SAPTHAGIRI NPS UNIVERSITY

Department of Mathematics

School of Engineering and Technology

Lab Manual

for

Linear Algebra and Calculus using Python Programming

24BTPHY102: LINEAR ALGEBRA USING PYTHON PROGRAMMING

LIST OF PROGRAMS:

Lab 1: Find the rank of a matrix and solution of linear equations by Gauss Seidel method.

Lab 2: Compute eigenvalues and Eigenvectors, and find the Dominant Eigenvalues and corresponding Eigenvector by Rayleigh power method.

Lab 3: 2-D plots of cartesian and polar curves.

Lab 4: Finding angle between two polar curves.

Lab 5: Find the partial derivatives and Jacobians.

Lab 6: Evaluation of improper integrals.

Lab 7: Solution of First-order Linear Differential Equation and Exact Differential Equation

Lab 8: Solution of second-order differential Equation

PYTHON PROGRAMMING LANGUAGE

Introduction: Python is a high-level, general-purpose programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It has become one of the most popular programming languages, widely used for web development, data science, artificial intelligence, automation, scripting, and more.

Key features of Python include:

1. **Readability:** Python's syntax is designed to be clear and readable, making it easy for developers to write and maintain code.

2. **Versatility:** Python can be used for various applications, from simple scripts to complex software development.

3. **Large Standard Library:** Python comes with a comprehensive standard library that provides modules and packages for various tasks, reducing the need for external dependencies.

4. **Community Support:** Python has a large and active community of developers, which contributes to the availability of numerous third-party libraries and frameworks.

5. **Interpreted Language:** Python is an interpreted language, meaning that code can be executed directly without compilation. This makes development and debugging more straightforward.

6. **Cross-platform:** Python is compatible with various operating systems, including Windows, macOS, and Linux, making it a cross-platform language.

Anaconda Navigator is a package manager, environment manager and python distribution with a collection of 1500 + open-source package with free community support.

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text.

It supports various programming languages, with Python being one of the most widely used.

Basics of Python:

- To run the program → Shift + Enter
- Addition → +
- Subtraction → -
- Multiplication → *
- Division → / (Quotient as answer)
- Floor division → // (Quotient value without decimal as answer)
- Modulus → % (Remainder as answer)
- Absolute value → abs() (Returns the absolute value of a number)
- Round value → round() (function rounds a number to a specified number of decimal places)
- Power value → ** or pow(base value, power value)
- math.sqrt(x) →Returns the square root of x.
- math.exp(x) →Returns the e raised to the power x.
- math.log(x, base) → Returns the logarithm of x to the given base. If the base is not specified, it returns the natural logarithm.
- math.log10(x) →Returns the base-10 logarithm of x.
- math.factorial(x) → Returns the factorial of x.
- math.gcd(a, b) →Returns the greatest common divisor of a and b.

Solved examples:

```
2+7
```

9

```
9-2
```

7

```
7*7
```

49

```
7/2
```

3.5

```
7//2
```
3

```
7%2
```
1

```
abs(-7)
```
7

```
round(3.6)
```
4

```
pow(2,3)
```
8
(or)

```
2**3
```
8

```
import math
math.sqrt(25)
```
5.0

```
import math
math.exp(2)
```
7.3890560

```
import math
math.log(1)
```
0.0
(or)

```
import math
math.log(4,2)
```
2

```
import math
math.factorial(3)
```
6

```
import math
math.gcd(2,3)
```
1

Syntax for creating the matrix:

Syntax:

```
import numpy as np

# Creating a 3x3 matrix
matrix = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

print(matrix)
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Trigonometric Functions:

Syntax:

```
math.sin(x), math.cos(x), math.tan(x):
```
Returns the sine, cosine, and tangent of x, respectively.

Examples:

```
math.sin(0)
```
0

Differentiation:

Syntax:

```
from sympy import*
x,y=symbols('x,y')
function=diff(function, independent variable)
display(function)
```

Example:

Differentiate $f = x^2 + 2x$ w.r.t. x

Code:

```
from sympy import*
x,y=symbols('x,y')
function=diff(x**2+2*x, x)
display(function)
```

2x+2

Differential equations:

Syntax:

```
from sympy import*
init_printing()
x,y=symbols(`x,y')
y=Function(``y'')(x)
y1=Derivative(y,x)z1=dsolve(Eq(),y)
display(z1)
```

Example: Solve $\frac{dy}{dx} + y\tan x - y^3 \sec x = 0$

Code:

```
from sympy import*
init_printing()
x,y=symbols('x,y')
y=Function("y")(x)
y1=Derivative(y,x)
z1=dsolve(Eq(y1+y*tan(x)-y**3*sec(x),0),y)
display(z1)
```

Output: $y(x) = -\sqrt{\frac{1}{C_1 - 2\sin x}\cos x}$ or $y(x) = \sqrt{\frac{1}{C_1 - 2\sin x}\cos x}$

Integrals:

Syntax:

```
from sympy import*
x,y=symbols('x,y')
function=integrate(function, independent variable)
display(function)
```

Example: Integrate $f = x^2 + 2x$

Code:

```
from sympy import*
x,y=symbols('x,y')
f=x**2+3*x
W1=integrate(f,x)
display(W1)
```

Output: $\frac{x^3}{3} + \frac{3x^2}{2}$

Double and triple integral integration:

Evaluate the integral $\int_0^1 \int_0^x (x^2 + y^2) dydx$

Code:

```
from sympy import*
x,y,z=symbols('x,y,z')
f=x**2+y**2
W1=integrate(f, (y,0,x), (x,0,1))
display(W1)
```

Output**:** 1/3

Built-in commands for plotting the graphs in Python:

In Python, several libraries provide functionalities for plotting graphs and visualizations. Two popular ones are Matplotlib and Seaborn. Below are some built-in commands for plotting graphs using these libraries:

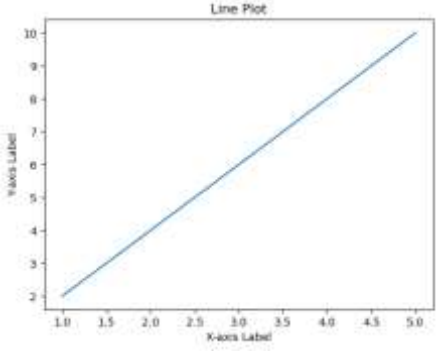
Matplotlib:

Line Plot:

Code:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Line Plot')
plt.show()
```

Output:



How to use if conditions?

1. if statement — for implementing one-way branching
2. if..else statements —for implementing two-way branching

```
#Syntax:
if condition:
statements
```

```
# Check if the given number is positive
a=int(input("Enter an integer: "))
```

```
if a>0:
  print("Entered value is positive")
```
Enter an integer: 5
Entered value is positive

```
# Synatx:
# if condition:
#statements 1
# else:
# statements 2
# If condition is True- statements 1 will be executed
# otherwise- statements 2 will be executed
a=int(input("Enter an integer: "))
if a>0:
    print("Number entered is positive")
else:
     print("Number entered is negative")
```
Enter an integer: -5
Number entered is negative

While loop

- Is used to execute a block of statements repeatedly until a given condition is satisfied.
- When the condition is false, immediately after the loop in the program is executed
- Syntax:

```
while expression:
 statement(s)
```

```
count = 1
while count <= 5:
    print(count)
    count += 1
```
1
2
3
4

```
# Print multiplication table
n=int(input("Enter the number: "))
i=1
while(i<11):
    print(n,'x',i,'=',n*i)
    i=i+1
```

Enter the number: 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

For loop

- are used for sequential traversal
- it falls under the category of definite iteration
- also used to access elements from a container (for example list, string,tuple)using built-in function range()
- Syntax:

```
for variable_name in sequence :
statement_1
statement_2
 ....
```

The range() function

Syntax:

- range(a): Generates a sequence of numbers from 0 to a, excluding a, incrementing by 1.

- range(a,b): Generates a sequence of numbers from a to b excluding b, incrementing by 1.
- range(a,b,c): Generates a sequence of numbers from a to b excluding b, incrementing by c.

```
#Print numbers from 101 to 130 with a step length 2
excluding 130.
for i in range(101,130,2):
    print(i)
```

```
101
103
105
107
109
111
113
115
117
119
121
123
125
127
129
```

```
# Sum of first n natural numbers
sum=0
n=int(input("Enter n: "))
for i in range(1,n+1): # i=1, sum=1; i=2, sum=3; ....
    sum=sum+i
    print("Sum of first ",n,"natural numbers = ",sum)
```

```
Enter n:  2
Sum of first  2 natural numbers =  1
Sum of first  2 natural numbers =  3
```

```
# Multiplication table
n=int(input("Enter the number"))
for i in range(1,11):
    print(n,'x',i,'=',n*i)
```

Enter the number 2
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

Lab1: Find the rank of a matrix and solution of linear equations by Gauss seidel method

Use python

1. To find Rank of the matrix
2. To check whether the given system is diagonally dominant or not.
3. To find the solution if the system is diagonally dominant.

Rank of the matrix:

1. Find the rank of the matrix $A = \begin{bmatrix} 2 & -1 & -3 & -1 \\ 1 & 2 & 3 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$

Code:

```
#Rank of a matrix
import numpy as np
from scipy . linalg import null_space
# Define a linear transformation interms of matrix
A = np . array ([[2, -1, -3, -1], [1 , 2 , 3, -1], [1 , 0 ,
1, 1], [0, 1, 1, -1]])
# Find the rank of the matrix A
rank = np . linalg . matrix_rank ( A )
print (" Rank of the matrix ", rank )
```

Output:

Rank of the matrix 3

2. Find the rank of the matrix $A = \begin{bmatrix} 0 & 1 & -3 & -1 \\ 1 & 0 & 1 & 1 \\ 3 & 1 & 0 & 2 \\ 1 & 1 & -2 & 0 \end{bmatrix}$

Code:

```
#Rank of a matrix
import numpy as np
from scipy . linalg import null_space
# Define a linear transformation interms of matrix
A = np . array ([[0, 1, -3, -1], [1 , 0 , 1, 1], [3 , 1 ,
0, 2], [1, 1, -2, 0]])
# Find the rank of the matrix A
```

```
rank = np . linalg . matrix_rank ( A )
print (" Rank of the matrix ", rank )
```

Output:
Rank of the matrix 2

Exercise:

3. Find the rank of the matrix $A = \begin{bmatrix} 0 & 2 & 3 & 4 \\ 2 & 3 & 5 & 4 \\ 4 & 8 & 13 & 12 \end{bmatrix}$

4. Find the rank of the matrix $A = \begin{bmatrix} 4 & 0 & 2 & 1 \\ 4 & 1 & 3 & 4 \\ 2 & 3 & 4 & 7 \\ 2 & 3 & 1 & 4 \end{bmatrix}$

Gauss-Seidel method:

Gauss-Seidel method is an iterative method to solve a system of linear equations. The method works if the system is diagonally dominant. That is $|a_{ij}| \geq \sum_{i \neq j} |a_{ij}|$ for all $i's$

5. Solve the system of equations using the Gauss-Seidel method:
   $20x + y - 2z = 17; 3x + 20y - z = -18; 2x - 3y + 20z = 25.$

Code:

```
#Gauss Seidel
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 setup
x0 = 0
y0 = 0
z0 = 0
count = 1
# Reading tolerable error
e = float ( input ('Enter tolerable error : ') )
# Implementation of Gauss Seidel Iteration
print ('\n Count \tx\ty\tz\n')
condition = True
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\n' %( count , x1 ,
y1 , z1 ) )
    e1 = abs ( x0-x1 ) ;
    e2 = abs ( y0-y1 ) ;
    e3 = abs ( z0-z1 ) ;
    count += 1
    x0 = x1
    y0 = y1
    z0 = z1
    condition = e1>e and e2>e and e3>e
print ('\n Solution : x=%0.3f , y=%0.3f and z = %0.3f\
n'% ( x1 , y1 , z1 ) )
```

Enter tolerable error : 0.001

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 and z = 1.000

6. Solve $10x + y + z = 12; x + 10y + z = 12; x + y + 10z = 12$ by Gauss -Seidel Iteration method.

Code:

```
# Gauss Seidel Iteration
f1 = lambda x ,y , z: ( 12-y-z )/10
f2 = lambda x ,y , z: ( 12-x-z )/10
f3 = lambda x ,y , z: ( 12-x-y )/10
x0,y0,z0 = 0,0,0
# Reading tolerable error
e = float ( input ('Enter tolerable error : ') )
print ('\t Iteration \t x\t y\t z\n')
for i in range (0 , 25 ):
```

```
    x1 = f1 ( x0 , y0 , z0 )
    y1 = f2 ( x1 , y0 , z0 )
    z1 = f3 ( x1 , y1 , z0 )
    # Printing the values of x, y, z in ith iteration
    print ('%d\t%0.4f\t%0.4f\t%0.4f\n' %(i , x1 , y1 ,
z1 ) )
    e1 = abs ( x0 - x1 ) ;
    e2 = abs ( y0 - y1 ) ;
    e3 = abs ( z0 - z1 ) ;
    x0 = x1
    y0 = y1
    z0 = z1
    if e1>e and e2>e and e3>e:
        continue
    else :
        break
print ('\n Solution : x=%0.3f , y=%0.3f and z =
%0.3f\n'% ( x1 , y1 , z1 ) )
```

Enter tolerable error : 0.001

	Iteration	x	y	z
0		1.2000	1.0800	0.9720
1		0.9948	1.0033	1.0002
2		0.9996	1.0000	1.0000

Solution : x=1.000 , y=1.000 and z = 1.000

Exercise:

7. Solve the system of equations using the Gauss-Seidel method:
   $10x + 2y + z = 9; x + 10y - z = -22; -2x + 3y + 10z = 22.$
8. Solve the system of equations using the Gauss-Seidel method:
   $5x + 2y + z = 12; x + 4y + 2z = 15; x + 2y + 5z = 20.$

Lab 2: Compute Eigen values and Eigen vectors, find the Dominant Eigen values and corresponding Eigen vector by Rayleigh power method.

Use python

1. To find dominant and corresponding eigenvector by Rayleigh power method.

Syntax for the commands used:

- np.linalg.eig(A): Compute the eigenvalues and right eigenvectors of a square array

```
np.linalg.eig(A)
```

- np.linalg.eigvals(A): Computes the eigenvalues of a non-symmetric array.
- np.array(parameter): Creates ndarray
  np.array([[1,2,3]]) is a one-dimensional array
  np.array([[1,2,3,6],[3,4,5,8],[2,5,6,1]]) is a multi-dimensional array

Largest eigenvalue and corresponding eigenvector by Rayleigh method:

For a given Matrix A and initial eigenvector X_0, the power method goes as follows: Consider AX_0 take the largest numbers say λ_1 from the column vector, and write $AX = \lambda_1 X_1$. At this stage, λ_1 is the approximate eigenvalue and X_1 will be the corresponding eigenvector. Now multiply the Matrix A with X_1 and continue the iteration. This method is going to give the dominant eigenvalue of the Matrix

1. Find the largest eigenvalues and the corresponding eigenvectors of matrix $\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{bmatrix}$ by Rayleigh's Power Method

Code:

```
#eigenvector by Rayleigh method
import numpy as np
def normalize ( x ):
```

```
    fac = abs( x ) .max ()
    x_n = x / x .max ()
    return fac , x_n
x = np . array ([1 , 0 , 0])
a = np . array ([[2, 0, 1 ],[0 ,2 , 0],[1 ,0 , 2]])
for i in range ( 10 ):
    x = np .dot(a , x )
    lambda_1 , x = normalize ( x )
print (' Eigenvalue :', lambda_1 )
print (' Eigenvector :', x )
```

Output:

Eigenvalue : 2.9998983946352364

Eigenvector : [1. 0. 0.99996613]

2. Find the largest eigenvalues and the corresponding eigenvectors of matrix $\begin{bmatrix} 4 & 1 & -1 \\ 2 & 3 & -1 \\ -2 & 1 & 5 \end{bmatrix}$ by taking the initial eigen vector as $[1, 0.8, -0.8]^T$

Code:

```
#eigenvector by Rayleigh method
import numpy as np
def normalize ( x ):
    fac = abs( x ) .max ()
    x_n = x / x .max ()
    return fac , x_n
x = np . array ([1, 0.8 , -0.8])
a = np . array ([[4 ,1 , -1 ],[2 ,3, -1],[-2 ,1 , 5]])
for i in range ( 10 ):
    x = np .dot(a , x )
    lambda_1 , x = normalize ( x )
print (' Eigenvalue :', lambda_1 )
print (' Eigenvector :', x )
```

Output:

Eigenvalue : 5.999977420010387

Eigenvector : [1. 0.99999624 -0.99999624]

Exercise:

3. Find the largest eigenvalues and the corresponding eigenvectors of matrix $\begin{bmatrix} 10 & 2 & 1 \\ 2 & 10 & 1 \\ 2 & 1 & 10 \end{bmatrix}$ by taking the initial eigen vector as $[0,\ 0,\ 1]^T$
4. Find the largest eigenvalues and the corresponding eigenvectors of matrix $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ by taking the initial eigen vector as $[1,\ 1,\ 1]^T$.

Lab 3: 2-D plots of cartesian and polar curves.

Use python

1. to plot Cartesian curves.
2. to plot polar curves.

Syntax for the commands used:

- Plot y versus x as lines and or markers using default line style, color, and other customizations.

```
plot(x, y, color='green', marker='o',
linestyle='dashed',linewidth=2,markersize=12)
```

- A scatter plot of y versus x with varying marker size and/or color.

```
scatter(x_axis_data, y_axis_data, s=None,
c=None, marker=None, cmap=None,vmin=None,
vmax=None,alpha=None,
linewidths=None,edgecolors=None)
```

- Return num evenly spaced numbers over a specified interval [start, stop]. The endpoint of the interval can optionally be excluded

```
numpy.linspace(start, stop, num=50, endpoint=True,
retstep=False,dtype=None, axis=0)
```

- Return evenly spaced values within a given interval. arange can be called with a varying number of positional arguments

```
numpy.arange([start, ]stop, [step,]dtype=None,
*, like=None)
```

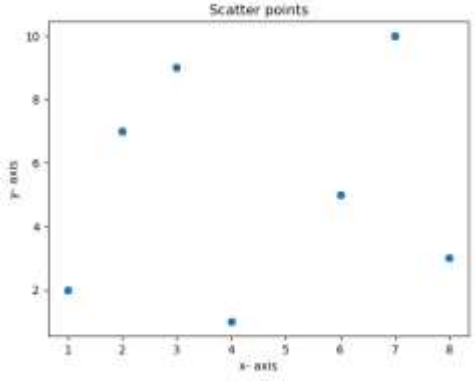
1. Plotting points (Scattered plot)

Code:

```
# importing the required module
 import matplotlib.pyplot as plt
 x = [1,2,3,4,6,7,8] # x axis values
```

```
y = [2,7,9,1,5,10,3] # corresponding y axis values
plt.scatter(x, y) # plotting the points
plt.xlabel('x- axis') # naming the x axis
plt.ylabel('y- axis') # naming the y axis
plt.title('Scatter points') # giving a title to my graph
plt.show() # function to show the plot
```

Output:

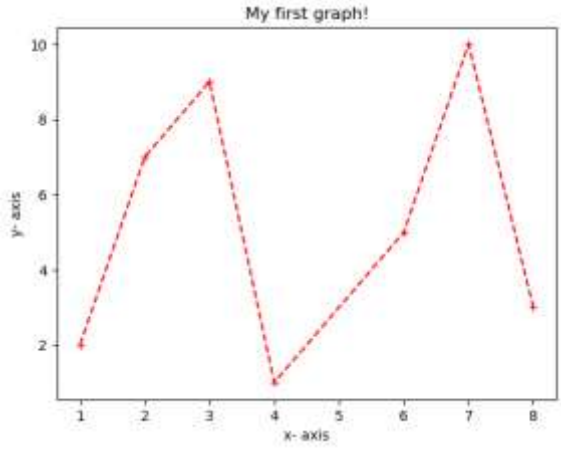


2. Plotting a line (Line plot)

Code:

```
# importing the required module
import matplotlib.pyplot as plt
x = [1,2,3,4,6,7,8] # x axis values
y = [2,7,9,1,5,10,3] # corresponding y axis values
plt.plot(x, y, 'r+--') # plotting the points
plt.xlabel('x- axis') # naming the x axis
plt.ylabel('y- axis') # naming the y axis
plt.title('My first graph!') # giving a title to my graph
plt.show() # function to show the plot
```

Output:



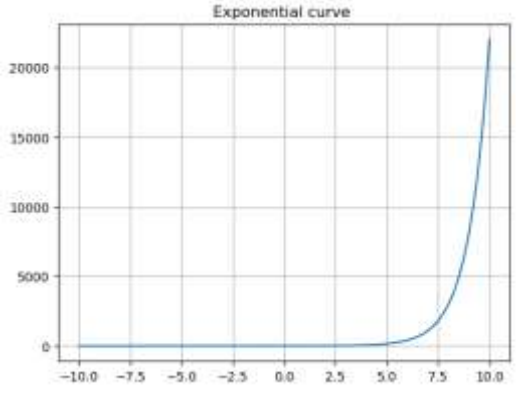
Functions

3. Exponential curve, $y = e^x$

Code:

```
 # importing the required modules
 import numpy as np
 import matplotlib.pyplot as plt
 x = np.arange(-10, 10, 0.001)
 y = np.exp(x) # Exponential function
 plt.plot(x,y) # plotting the points
 plt.title("Exponential curve ") # giving a title to the
graph
 plt.grid() # displaying the grid
 plt.show() # shows the plot
```

Output:

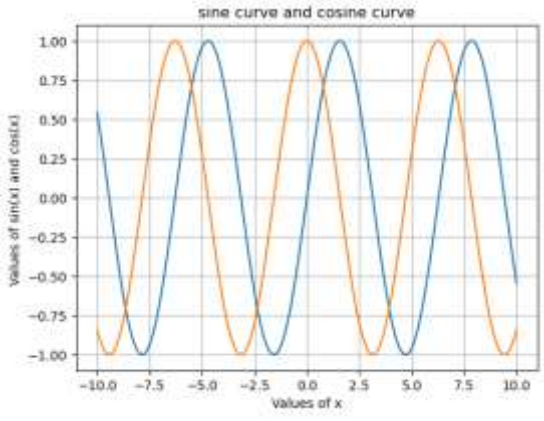


4. Sine and Cosine curves

Code:

```
import numpy as np
 import matplotlib.pyplot as plt
 x = np.arange(-10, 10, 0.001)
 y1 = np.sin(x)
 y2=np.cos(x)
 plt.plot(x,y1,x,y2)
 plt.title("sine curve and cosine curve")
 plt.xlabel("Values of x")
 plt.ylabel("Values of sin(x) and cos(x) ")
 plt.grid()
 plt.show()
```

Output:

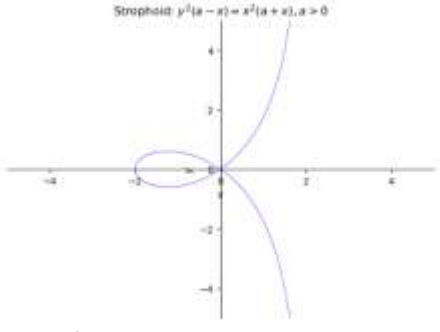


Implicit Function:

5. Strophoid: $y^2(a-x) = x^2(a+x), a > 0$

Code:

```
from sympy import *
x, y = symbols('x y')
p3= plot_implicit(Eq((y**2)*(2-x), (x**2)*(2+x)), (x,-5, 5), (y,-5, 5),
                  title= 'Strophoid: $y^2 (a-x)=x^2 (a+x), a> 0$') # a=2
```



Exercise:

6. Plot the graph Circle:$x^2 + y^2 = 5$
7. Plot the graph Lemniscate: $a^2y^2 = x^2(a^2 - x^2)$

Polar Curves

The matplotlib.pyplot.polar() function in the pyplot module of matplotlib python library is used to plot the curves in polar coordinates.

Syntax:

```
matplotlib.pyplot.polar(theta, r, **kwargs)
```

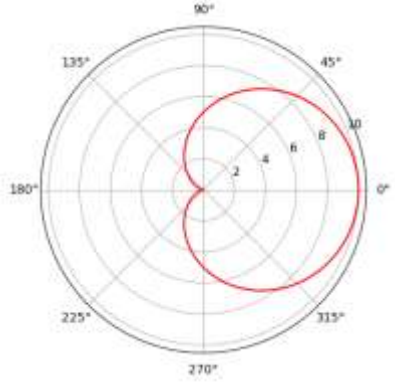
- Theta: This is the angle at which we want to draw the curve.
- r: It is the distance.

8. Cardioid: $r = 5(1 + cos\theta)$

Code:

```
#Plot cardioid r=5(1+cos theta)
from pylab import *
theta=linspace(0,2*np.pi,1000)
r1=5+5*cos(theta)
polar(theta,r1,'r')
show()
```

Output:



Exercise:

9. Four leaved Rose: $r = 2|\cos(2x)|$
10. Limacon: $r = a + bcos\theta$

Lab 4: Finding angle between two polar curves.

Use python

1. To find angle between two polar curves.

Syntax for the commands used:

- diff()

```
diff(function,variable)
```

- solve()
  This line solves the equation, finding the points of intersection between the two curves.

- syntax of Substitute: subs()

Angle between two polar curves:

Angle between radius vector and tangent is given by $tan\varphi = r\frac{d\theta}{dr}$

If $tan\varphi_1$ and $tan\varphi_1$ are angle between radius vector and tangent of two curves then $|\varphi_1 - \varphi_2|$ is the angle between two curves at the point of intersection.

1. Find the angle between the curves $r = 4(1 + \cos t)$ and $r = 5(1 - \cos t)$.

Code:

```
from sympy import *
r,t =symbols('r,t')

r1=4* (1+cos(t));
r2=5* (1-cos(t));
dr1=diff(r1,t)
dr2=diff(r2,t)
t1=r1/dr1
t2=r2/dr2
q=solve(r1-r2,t)
w1=t1.subs({t:float(q[1])})
w2=t2.subs({t:float(q[1])})
```

```
y1=atan(w1)
y2=atan(w2)
w=abs(y1-y2)
print('Angle between curves in radians is %0.3f'%(w))
```

Output

Angle between curves in radians is 1.571

2. Find the angle between the curves $r = 4\cos t$ and $r = 5\sin t$.

   Code:

```
from sympy import *
r,t =symbols('r,t')
r1=4*(cos(t));
r2=5*(sin(t));
dr1=diff(r1,t)
dr2=diff(r2,t)
t1=r1/dr1
t2=r2/dr2
q=solve(r1-r2,t)
w1=t1.subs({t:float(q[0])})
w2=t2.subs({t:float(q[0])})
y1=atan(w1)
y2=atan(w2)
w=abs(y1-y2)
print('Angle between curves in radians is
%0.4f'%float(w))
```

   Output

   Angle between curves in radians is 1.5708

Exercise:

3. Find the angle between the radius vector and tangent to the following polar curves $r = a\theta$ and $r = \frac{a}{\theta}$
4. Find the angle between the radius vector and tangent to the following polar curves $r = 2\sin\theta$ and $r = 2\cos\theta$

Lab 5: Find the partial derivatives and Jacobians.

Use python

1. to find partial derivatives of functions of several variables.
2. to find Jacobian of function of two and three variables.

Syntax for the commands used:

- To create a matrix:

```
Matrix([[row1],[row2],[row3]....[rown]])
```

Ex: A 3×3 matrix can be defined as a

```
Matrix([[a11,a12,a13],[a21,a22,a23],[a31, a32 a33]])
```

- Evaluate the determinant of a matrix M:

```
Determinant(M)
det(M)
```

- If a function is of two or more than two independent variables then it differentiates the function partially w.r.t variable.
  If $u = u(x, y)$ then,
  - $\frac{\partial u}{\partial x} = diff(u, x)$
  - $\frac{\partial u}{\partial y} = diff(u, y)$
  - $\frac{\partial^2 u}{\partial x^2} = diff(u, x, x)$
  - $\frac{\partial^2 u}{\partial y^2} = diff(u, y, y)$

1. Prove that Mixed partial derivatives $u_{xy} = u_{yx}$ for $u = e^x(x\cos y - y\sin y)$.

   Code:

```
from sympy import *
x,y =symbols('x y')
u=exp(x)*(x*cos(y)-y*sin(y))
dux=diff(u,x)
```

```
duy=diff(u,y)
duxy=diff(dux,y)
duyx=diff(duy,x)
if duxy==duyx:
    print('Mixed partial derivatives are equal')
else:
    print('Mixed partial derivatives are not equal')
```

Output:

Mixed partial derivatives are equal.

2. Prove that $u = e^x(x\cos y - y\sin y)$ then $u_{xx} + u_{yy} = 0$.

Code:

```
from sympy import *
x,y =symbols('x y')
u=exp(x)*(x*cos(y)-y*sin(y))
display(u)
dux=diff(u,x)
duy=diff(u,y)
uxx=diff(dux,x)
uyy=diff(duy,y)
w=uxx+uyy
w1=simplify(w)
print('Ans:',float(w1))
```

Output:

$e^x(x\cos(y) - y\sin(y))$, Ans 0.

3. If $u = \frac{xy}{z}, v = \frac{yz}{x}, w = \frac{zx}{y}$ then prove that $J = 4$.

Code:

```
from sympy import *
x,y,z=symbols('x,y,z')
u=x*y/z
v=y*z/x
w=z*x/y
```

```
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)
J=Matrix([[dux,duy,duz],[dvx,dvy,dvz],[dwx,dwy,dwz]])
;

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

# Find the determinat of Jacobian Matrix
Jac=det(J).doit()
print('\n\n J = ', Jac)
```

Output:

The Jacobian matrix is

$$\begin{matrix} \frac{y}{z} & \frac{x}{z} & \frac{-xy}{z^2} \\ \frac{-y^2}{x^2} & \frac{z}{x} & \frac{y}{x} \\ \frac{z}{y} & \frac{-xz}{y^2} & \frac{x}{y} \end{matrix}$$

J=4

4. $X = \rho * \cos(\phi) * \sin(\theta),\ Y = \rho * \cos(\phi) * \cos(\theta),\ Z = \rho * \sin(\phi)$ then find $\frac{\partial(X,Y,Z)}{\partial(\rho,\phi,\theta)}$.

   Code:

```
from sympy import *
from sympy.abc import rho, phi, theta

X=rho*cos(phi)*sin(theta);
Y=rho*cos(phi)*cos(theta);
Z=rho*sin(phi);

dx=Derivative(X,rho).doit()
dy=Derivative(Y,rho).doit()
dz=Derivative(Z,rho).doit()
dx1=Derivative(X,phi).doit();
dy1=Derivative(Y,phi).doit();
dz1=Derivative(Z,phi).doit()
dx2=Derivative(X,theta).doit()
dy2=Derivative(Y,theta).doit();
dz2=Derivative(Z,theta).doit();

J=Matrix([[dx,dy,dz],[dx1,dy1,dz1],[dx2,dy2,dz2]]);
print('The Jacobian matrix is ')
display(J)

print('\n\n J = \n')
display(simplify(Determinant(J).doit()))
```

Output:

The Jacobian matrix is

$$\begin{matrix} \sin(\theta)\cos(\varphi) & \cos(\varphi)\cos(\theta) & \sin(\varphi) \\ -\rho\sin(\varphi)\sin(\theta) & -\rho\sin(\varphi)\cos(\theta) & \rho\cos(\varphi) \\ \rho\cos(\varphi)\cos(\theta) & -\rho\sin(\theta)\cos(\varphi) & 0 \end{matrix}$$

$$J = \rho^2\cos(\varphi)$$

Exercise:

1. If $u = tan^{-1}\left(\frac{y}{x}\right)$. Verify that $\frac{\partial^2 u}{\partial y\,\partial x} = \frac{\partial^2 u}{\partial x\,\partial y}$.
2. If $u = \log\left(\frac{x^2+y^2}{x+y}\right)$. Show that $xu_x + yu_y = 1$.

3. If $x = u - v, y = v - uvw$ and $z = uvw$ find Jacobian of x, y, z $w.r.t\, u, v, w$.
4. If $u = x + 3y^2 - z^3, v = 4x^2yz$ and $w = 2z^2 - xy$ find $\frac{\partial(u,v,w)}{\partial(x,y,z)}$ $at\ (-2,-2,1)$.

Lab 6: Evaluation of improper integrals

Use python

1. to evaluate improper integrals

Syntax for the commands used:

- Data pretty printer in Python:
  `pprint()`

- Integrate:
  `integrate(function,(variable, min_limit, max_limit))`

Note: We can evaluate improper integral involving infinity by using inf.

1. Evaluate $\int_0^{\infty} e^{-x}\, dx$.

   Code:

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x),(x,0,float('inf')))
print(simplify(w1))
```

   Output:

   1

2. Evaluate $\int_0^{\frac{\pi}{2}} \sin^7(x)\, dx$.

   Code:

```
from sympy import *
x=symbols('x')
w1=integrate(sin(x)**7,(x,0,float(pi/2)))
```

```
print(simplify(w1))
```

Output:

0.457142857142857

3. Evaluate $\int_0^{\frac{\pi}{2}} \sin^9(x)\, dx$.

Code:

```
from sympy import *
x=symbols('x')
w1=integrate(sin(x)**9,(x,0,float(pi/2)))
print(simplify(w1))
```

Output:

0.406349206349206

4. Evaluate $\int_0^{\frac{\pi}{2}} \sin^6(x)\cos^4(x)\ \ dx$.

Code:

```
from sympy import *
x=symbols('x')
w1=integrate(sin(x)**6 *
cos(x)**4,(x,0,float(pi/2)))
print(simplify(w1))
```

Output:

0.0184077694546277

Exercise:

5. Evaluate $\int_0^{\infty} \frac{x^2}{(1+x^2)^2} dx$.
6. Evaluate $\int_0^{2\pi} \cos^4(x)\, dx$.
7. Evaluate $\int_0^{\frac{\pi}{2}} \cos^5(4x)\, dx$.

8. Evaluate $\int_0^{\pi} \sin^8(x)\cos(x)\ \ dx$.

Lab 7: Solution of First-order Linear Differential Equation and Exact Differential Equation

Use python

1. to find solution to the linear differential equation
2. to find the solution of the exact differential equation

Linear Differential Equation

The syntax for the commands used:

- **sympy**: This imports all functions and classes from the `sympy` library. This is generally not recommended for larger projects due to namespace pollution, but it can be convenient for small scripts or interactive sessions.
- **Define the symbols**: The symbols function creates symbolic variables `x` and `y`. These symbols can be used to define mathematical expressions and equations.
- **Function("y")(x):** Here, Function ("y") creates a symbolic function y, and (x) indicates that y is a function of the variable x. Now, y represents $y(x)$.
- **Eq(…., 0):** This creates a symbolic equation
- **dsolve(..., y):** This function attempts to solve the differential equation for y.
- **display():** This function displays the result nicely formatted**.**

1. Solve $\frac{dy}{dx} + \frac{2}{x}y = x^3$

Code:

```
from sympy import *

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

# Define the differential equation
diff_eq = Eq(y.diff(x) + (2/x)*y, x**3)

# Display the differential equation
```

```
print("Differential Equation:")
display(diff_eq)

# Solve the differential equation
solution = dsolve(diff_eq, y)

# Display the general solution
print("General Solution:")
display(solution)
```

Output:

Differential Equation:

$$\frac{d}{dx}y(x) + \frac{2y(x)}{x} = x^3$$

General Solution:

$$y(x) = \frac{C_1 + \frac{x^6}{6}}{x^2}$$

2. Solve $\frac{dy}{dx} + \frac{2}{x}y = xlogx$

Code:

```
from sympy import *

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

# Define the differential equation
diff_eq = Eq(y.diff(x) + (2/x)*y, x*log(x))

# Display the differential equation
print("Differential Equation:")
display(diff_eq)

# Solve the differential equation
solution = dsolve(diff_eq, y)
```

```
# Display the general solution
print("General Solution:")
display(solution)
```

Output:

Differential Equation:

$$\frac{dy}{dx}y(x) + \frac{2y(x)}{x} = xlogx$$

General Solution:

$$y(x) = \frac{c_1 + x^4(4\log(x) - 1)}{x^2}$$

Exercise:

1. Solve $sin^2x\frac{dy}{dx} - y = cotx$
2. Solve $\frac{dy}{dx} + ysecx = tanx$

Exact Differential Equation:

1. Solve $(2xy + 3y)dx + (x^2 + 3x)dy = 0$

Code:

```
from sympy import *

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

# Define M and N
M = 2*x*y + 3*y
N = x**2 + 3*x

# Check for exactness
exact_condition = diff(M, y) == diff(N, x)

if exact_condition:
    print("The given equation is exact.")
```

```
    # Find the potential function
    f_x = integrate(M, x) + integrate(N - diff(integrate(M,
x), y), y)

    print("The potential function is:", f_x)

    # Find the solution by setting f(x, y) equal to a
constant
    C = symbols('C')
    solution = Eq(f_x, C)

    print("The solution is:", solution)

else:
    print("The given equation is not exact.")
```

Output:

```
The given equation is exact.
The potential function is: x**2*y + 3*x*y
The solution is: Eq(x**2*y + 3*x*y, C)
```

2. Solve $(e^y + 1)\cos(x)dx + e^y\sin(x)dy = 0$

Code:

```
from sympy import *

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

# Define M and N
M = (exp(y) + 1) * cos(x)
N = exp(y) * sin(x)

# Check for exactness
exact_condition = diff(M, y) == diff(N, x)

if exact_condition:
    print("The given equation is exact.")
```

```
    # Find the potential function
    f_x = integrate(M, x) + integrate(N - diff(integrate(M,
x), y), y)

    print("The potential function is:", f_x)

    # Find the solution by setting f(x, y) equal to a
constant
    C = symbols('C')
    solution = Eq(f_x, C)

    print("The solution is:", solution)

else:
    print("The given equation is not exact.")
```

Output:

```
The given equation is exact.
The potential function is: (exp(y) + 1)*sin(x)
The solution is: Eq((exp(y) + 1)*sin(x), C)
```

Exercise:

1. Solve $\frac{dy}{dx} = \frac{x^2 - ay}{ax - y^2}$
2. Solve $\left(3x^2y + \frac{y}{x}\right)dx + (x^3 + logx)dy = 0$

Lab 8: Solution of second-order differential Equation

A second-order differential equation is defined as

$\frac{d^2y}{dx^2} + P(x)\frac{dy}{dx} + Q(x)y = f(x)$, where $P(X), Q(x)$, and $f(x)$ are functions of x.

When $f(x) = 0$, the equation is called a homogenous second-order differential equation Otherwise, the second-order differential equation is non-homogenous.

1. Solve $(D^2 - 6D + 1)y = 3e^{-4x}$

Code:

```
from sympy import *
# Define the variables
x = symbols('x')
y = Function('y')(x)

# Define the differential operators
D2 = y.diff(x, 2)
D1 = y.diff(x)

# Define the differential equation
lhs = D2 -6*D1 + 9*y
rhs = 3*exp(-4*x)
diff_eq = Eq(lhs, rhs)

# Solve the differential equation
solution = dsolve(diff_eq, y)

# Display the solution
print("The solution to the differential equation is:")
solution
```

Output:

The solution to the differential equation is:

$$y(x) = (c_1 + c_2 x)e^{3x} + \frac{3e^{-4x}}{49}$$

2. Solve $(D^3 + D^2 + D + 1)y = e^{3x+4}$

Code:

```
from sympy import *

# Define the variables
x = symbols('x')
y = Function('y')(x)

# Define the differential operator
D3 = y.diff(x, 3)
D2 = y.diff(x, 2)
```

```
D1 = y.diff(x)

# Define the differential equation
lhs = D3 + D2 + D1 + y
rhs = exp(3*x + 4)
diff_eq = Eq(lhs, rhs)

# Solve the differential equation
solution = dsolve(diff_eq, y)

# Display the solution
print("The solution to the differential equation is:")
solution
```

Output:

The solution to the differential equation is:

$$y(x) = c_1 e^{-x} + c_2 sin(x) + c_3 cos(x) + \frac{e^{3x} + 4}{40}$$

3. Solve $(D^3 - 1)y = 3\cos 2x$

```
from sympy import *

# Define the variables
x = symbols('x')
y = Function('y')(x)

# Define the differential operator
D3 = y.diff(x, 3)
D2 = y.diff(x, 2)
D1 = y.diff(x)

# Define the differential equation
lhs = D3 + D2 + D1 + y
rhs = exp(3*x + 4)
diff_eq = Eq(lhs, rhs)

# Solve the differential equation
solution = dsolve(diff_eq, y)
```

```
# Display the solution
print("The solution to the differential equation is:")
solution
```

Output:

The solution to the differential equation is:

$$y(x) = C_3 e^x + \left(C_1 sin\left(\frac{\sqrt{3x}}{2}\right) + C_2 cos\left(\frac{\sqrt{3x}}{2}\right)\right) e^{\frac{-x}{2}} - \frac{24\ sin(2x)}{65} - \frac{3\cos(2x)}{65}$$

Exercise:

1. Solve $(4D^2 - 4D + 1)y = 4$
2. Solve $(D^2 - 3D + 2)y = e^{5x}$