, z0)
18
19     # Print iteration values
20     print('%d\t\t%0.4f\t%0.4f\t%0.4f' % (i, x1, y1, z1))
21
22     # Check error
23     e1 = abs(x1 - x0)
24     e2 = abs(y1 - y0)
25     e3 = abs(z1 - z0)
26
```

```
27      x0, y0, z0 = x1, y1, z1
28
29      if e1 <= e and e2 <= e and e3 <= e:
30          break
31
32  print('\nSolution: x = %0.3f, y = %0.3f, z = %0.3f\n' % (x1, y1,
    z1))
```

Output

```
Enter tolerable error: 0.01

Iteration  x   y   z

1  0.3333 1.3333 0.6667
2  0.3333 1.6667 0.7778
3  0.3704 1.7037 0.7778
4  0.3827 1.6975 0.7716
5  0.3848 1.6934 0.7695

Solution: x = 0.385, y = 1.693, z = 0.770
```

Lab 10: Determine Eigenvalues and Eigenvectors

Aim: To write a Python program to determine the eigenvalues and eigenvectors of a given square matrix using `numpy.linalg.eig()`.

10.1: Determine the eigenvalues and eigenvectors of the matrix

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}.$$

Code:

```
import numpy as np

A = np.array([[ 2.,  1., -1.],
              [-3., -1.,  2.],
              [-2.,  1.,  2.]])

# Compute eigenvalues and right eigenvectors
w, v = np.linalg.eig(A)

print("Eigenvalues:\n", w)
print("\nEigenvectors (columns):\n", v)
```

Output

```
Eigenvalues:
 [ 3.21431974  0.46081068 -0.67513042]

Eigenvectors (columns):
 [[-0.22113631  0.58085679  0.4669257 ]
 [ 0.54213894 -0.0844347  -0.65511165]
 [ 0.81066862  0.80961414  0.59397557]]
```