

A J Institute of Engineering & Technology

Mangalore - 575006



LAB MANUAL
MASTER COPY

Mathematics for Electrical & Electronics Engineering Streams-I

BMATE101

Prepared By:

Dr.Pavitra Shanbhag

DEPARTMENT OF
ELECTRONICS & COMMUNICATION
ENGINEERING

Vision

To be recognized as a center of excellence in the area of Electronics and Communication Engineering by nurturing the young innovative minds into skillful and ethical professionals to cater the industrial and societal needs.

Mission Statements

M1: To establish state-of-the art laboratories to facilitate research and innovation to upgrade the knowledge and skills in healthcare sector and IoT.

M2: To provide industry interaction for training programs on latest technology

M3: To provide ethical and value based education by promoting activities addressing the societal needs.

PEOs	Description
PEO1	Exhibit a desire for lifelong learning through professional and societal activities.
PEO2	Exhibit and apply their technical skills and knowledge in Electronics and Communication Engineering for industry and societal needs
PEO3	Exhibit leadership qualities, professional skills, management skills and ethics needed for successful career.

S.No.	POs	Description
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
2	Problem Analysis	Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural science and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal and environmental considerations.
4	Conduct investigations of complex problems	Use research based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5	Modern tool usage:	create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations

S.No.	POs	Description
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment sustainability	Understand the impact of the professional engineering solutions in the societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9	Individual and team work	Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.
10	Communication	communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to ones own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12	Lifelong learning	recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broader context of technological change.

PSOs	Description
PSO1	Embedded Systems: Ability to apply the fundamental knowledge of core Electronics and Communication Engineering courses in the analysis, design, development of integrated electronic systems and healthcare devices.
PSO2	Communication Systems: Ability to apply the fundamental knowledge of signal processing in the analysis, design, and development of communication systems.
PSO3	Simulation: Ability to use modern electronic tools such as MATLAB, Xilinx, Multisim etc, to design and analyse the complex electronics and communication systems.

Ist SEMESTER ELECTRONICS AND COMMUNICATION ENGINEERING

Mathematics-I for Electrical & Electronics Engineering Stream

Sub Code	Number of Lab Hours/Week	CIE Marks	RBT Levels	Exam Hours
BMATE101	02	50	L1, L2, L3	02Hrs

COURSE OBJECTIVES:

This course will enable the students to:

1	Familiarize the importance of calculus associated with one variable and multivariable for Electrical and Electronics engineering.
2	Analyze Electrical and Electronics engineering problems by applying Ordinary Differential Equations.
3	Familiarize the important tools in Integral Calculus that are essential in Electrical and Electronics engineering.
4	Develop the knowledge of Linear Algebra to solve the system of equations.

Contents

1. Introduction

I Basics of Python

II Programming Structure

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

Lab 2. Finding Angle between Two Polar Curves, Curvature and Radius of Curvature.

Lab 3. Finding Partial Derivatives and Jacobian.

Lab 4. Taylor Series Expansion and L'Hospital's Rule.

Lab 5. Solution of First Order Differential Equations and Plotting the Solution Curve.

Lab 6. Program to compute area, volume and Centre of gravity.

Lab7. Evaluation of improper integrals.

Lab 8. Numerical Solution of System of Equations, Test for Consistency and Graphical Representation Of the Solution.

Lab 9. Solution of Linear Equations by Gauss-Seidel Method.

Lab 10. Compute Eigen Value and Corresponding Eigen Vectors, Find the Dominant Eigen Value and Corresponding Eigen Vector by Rayleigh Power Method.

Course Outcomes:

At the end of the course the student will be able to:

Course Outcome	Description
CO:1	Apply the knowledge of calculus to solve problems related to polar curves and its applications in determining the bentness of a curve and demonstrate using python.
CO:2	Outline the notion of partial differentiation to calculate rate of change of multivariate functions and solve problems related to composite functions, Jacobian and demonstrate using python.
CO:3	Analyze the solution of linear and nonlinear ordinary differential equations and demonstrate the same using python.
CO:4	Apply the concept of change of order of integration and variables to evaluate multiple integrals and their usage in computing area, volume and demonstrate using python.
CO:5	Solve the system of linear equations using matrix theory and compute eigenvalues and eigenvectors and demonstrate using python.

DO'S:-

1. Maintain silence and discipline.
2. Before entering, leave the footwear outside the lab.
3. Proper dress code has to be maintained while entering in the Lab.
4. Students should carry lab observation book and record book completed in all aspects.
5. Read and understand logic of the program thoroughly before coming to the laboratory.
6. Enter in the login register before switching on the computers.
7. Students should be at their concerned table; unnecessary movement is restricted.
8. Students should maintain same computer until end of the semester.
9. Report any problems in computers to the faculty member/laboratory technician immediately.
10. Note down the experimental results legibly in the Observation book and get the same verified & signed by the Faculty.
11. After completing the experiments, students should switch off the computers, enter logout time and arrange chairs neatly.

DON'Ts:-

1. Do not come late to the Lab.
2. Don't enter the lab without valid ID card.
3. Do not leave the lab without the permission of the Faculty In-Charge.
4. Never eat, drink while working in the laboratory.
5. Do not exchange the computers with others.
6. Do not misbehave in the laboratory.
7. Do not alter computer settings/software settings and pen drives should not be connected to computers without permission. Doing so will attract fines.
8. Do not remove anything from the experimental set up without permission.

Instructions and method of evaluation

1. In each Lab student have to show the record of previous Lab.
2. Each Lab will be evaluated for 15 marks and finally average will be taken for 15 marks.
3. Viva questions shall be asked in labs and attendance also can be considered for everyday Lab evaluation.
4. Tests shall be considered for 5 marks and final Lab assessment is for 20 marks.
5. Student has to score minimum 8 marks out of 20 to pass Lab component.

LIST OF LABS

		Page No:
Sl.No	LABS	
I	Introduction I Basics of Python II Programming Structure	10-15
1	2D-Plots of Cartesian and Polar Curves.	16-28
2	Finding angle between polar curves, curvature and radius of curvature of a given curve.	29-34
3	Finding Partial Derivatives and Jacobian.	35-39
4	Taylor Series Expansion and L'Hospital's Rule.	40-45
5	Solution of First Order Differential Equations and Plotting the Solution Curve.	46-50
6	Program to compute area, volume and Centre of gravity.	51-54
7	Evaluation of improper integrals.	55-57
8	Numerical solution of system of linear equations, test for consistency and Graphical representation.	58-63
9	Solution of system of linear equations using Gauss-Seidel iteration.	64-66
10	Compute eigenvalues and eigenvectors and find the largest and smallest eigenvalue by Rayleigh power method.	67-70

Introduction:

Programming Structures

Conditional structure

- Based on certain conditions the flow of execution of the program
- Decision-making statements in Python
 - if statement ---for implementing One-way branching
 - if..else statements---for implementing Two-way branching
 - nested if statements---for implementing Multiple branching
 - if-elif ladder --- for implementing Multiple branching

```
# Syntax:
# if condition:
#     statements
# Here condition is boolean expression which gives either True or False.

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

```
Enter an integer: 1
Entered value is positive
```

```
# Syntax:
# if condition 1:
#     statements 1
# elif condition 2:
#     statements 2
# elif condition 3:
#     statements 3
# else:
#     statements 4
# If condition 1 is True - Statements 1 will be executed.
# else if condition 2 is True - Statements 2 will be executed and so on.
# If any of the conditions is not True then statements in else block is executed.
# Example:
perc=float(input("Enter the percentage of marks obtained by a student:"))
if perc >= 75:
    print(perc,'% - Grade: Distinction')
elif perc >= 60:
    print(perc,'% - Grade: First class')
elif perc >=50:
    print(perc,'% - Grade: Second class')
else:
    print(perc,'% - Grade: Fail')
```

Enter the percentage of marks obtained by a student:71
71.0 % - Grade: First class

```
# Conversion Celsius to Fahrenheit and vice-versa:
def print_menu():
    print("1. Celsius to Fahrenheit")
    print("2. Fahrenheit to Celsius")

def Far():
    c=float(input("Enter Temperature in Celsius: "))
    f=c*(9/5)+32
    print("Temperature in Fahrenheit: {0:0.2f}".format(f))

def Cel():
    f=float(input("Enter Temperature in Fahrenheit: "))
    c=(f-32)*(5/9)
    print("Temperature in Celsius: {0:0.2f}".format(c))

print_menu()
choice=input("Which conversion would you like: ")
if (choice=='1'):
    Far()
elif (choice=='2'):
    Cel()
else :print("INVALID")
```

```
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
Which conversion would you like: 1
Enter Temperature in Celsius: 34
Temperature in Fahrenheit: 93.20
```

Control flow (Loops)

Loop types:

while loop

- Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

for loop

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

nested loops

- You can use one or more loop inside any another while, for or do..while loop.

1. While loop

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

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

```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1      #First step :a=0;b=1  second step:a=1;b=1+0
while a < 10:
    a,b=b,a+b
    print(a)
```

1
1
2
3
5
8
13

break statement

- It terminates the current loop and resumes execution at the next statement.
- The most common use for break is when some external condition is triggered requiring a hasty exit from a loop.
- The break statement can be used in both while and for loops.
- If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

```
# Use of break statement
i=1
while i<6:
    print(i)
    if i==3:
        break
    i+=1
```

1
2
3

Continue statement

- The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- The continue statement can be used in both while and for loops.

```
i=0
while i<6:
    i+=1
    if i==3:
        continue
    print(i)
```

1
2
4
5
6

2. 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
```

One can type the following examples and observe the outputs.

```
# 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; i=4, sum=7, ....
    sum=sum+i
print("Sum of first ",n,"natural numbers = ",sum)
```

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

```
# printing the elements of a list
fruits=['apple', 'banana','cherry','orange']
for x in fruits:
    print(x)
```

```
apple
banana
cherry
orange
```

Exercise:

1. Finding the factors of a number using for loop.
2. Check the given number is prime or not.
3. Find largest of three numbers.
4. Write a program to print even numbers between 25 and 45.
5. Write a program to print all numbers divisible by 3 between 55 and 75.

AJEEA

LAB 1: 2D plots of Cartesian and polar curves.

Objectives:

1. To plot cartesian curves
3. To plot polar curves
3. To plot implicit functions, using python

Syntax used in this Lab:

- `plot(x, y)`: plot x and y using default line style and color

Customizations:

- `plot(x, y, 'bo')`: plot x and y using blue circle markers
- `plot(y)`: plot y using x as index array 0..N-1
- `plot(y, 'r+')`: ditto, but with red plusses
- `plot(x, y, 'go-', linewidth=2, markersize=12)`

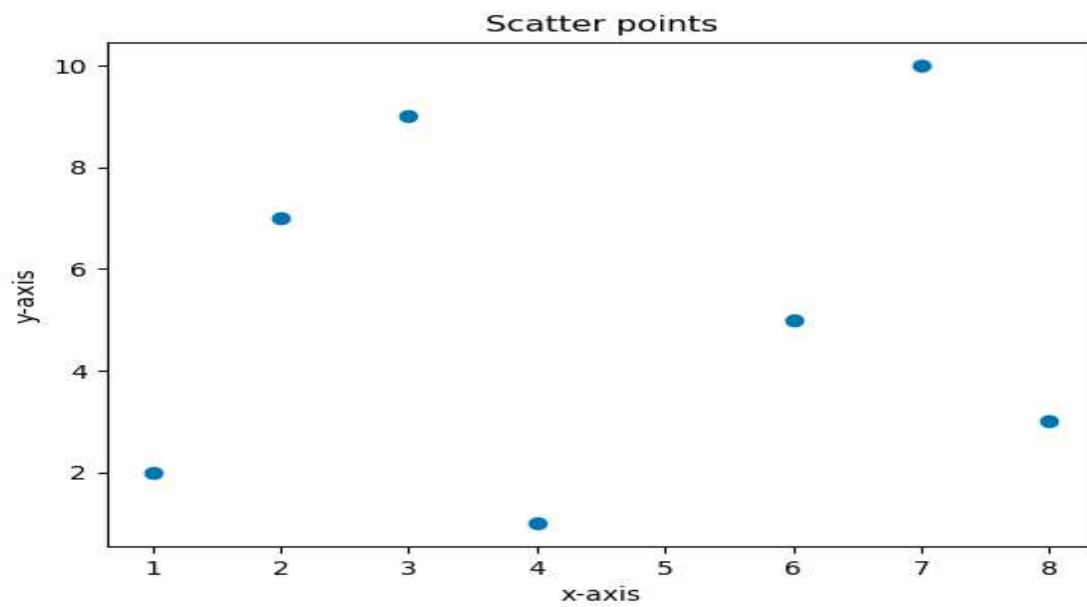
```
plot(x, y, color='green', marker='o',
      linestyle='dashed',linewidth=2,markersize=12)
```

Syntax:

```
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)
```

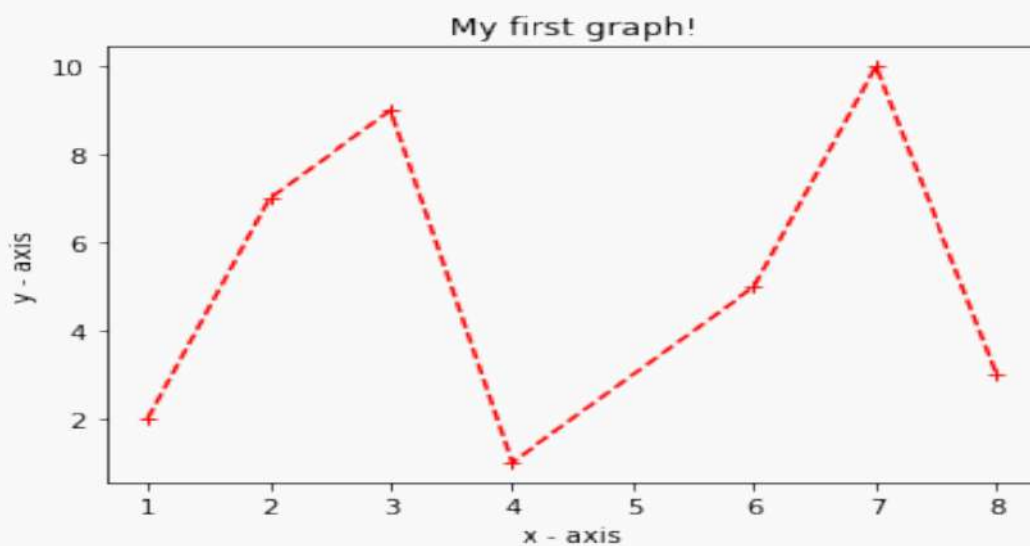
```
#example: Plotting points (scattered plots)
# importing the required module
import matplotlib.pyplot as plt

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 tittle to my graph
plt.show() #function to show the plot
```



Example: Plotting a line(Line plot)

```
import matplotlib.pyplot as plt
x=[1,2,3,4,6,7,8]
y=[2,7,9,1,5,10,3]
plt.plot(x,y,'r+--')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('ranjith first grafh')
ply.show()
```



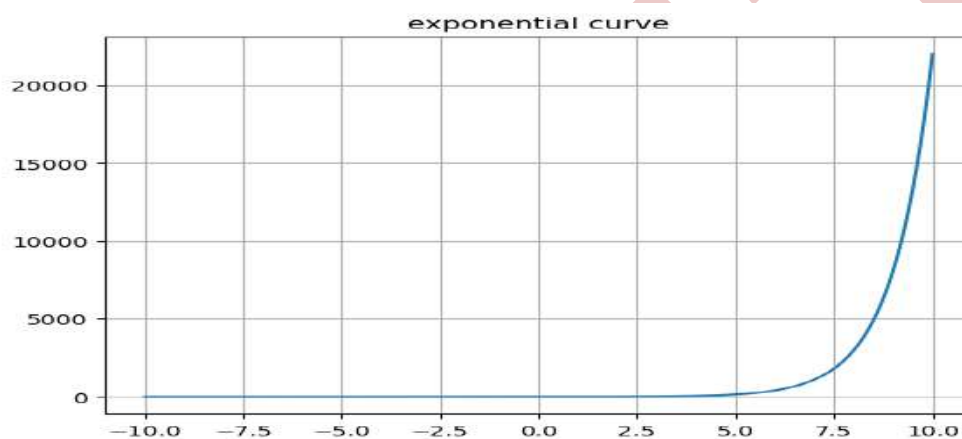
Functions

1. Exponential curve

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

x=np.arange(-10,10,0.001)

y=np.exp(x)
plt.plot(x,y)
plt.title("exponential curve")
plt.grid()
plt.show()
```

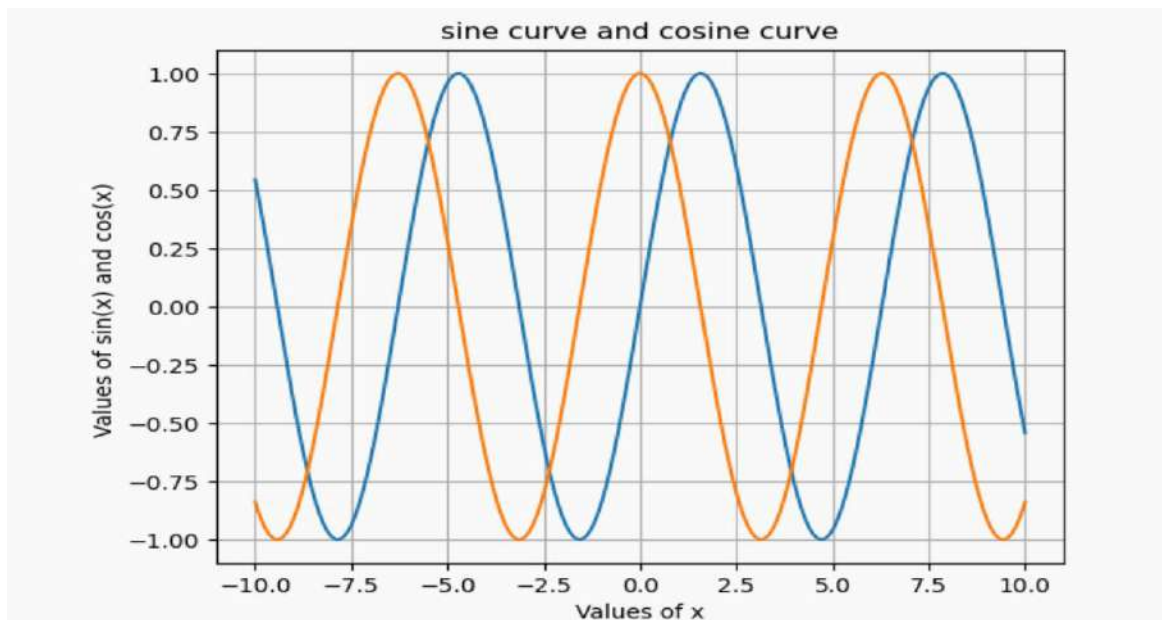


2. sine and cos curves

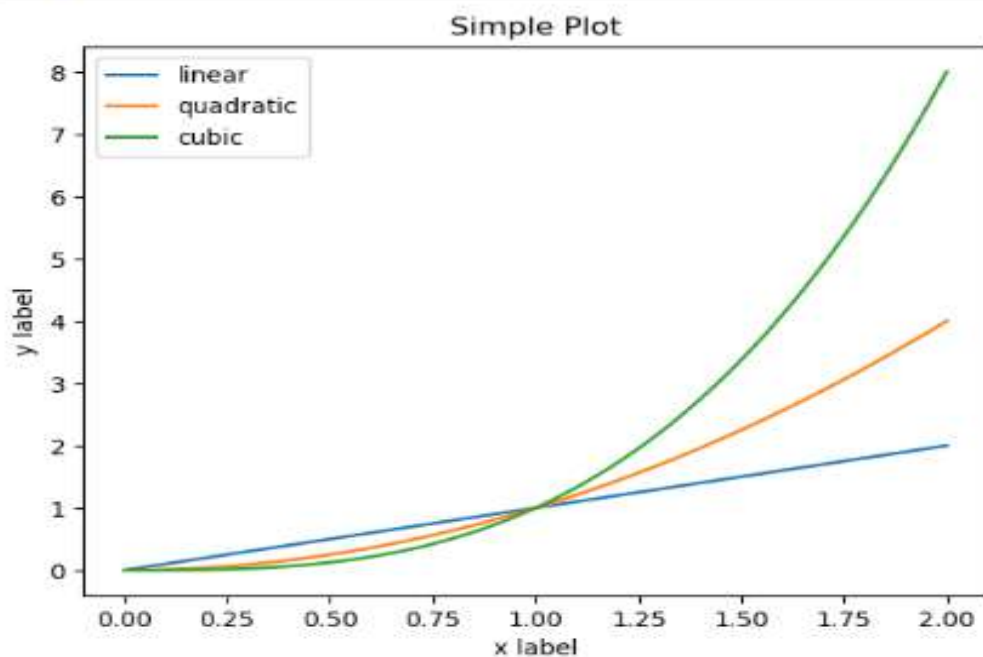
```
import numpy as np
import matplotlib.pyplot as plt

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()
```



```
In [1]: 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 label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



Implicit Function

Syntax:

```
plot_implicit(expr, x_var=None, y_var=None, adaptive=True, depth=0, points=300, line_color='blue', show=True, **kwargs)
```

- `expr` : The equation / inequality that is to be plotted.
- `x_var` (optional) : symbol to plot on x-axis or tuple giving symbol and range as (symbol, xmin, xmax)
- `y_var` (optional) : symbol to plot on y-axis or tuple giving symbol and range as (symbol, ymin, ymax)

If neither `x_var` nor `y_var` are given then the free symbols in the expression will be assigned in the order they are sorted.

- The following keyword arguments can also be used:

**** adaptive**: Boolean. The default value is set to True. It has to be set to False if you want to use a mesh grid.

**** depth** : integer. The depth of recursion for adaptive mesh grid. Default value is 0. Takes value in the range (0, 4).

**** points**: integer. The number of points if adaptive mesh grid is not used. Default value is 300.

**** show**: Boolean. Default value is True. If set to False, the plot will not be shown. See Plot for further information.

- `title` string. The title for the plot.
- `xlabel` string. The label for the x-axis
- `ylabel` string. The label for the y-axis

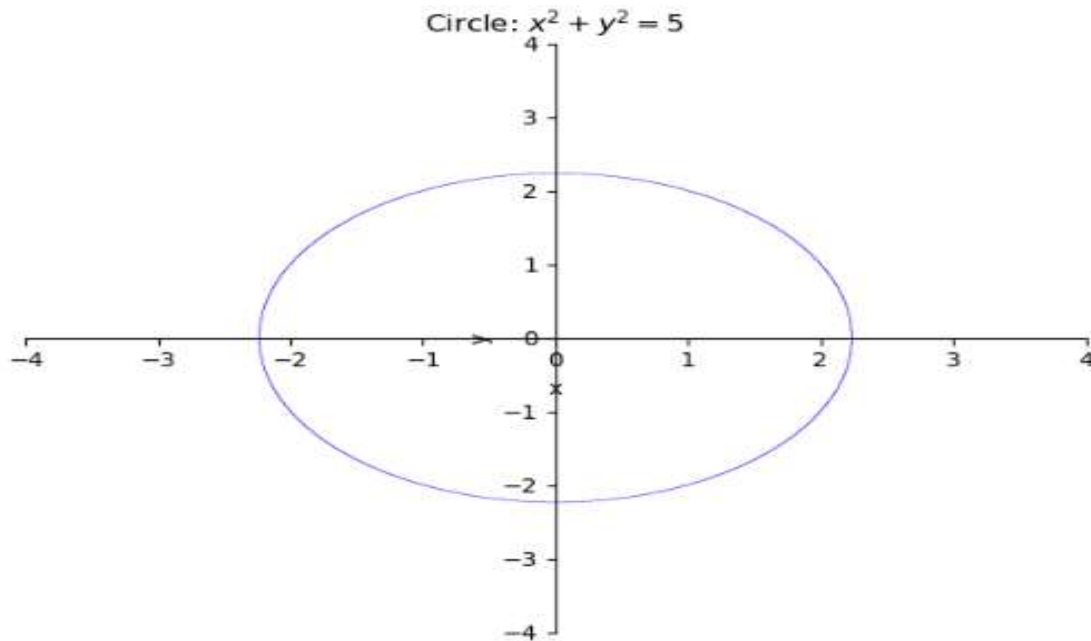
Aesthetics options:

- `line_color`: float or string. Specifies the color for the plot

Plot the following :

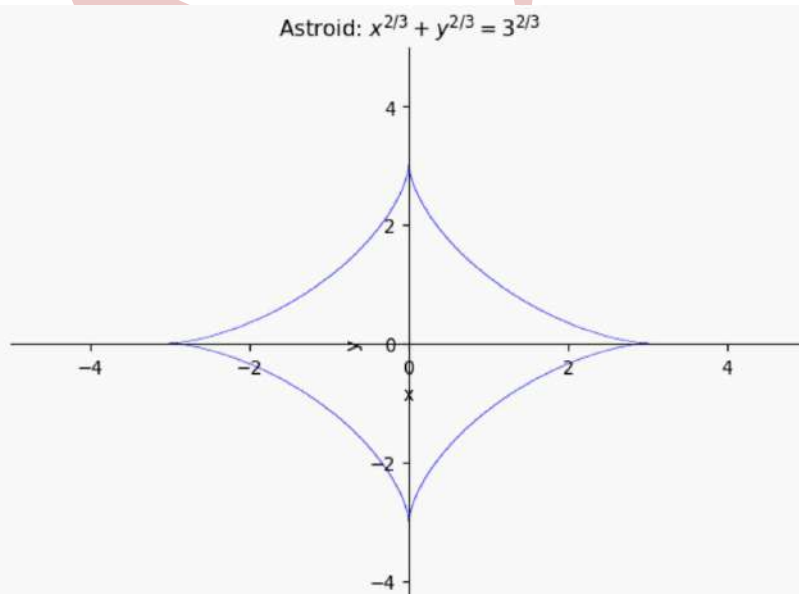
1. Circle: $x^2 + y^2 = 5$

```
#importing sympy package with plot_implicit, symbols and eq functions
from sympy import plot_implicit, symbols, Eq
x, y= symbols('x y')
p =plot_implicit(Eq(x**2 + y**2, 5),(x,-4,4),(y,-4,4), title= 'Circle: $x^2+y^2=5$')
```



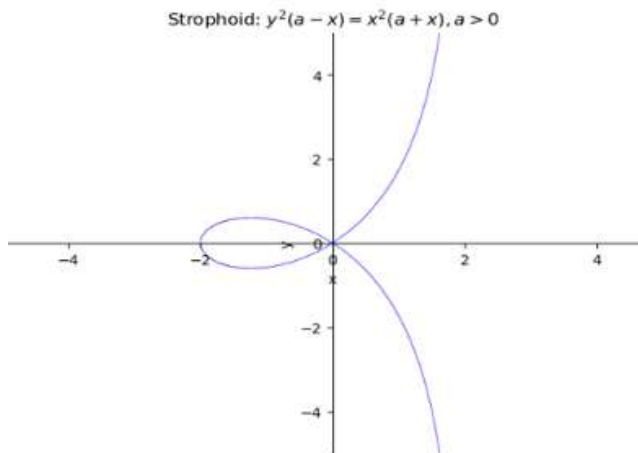
2. Astroid: $x^{\frac{2}{3}} + y^{\frac{2}{3}} = a^{\frac{2}{3}}, a > 0$

```
p2 = plot_implicit(
    Eq(x**(2/3) + y**(2/3), 3**(2/3)), (x, -5, 5), (y, -5, 5),title= 'Astroid: $x^{2/3}+y^{2/3}$')
```



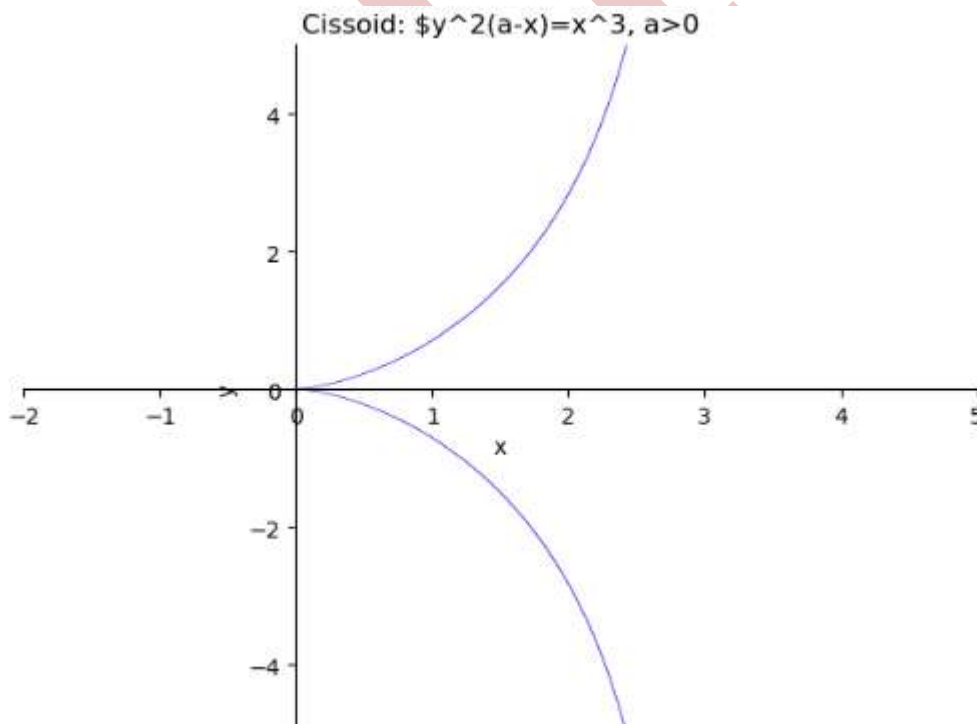
3. Strophoid: $y^2(a - x) = x^2(a + x), a > 0$

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



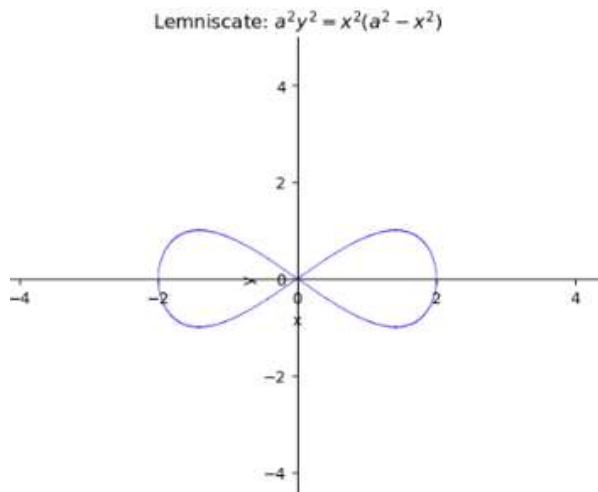
4. Cissoid: $y^2(a - x) = x^3, a > 0$

```
#cissoid :
from sympy import plot_implicit, symbols, Eq
x, y = symbols('x y')
p=plot_implicit( Eq((y**2)*(3-x),x**3), (x, -2,5), (y, -5,5),title= 'Cissoid: $y^2(a-x)=x^3, a>0$')
```



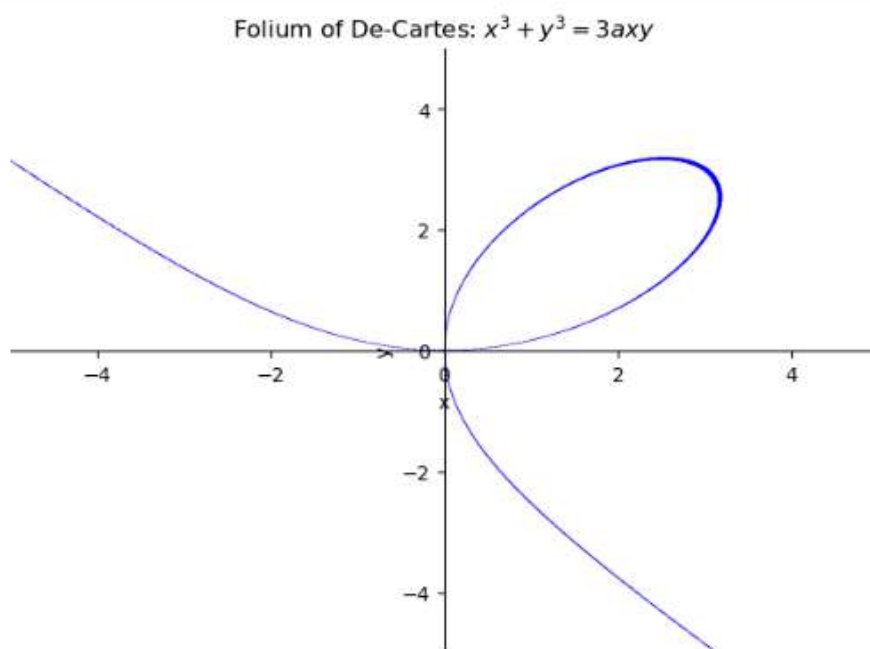
5. Lemniscate: $a^2y^2 = x^2(a^2 - x^2)$

```
#Lemniscate:
from sympy import plot_implicit, symbols, Eq
x, y = symbols('x y')
p= plot_implicit( Eq(4*(y**2),(x**2)*(4-x**2)), (x, -5, 5), (y, -5, 5), title= 'Lemniscate: $a^2y^2 = x^2(a^2 - x^2)$')
```



6. Folium of De-Cartes: $x^3 + y^3 = 3axy$

```
#5.Folium of de-cartes;
from sympy import plot_implicit, symbols, Eq
x, y = symbols('x y')
p= plot_implicit( Eq(x**3+y**3,3*x*y), (x, -5, 5), (y, -5, 5), title= 'Folium of De-Cartes: $x^3+y^3=3axy$')
```



Polar Curves

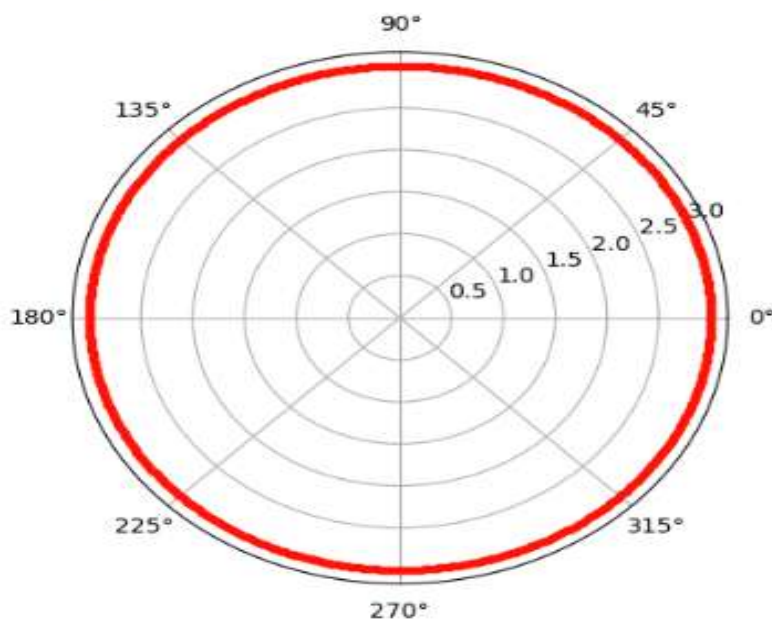
The `matplotlib.pyplot.polar()` function in `pyplot` module of `matplotlib` python library is used to plot the curves in polar coordinates.

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.

1. Circle: $r = p$, Where p is the radius of the circle

```
#Polar Curves
#circles
import numpy as np
import matplotlib.pyplot as plt
plt.axes(projection='polar')
r= 3
rads = np.arange (0, (2* np.pi),0.01)
#plotting the circle
for i in rads:
    plt.polar(i, r, 'r.')
plt.show()
```



2. Circle: $x = 5\cos\theta$; $y = 5\sin\theta$

```
#Plot circle in polar form
import numpy as np
from pylab import *
```

```
theta=linspace(0,2*np.pi,1000) #theta values calculated 1000 points from 0 to 2pi
```

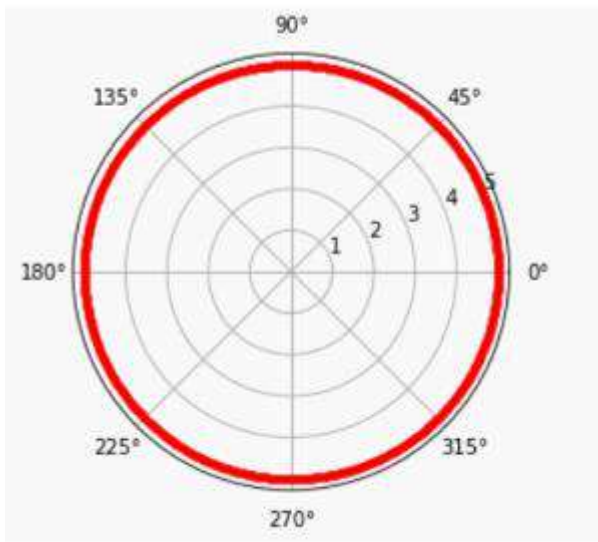
```
x=5*cos(theta)
```

```
y=5*sin(theta)
```

```
r=sqrt(x**2+y**2)
```

```
polar(theta,r,'r.')
```

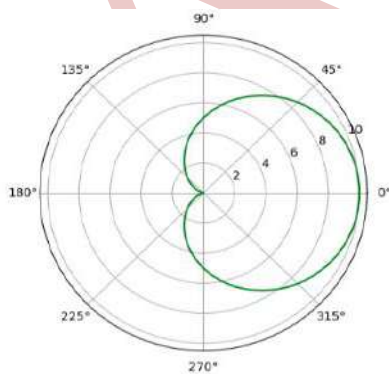
```
show()
```



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

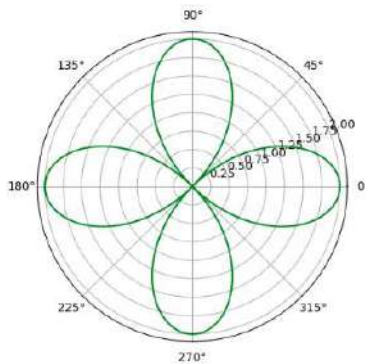
```
#cardioid:
from pylab import *
theta=linspace(0,2*np.pi,1000)
r1=5+5*cos(theta)

polar(theta,r1,'g')
show()
```



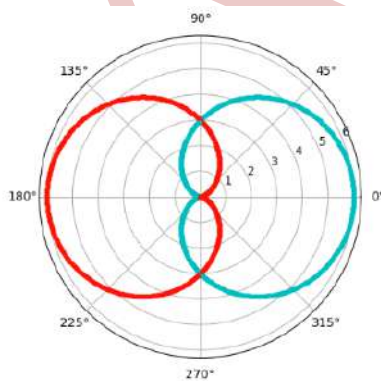
4. Four leaved Rose: $r = 2|\cos 2x|$

```
#four Leaved rose:
from pylab import *
theta= linspace(0,2*pi,1000)
r=2*abs(cos(2*theta))
polar(theta,r,'g')
show()
```



5. Cardioids: $r = a + a\cos(\theta)$ and $r = a - a\cos(\theta)$

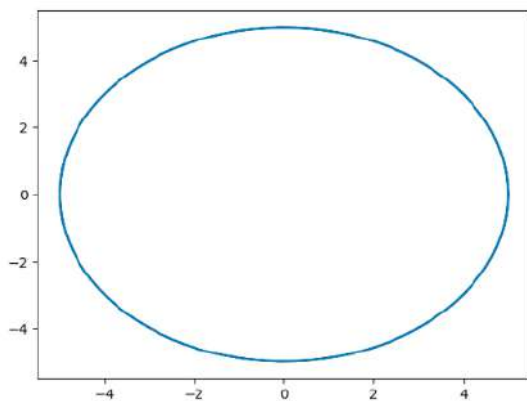
```
#cardioids
import numpy as np
import matplotlib.pyplot as plt
import math
plt.axes(projection='polar')
a=3
rad = np.arange(0, (2*np.pi), 0.01)
#plotting the cardioid
for i in rad:
    r= a+(a*np.cos(i))
    plt.polar(i,r,'c.')
    r1=a-(a*np.cos(i))
    plt.polar(i,r1,'r.')
    #display the polar plot
plt.show()
```



Parametric Equation

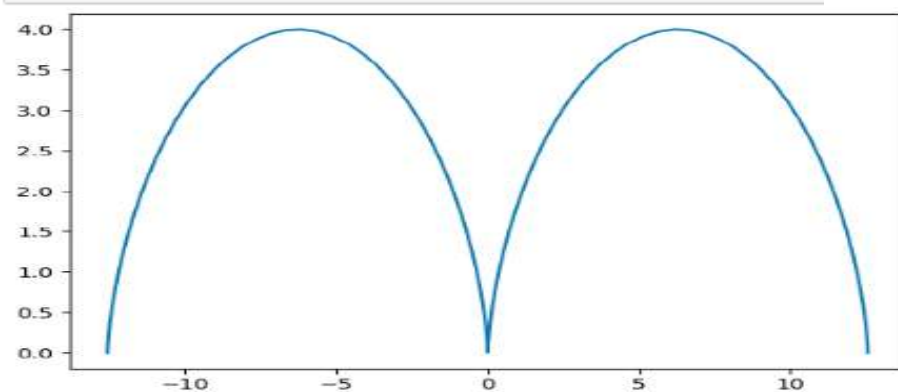
1. Circle: $x = a\cos(\theta); y = a\sin(\theta)$

```
import numpy as np
import matplotlib.pyplot as plt
def circle(r):
    x= []
    y= []
    for theta in np.linspace(-2*np.pi,2*np.pi,100):
        x.append(r*np.cos(theta))
        y.append(r*np.sin(theta))
    plt.plot(x,y)
    plt.show()
circle(5)
```



2. Cycloid: $x = a(\theta - \sin\theta); y = a(1 - \sin\theta)$

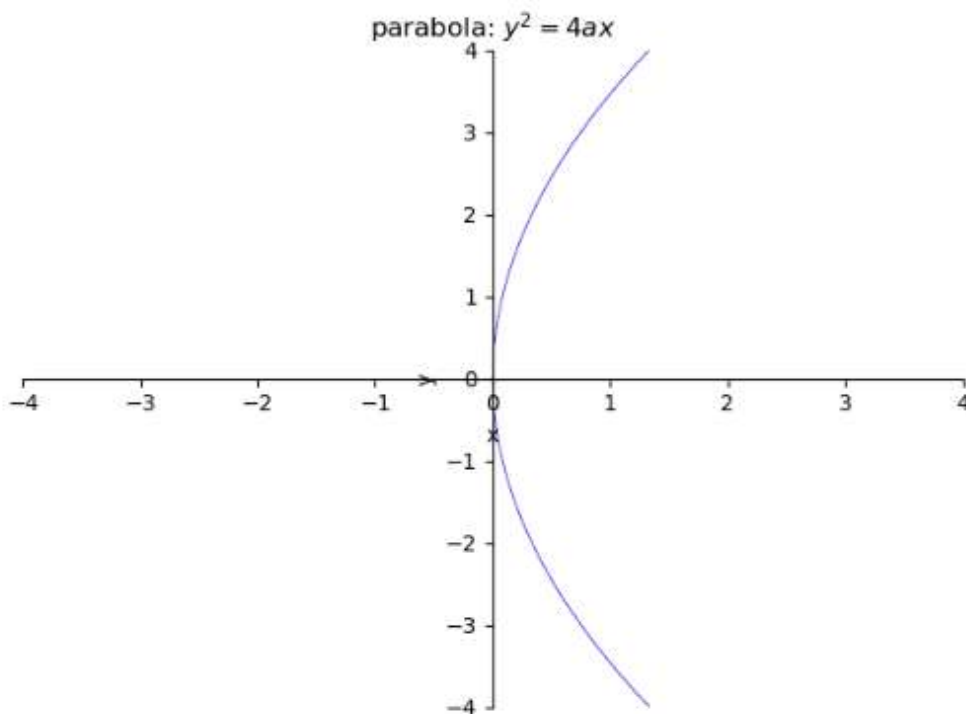
```
import numpy as np
import matplotlib.pyplot as plt
def cycloid(r):
    x=[]
    y=[]
    for theta in np.linspace(-2*np.pi,2*np.pi,100):
        x.append(r*(theta-np.sin(theta)))
        y.append(r*(1-np.cos(theta)))
    plt.plot(x,y)
    plt.show()
cycloid(2)
```



▼ Exercise:

- Plot the following:
 - Parabola $y^2 = 4ax$
 - Hyperbola $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$
 - Lower half of the circle: $x^2 + 2x = 4 + 4y - y^2$
 - $\cos\left(\frac{\pi x}{2}\right)$
 - $1 + \sin\left(x + \frac{\pi}{4}\right)$
 - Spiral of Archimedes: $r = a + b\theta$
 - Limacon: $r = a + b\cos\theta$

```
#excercise
#parabola:
from sympy import plot_implicit, symbols, Eq
x, y = symbols('x y')
p = plot_implicit(Eq(y**2, 4*a*x), (x, -4, 4), (y, -4, 4), title= 'parabola: $y^2=4ax$')
```



LAB 2: Finding angle between two polar curves, curvature and radius of curvature.

1. Angle between two polar curves

Objectives:

1. To find angle between two polar curves
2. To find radius of curvature , using python

Angle between radius vector and tangent is given by

$$\tan\phi = r \frac{d\theta}{dr}$$

- If $\tan\phi_1$ and $\tan\phi_2$ are angle between radius vector and tangent of two curves then $|\phi_1 - \phi_2|$ is the angle between two curves at the point of intersection.

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

Syntax of diff():

diff(function,variable)

▼ syntax of Substitute : subs()

math_expression.subs(variable, substitute)

Parameters: *variable* – It is the variable or expression which will be substituted.

substitute – It is the variable or expression or value which comes as substitute.

Returns: Returns the expression after the substitution.

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

```
#Angle between two curves:
from sympy import *
import math as mt
r, t= symbols ('r,t')
r1=4*(1+cos (t));
r2=5*(1-cos (t));
dr1=diff(r1, t)
dr2=diff(r2,t)
tanphi1=r1/dr1
tanphi2=r2/dr2
q=solve (r1-r2, t)
tanphi1=tanphi1.subs({t: float (q[1])})
tanphi2=tanphi2.subs({t: float (q[1])})
phi1=atan (tanphi1)
phi2= atan (tanphi2)
w=abs(phi1-phi2)
print('Angle between curves in radians is %0.3f'%(w))
print('angle between curves in degees is',mt.degrees(w))
```

Angle between curves in radians is 1.571
angle between curves in degees is 90.0

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

```
#r=4cost and r=5sint
from sympy import *
import math as mt
r, t=symbols ('r,t')
r1=4*(cos(t))
r2=5*(sin(t))
dr1=diff(r1,t)
dr2=diff(r2,t)
tanphi1=r1/dr1
tanphi2=r2/dr2
q=solve(r1-r2,t)
tanphi1=tanphi1.subs({t:float (q[0])})
tanphi2=tanphi2.subs({t:float (q[0])})
phi1=atan (tanphi1)
phi2=atan(tanphi2)
w=abs(phi1-phi2)
print('Angle between curves in radians is %0.3f'%(w))
print('angle between curves in degrees is',mt.degrees(w))
```

Angle between curves in radians is 1.571
angle between curves in degrees is 90.0

2. Radius of curvature

Formula to calculate Radius of curvature in cartesian form is

$$\rho = \frac{(1+y_1^2)^{3/2}}{y_2}$$

Formula to calculate Radius of curvature in polar form is

$$\rho = \frac{(r^2+r_1^2)^{3/2}}{r^2+2r_1^2-rr_2}$$

▾ Synatx

1. Derivative()

Derivative(expression, reference variable)

expression – A SymPy expression whose unevaluated derivative is found.

reference variable– Variable with respect to which derivative is found.

Returns: Returns an unevaluated derivative of the given expression

2. doit()

doit(x)

Return : evaluated object

3. simplify() **simplify(expression) expression** – It is the mathematical expression which needs to be simplified.

Returns: Returns a simplified mathematical expression corresponding to the input expression.

4. display() **display(expression) expression** – It is the mathematical expression which needs to be simplified. **Returns:** Displays the expression.

1. Find the radius of curvature, $r = 4(1 + \cos t)$ at $t = \pi/2$.

```
from sympy import*
t=Symbol('t')
r=Symbol('r')
r=4*(1+cos(t))
r1=Derivative(r,t).doit()
r2=Derivative(r1,t).doit()
rho=(r**2+r1**2)**(1.5)/(r**2+2*r1**2-r*r2);
rho1=rho.subs(t,pi/2)
print('the radius of curvature is %3.4f units' %rho1)
```

the radius of curvature is 2.4495 units

2. Find the radius of curvature for $r = a \sin(nt)$ at $t = \pi/2$ and $n = 1$.

S

```
from sympy import*
t,r,a,n=symbols('t,r,a,n')
r=a*sin(n*t)
r1=diff(r,t)
r2=diff(r1,t)
rho=(r**2+r1**2)**1.5/(r**2+2*r1**2-r*r2);
rho1=rho.subs(t,pi/2)
rho1.subs(n,1)
print("the radius of curavture is")
display(simplify(rho1))
```

the radius of curavture is

$$\frac{(a^2 (n^2 \cos^2 (\frac{\pi n}{2}) + \sin^2 (\frac{\pi n}{2})))^{1.5}}{a^2 (-n^2 \sin^2 (\frac{\pi n}{2}) + 2n^2 + \sin^2 (\frac{\pi n}{2}))}$$

▼ Parametric curves

The formula to calculate Radius of curvature is $\rho = \frac{(x'^2 + y'^2)^{\frac{3}{2}}}{y''x' - x''y'}$.

$$x' = \frac{dx}{dt}, x'' = \frac{d^2x}{dt^2}, y' = \frac{dy}{dt}, y'' = \frac{d^2y}{dt^2}$$

1. Find radius of curvature of $x = a\cos(t)$, $y = a\sin(t)$.

```

: from sympy import*
from sympy.abc import rho,x,y,r,K,t,a,b,c,alpha
y=a*sin(t)
x=a*cos(t)
dydx=simplify(Derivative(y,t).doit())/simplify(Derivative(x,t).doit())
rho=simplify((1+dydx**2)**1.5/(Derivative(dydx,t).doit()/(Derivative(x,t).doit()))
print('radius of curvature is')
display(ratsimp(rho))
t1=pi/2
r1=5
rho1=rho.subs(t,t1);
rho2=rho1.subs(a,r1);
print('\n\nradius of curvature at r=5 and t=pi/2is',simplify(rho2));
curvature=1/rho2;
print('\n\n curvature at (5,pi/2)is',float(curvature))

```

radius of curvature is

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

radius of curvature at r=5 and t=pi/2is -5

curvature at (5,pi/2)is -0.2

2. Find the radius of curvature of $y = (a\sin(t))^{3/2}$; $x = (a\cos(t))^{3/2}$.

```

: from sympy import*
from sympy.abc import rho,x,y,r,k,t,a,b,c,alpha
y=(a*sin(t))**(3/2)
x=(a*cos(t))**(3/2)
dydx=simplify(Derivative(y,t).doit())/simplify(Derivative(x,t).doit())
rho=simplify((1+dydx**2)**1.5/(Derivative(dydx,t).doit()/(Derivative(x,t).doit()))
print('radius of curvatureis')
display(ratsimp(rho))
t1=pi/4
r1=1;
rho1=rho.subs(t,t1);
rho2=rho1.subs(a,r1);
display('\n\nRadius of curvature at r=1 and t=pi/4is',simplify(rho2));
curvature=1/rho2;
print('\n\n curvature at(1,pi/4)is',float(curvature))

```

radius of curvatureis

$$\frac{3.0(a \cos(t))^{3.0} \left(\frac{(a \sin(t))^{3.0}}{(a \cos(t))^{3.0} \tan^4(t)} + 1 \right)^{1.5} \sin^2(t) \tan^2(t)}{(a \sin(t))^{1.5}}$$

'\n\nRadius of curvature at r=1 and t=pi/4is'

-2.52268924576114

curvature at(1,pi/4)is -0.39640237166757364

▼ Exercise:

1. Find the angle between radius vector and tangent to the following polar curves
 - a) $r = a\theta$ and $r = \frac{a}{\theta}$
 - b) $r = 2\sin(\theta)$ and $r = 2\cos(\theta)$
2. Find the radius of curvature of $r = 2(1 - \cos(t))$ at $t = \frac{\pi}{2}$.
3. Find radius of curvature of $x = a\cos^3(t)$, $y = a\sin^3(t)$ at $t = 0$.
4. Find the radius of curvature of $r = a\cos(t)$ at $t = \frac{\pi}{4}$.
5. Find the radius of curvature of $x = a(t - \sin(t))$ and $y = a(1 - \cos(t))$ at $t = \pi$.

LAB 3: Finding partial derivatives and Jacobian of functions of several variables.

Objectives:

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:

1. To create a matrix:

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

Ex: A 3×3 matrix can be defined as

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

2. Evaluate the determinant of a matrix M .

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

3. To evaluates derivative of function w.r.t variable.

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

4. If function is of two or more than two independent variable then it differentiates the function partially w.r.t variable.

If $u = u(x, y)$ then,

- $\frac{\partial u}{\partial x} = \text{diff}(u, x)$
- $\frac{\partial u}{\partial y} = \text{diff}(u, y)$
- $\frac{\partial^2 u}{\partial x^2} = \text{diff}(u, x, x)$
- $\frac{\partial^2 u}{\partial x \partial y} = \text{diff}(u, x, y)$

I. Partial derivatives

The partial derivative of $f(x, y)$ with respect to x at the point (x_0, y_0) is

$$f_x = \frac{\partial f}{\partial x} \text{at}(x_0, y_0) = \lim_{h \rightarrow 0} \frac{f(x_0 + h, y_0) - f(x_0, y_0)}{h}.$$

The partial derivative of $f(x, y)$ with respect to xy at the point (x_0, y_0) is

$$f_y = \frac{\partial f}{\partial y} \text{at}(x_0, y_0) = \lim_{h \rightarrow 0} \frac{f(x_0, y_0 + h) - f(x_0, y_0)}{h}.$$

1. Prove that mixed partial derivatives , $u_{xy} = u_{yx}$ for $u = \exp(x)(x\cos(y) - y\sin(y))$.

```
from sympy import *
x,y=symbols('x y')
u=exp(x)*(x*cos(y)-y*sin(y))
ux=diff(u,x)
uy=diff(u,y)
uxy=diff(ux,y)
uyx=diff(uy,x)
if uxy==uyx:
    print('mixed partial derivatives are equal')
else:
    print('mixed partial derivatives are not equal')
```

mixed partial derivatives are equal

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

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

$(x\cos(y) - y\sin(y))e^x$

ans: 0.0

II Jacobians

Let $x = g(u, v)$ and $y = h(u, v)$ be a transformation of the plane. Then the Jacobian of this transformation is

$$\mathbf{J} = \frac{\partial(x, y)}{\partial(u, v)} = \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix}.$$

1. If $u = xy/z, v = yz/x, w = zx/y$ then prove that $J = 4$.

```

from sympy import *
x,y,z=symbols('x y z')
u=x*y/z
v=y*z/x
w=z*x/y
ux=diff(u,x)
uy=diff(u,y)
uz=diff(u,z)
vx=diff(v,x)
vy=diff(v,y)
vz=diff(v,z)
wx=diff(w,x)
wy=diff(w,y)
wz=diff(w,z)
J=Matrix([[ux,uy,uz],[vx,vy,vz],[wx,wy,wz]]);
print("the jacobian matrix is\n")
display(J)
Jac=det(J)
print('\n\nJ=',Jac)

```

the jacobian matrix is

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

$J = 4$

2. If $u = x + 3y^2 - z^3$, $v = 4x^2yz$, $w = 2z^2 - xy$ then prove that at $(1, -1, 0)$, $J = 20$.

```

from sympy import *
x,y,z=symbols('x y z')
u=x+3*y**2-z**3
v=4*x**2*y*z
w=2*z**2-x*y
ux=diff(u,x)
uy=diff(u,y)
uz=diff(u,z)
vx=diff(v,x)
vy=diff(v,y)
vz=diff(v,z)
wx=diff(w,x)
wy=diff(w,y)
wz=diff(w,z)

J=Matrix([[ux,uy,uz],[vx,vy,vz],[wx,wy,wz]]);

print("the jacobian matrix is\n")
display(J)
Jac=det(J)
print('\n\n J=\n')
display(Jac)
J1=J.subs([(x,1),(y,-1),(z,0)])
print('\n\n J at (1,-1,0):\n')
Jac1=det(J1)
display(Jac1)

```

the jacobian matrix is

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

J=

$$4x^3y - 24x^2y^3 + 12x^2yz^3 + 16x^2z^2 - 192xy^2z^2$$

J at (1, -1, 0):

20

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

```
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=diff(X,rho)
dy=diff(Y,rho)
dz=diff(Z,rho)
dx1=diff(X,phi)
dy1=diff(Y,phi)
dz1=diff(Z,phi)
dx2=diff(X,theta)
dy2=diff(Y,theta)
dz2=diff(Z,theta)

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)))
```

the jacobian matrix is

$$\begin{bmatrix} \sin(\theta) \cos(\phi) & \cos(\phi) \cos(\theta) & \sin(\phi) \\ -\rho \sin(\phi) \sin(\theta) & -\rho \sin(\phi) \cos(\theta) & \rho \cos(\phi) \\ \rho \cos(\phi) \cos(\theta) & -\rho \sin(\theta) \cos(\phi) & 0 \end{bmatrix}$$

J=

$$\rho^2 \sin^2(\phi) \sin^2(\theta) \cos(\phi) + \rho^2 \sin^2(\phi) \cos(\phi) \cos^2(\theta) + \rho^2 \sin^2(\theta) \cos^3(\phi) + \rho^2 \cos^3(\phi) \cos^2(\theta)$$

Exercise:

Plot the following:

1. If $u = \tan^{-1}(y/x)$ verify that $\frac{\partial^2 u}{\partial y \partial x} = \frac{\partial^2 u}{\partial x \partial y}$.
Ans: True
2. If $u = \log\left(\frac{x^2+y^2}{x+y}\right)$ show that $xu_x + yu_y = 1$.
Ans: True
3. If $x = u - v, y = v - uvw$ and $z = uvw$ find Jacobian of x, y, z w.r.t u, v, w .
Ans: uv
4. If $x = r\cos(t)$ and $y = r\sin(t)$ then find the $\frac{\partial(x,y)}{\partial(r,t)}$.
Ans: $J = r$
5. 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 $(-2,-1,1)$.
Ans: 752

LAB 4: Applications of Maxima and Minima of functions of two variables, Taylor series expansion and L'Hospital's Rule

Objectives:

1. to find the maxima and minima of function of two variables.
2. to expand the given single variable function as Taylor's and Maclaurin series.
3. to find the limiting value of the given function $f(x)$ as $x \rightarrow a$.

Syntax for the commands used:

1. To solve

```
sympy.solve(expression)
```

Returns the solution to a mathematical expression/polynomial.

2. To evaluate an expression

```
sympy.evalf()
```

Returns the evaluated mathematical expression.

3. To construct an instant function

```
sympy.lambdify(variable, expression, library)
```

Converts a SymPy expression to an expression that can be numerically evaluated. lambdify acts like a lambda function, except it, converts the SymPy names to the names of the given numerical library, usually NumPy or math.

4. To find the limit of a function

```
Limit(expression, variable, value)
```

Maxima and minima problem

In []: `Extremise row(x,y)=x^3+3xy^2-15x^2-15y^2+72x`

```

: from sympy import*
var('x y')
f=x**2+y**2+3*x-3*y+4
fx=diff(f,x)
fy=diff(f,y)
soln=solve([fx,fy],(x,y))
display(soln)
c1=soln[x]
c2=soln[y]
print(f"The critical points are {c1} and {c2}")
A=fx=diff(fx,x)
B=fxy=diff(fx,y)
C=fyy=diff(fy,y)
A=A.subs({x:c1},{y:c2}).evalf()
B=B.subs({x:c1},{y:c2}).evalf()
C=C.subs({x:c1},{y:c2}).evalf()
display(A,B,C)
if(A<0) and(A*C*B**2)>0:
    print(f"given function f has maximum value at ({c1},{c2}")
    f_max=f.subs({x:c1,y:c2}).evalf()
    print(f"the maximum of the function is f_max={f_max}")
elif(A>0) and (A*C-B**2)>0:
    print(f"given function f has minimum value at({c1},{c2}")
    f_min=f.subs({x:c1,y:c2}).evalf()
    print(f"the minimum of the function is f_min={f_min}")
else:
    print(f"the function requires further investigation")

```

{x: -3/2, y: 3/2}

The critical points are -3/2 and 3/2

2.0

0

2.0

given function f has minimum value at(-3/2,3/2)
the minimum of the function is f_min=-0.5000000000000000

In []: `Extremise row(x,y)=x^3+3xy^2-15x^2-15y^2+72x`

```

In [24]:
from sympy import*
x,y=symbols('x,y')
f=x**3+3*x*y**2-15*x**2-15*y**2+72*x
fx=diff(f,x)
fy=diff(f,y)
eq1,eq2=Eq(fx,0),Eq(fy,0)
sol=solve([eq1,eq2],(x,y))
print("The critical points are",sol)
fxx=diff(fx,x)
fxy=diff(fx,y)
fyy=diff(fy,y)
for i,j in sol:
    A=fxx.subs({x:i,y:j}).evalf()
    B=fxy.subs({x:i,y:j}).evalf()
    C=fyy.subs({x:i,y:j}).evalf()
    if(A*C-B*B>0 and A>0):
        print("The function has minimum at (" ,i," ,",j,")")
        f_min=f.subs({x:i,y:j}).evalf()
        print("Minimum value:",f_min)
    elif(A*C-B*B>0 and A<0):
        print("The function has maximum at (" ,i," ,",j,")")
        f_max=f.subs({x:i,y:j}).evalf()
        print("Maximum value:",f_max)
    elif(A*C-B*B<0):
        print("(" ,i," ,",j,")is a saddle point")
    elif(A*C-B*B==0):
        print("it needs further investigations")

```

```

The critical points are [(4, 0), (5, -1), (5, 1), (6, 0)]
The function has maximum at ( 4 , 0 )
Maximum value: 112.000000000000
( 5 , -1 )is a saddle point
( 5 , 1 )is a saddle point
The function has minimum at ( 6 , 0 )
Minimum value: 108.000000000000

```

```
In [ ]: extremise integration(x,y)=x**3+y**3-3x-12y+20
```

```
In [7]: from sympy import*
x,y=symbols('x,y')
f=x**3+y**3-3*x-12*y+20
fx=diff(f,x)
fy=diff(f,y)
eq1,eq2=Eq(fx,0),Eq(fy,0)
sol=solve([eq1,eq2],(x,y))
print("The critical points are",sol)
fxx=diff(fx,x)
fxy=diff(fx,y)
fyy=diff(fy,y)
for i,j in sol:
    A=fxx.subs({x:i,y:j}).evalf()
    B=fxy.subs({x:i,y:j}).evalf()
    C=fyy.subs({x:i,y:j}).evalf()
    if(A*C-B*B>0 and A>0):
        print("the function has minimum at(",i,",",j,")")
        f_min=f.subs({x:i,y:j}).evalf()
        print("Minimum value:",f_min)
    elif(A*C-B*B>0 and A<0):
        print("The function has maximum at(",i,",",j,")")
        f_max=f.subs({x:i,y:j}).evalf()
        print("Maximum value:",f_max)
    elif(A*C-B*B<0):
        print("(",i,",",j,")is a saddle point")
    elif(A*C-B*B==0):
        print("it needs further investigations")

```

```

The critical points are [(-1, -2), (-1, 2), (1, -2), (1, 2)]
The function has maximum at( -1 , -2 )
Maximum value: 38.0000000000000
( -1 , 2 )is a saddle point
( 1 , -2 )is a saddle point
the function has minimum at( 1 , 2 )
Minimum value: 2.00000000000000

```

Maclaurin Series

In []: Find the Maclaurin series expansion of $\sin(x)+\cos(x)$ upto 3rd degree term

```
from sympy import *
var('x')
y=sin(x)+cos(x)
y1=diff(y,x)
y2=diff(y,x,2)
y3=diff(y,x,3)
y4=diff(y,x,4)
display(y,y1,y2,y3,y4)
y=lambdify(x,y)
y1=lambdify(x,y1)
y2=lambdify(x,y2)
y3=lambdify(x,y3)
y4=lambdify(x,y4)
y=y(0)+x*y1(0)+(x**2)/2*y2(0)+(x**3)/6*y3(0)+(x**4)/24*y4(0)
print(f"Required maclaurin series is{y}")
```

$\sin(x) + \cos(x)$

$-\sin(x) + \cos(x)$

$-(\sin(x) + \cos(x))$

$\sin(x) - \cos(x)$

$\sin(x) + \cos(x)$

Required maclaurin series is $0.0416666666666667*x**4 - 0.166666666666667*x**3 - 0.5*x**2 + 1.0*x + 1.0$

In []: Find the Maclaurin series expansion of $\log(1+x)$ upto 3rd degree term

```
from sympy import *
var('x')
y=log(1+x)
y1=diff(y,x)
y2=diff(y,x,2)
y3=diff(y,x,3)
y4=diff(y,x,4)
display(y,y1,y2,y3,y4)
y=lambdify(x,y)
y1=lambdify(x,y1)
y2=lambdify(x,y2)
y3=lambdify(x,y3)
y4=lambdify(x,y4)
y=y(0)+x*y1(0)+(x**2)/2*y2(0)+(x**3)/6*y3(0)+(x**4)/24*y4(0)
print(f"Required maclaurin series is{y}")
```

$\log(x + 1)$

$$\frac{1}{x+1}$$

$$-\frac{1}{(x+1)^2}$$

$$\frac{2}{(x+1)^3}$$

$$-\frac{6}{(x+1)^4}$$

Required maclaurin series is $-0.25*x**4 + 0.3333333333333333*x**3 - 0.5*x**2 + 1.0*x$

L'HOSPITAL'S RULE

1. $\lim_{x \rightarrow 0} \frac{\sin(x)}{x}$

```
from sympy import Limit, Symbol, exp, sin
x=Symbol('x')
l=Limit((sin(x))/x,x,0).doit()
print(l)
```

2. Evaluate $\lim_{x \rightarrow 1} \frac{(5x^4 - 4x^2 - 1)}{(10 - x - 9x^3)}$

```
from sympy import *
x=Symbol('x')
l=Limit(((5*x**4-4*x**2-1)/(10-x-9*x**3)),x,1).doit()
print(l)
```

3. Prove that $\lim_{x \rightarrow \infty} (1 + \frac{1}{x})^x = e$

```
from sympy import *
from math import inf
x=Symbol('x')
l=Limit(((1+1/x)**x),x,inf).doit()
display(l)
```

In []: Evaluate limit x to $\theta(a^x + b^x + c^x/3)1/x$

```
In [3]: from sympy import*
var('a b c x')
k=limit((((a**x+b**x+c**x)/3)**(1/x)),x,0)
display(k)
print(k)
```

$$\sqrt[3]{a}\sqrt[3]{b}\sqrt[3]{c}$$

$$a**(1/3)*b**(1/3)*c**(1/3)$$

In []: Evaluate limit x to 0 (sin(x)/x)1/x^2

```
In [4]: from sympy import*
var('x')
k=limit((sin(x)/x)**(1/(x**2)),x,0)
display(k)
print(k)
pprint(k)
```

$$e^{-\frac{1}{6}}$$

$$\exp(-1/6)$$

$$-1/6$$

$$e$$

Exercise:

Plot the following:

- Find the Taylor Series expansion of $y = e^{-2x}$ at $x = 0$ upto third degree term.
Ans: $-0.333333333333333 * x^3 + 0.666666666666667 * x^2 - 1.0 * x + 1.0$
- Expand $y = xe^{-3x^2}$ as Maclaurin's series upto fifth degree term.
Ans: $x * (0.75 * x^4 - 0.75 * x^2 + 0.5)$
- Find the Taylor Series expansion of $y = \cos(x)$ at $x = \frac{\pi}{3}$.
Ans: $0.010464x^4 + 0.00544x^3 - 0.155467x^2 - 0.1661389657x + 0.827151505$
- Find the Maclaurin's series expansion of $y = e^{-\sin^{-1}(x)}$ at $x = 0$ upto x^3 term. Also Plot the graph.
Ans: $-0.0833333333333333x^3 + 0.166666666666667x^2 - 0.5x + 1.0$
- Evaluate $\lim_{x \rightarrow 0} \frac{2\sin x - \sin 2x}{x - \sin x}$
Ans: 6
- Evaluate $\lim_{x \rightarrow \infty} [\sqrt{x^2 + x + 1} - \sqrt{x^2 + 1}]$.
Ans: 0.5

LAB 5: Solution of First order differential equation and plotting the solution curves

Objectives:

1. To find the solution of first order differential equations.
2. To represent the solution graphically.

Syntax for the commands used:

1. dsolve()

```
sympy.solvers.ode.dsolve(eq, func=None, hint='default', simplify=True, ics=None, xi=None, eta=None, x0=0, n=6, **kwargs)
```

Parameters

- **eq:** eq can be any supported ordinary differential equation (see the ode docstring for supported methods). This can either be an Equality, or an expression, which is assumed to be equal to 0.
- **func:** f(x) is a function of one variable whose derivatives in that variable make up the ordinary differential equation eq. In many cases it is not necessary to provide this; it will be autodetected (and an error raised if it could not be detected).
- **hint:** hint is the solving method that you want dsolve to use. Use `classify_ode(eq, f(x))` to get all of the possible hints for an ODE. The default hint, default, will use whatever hint is returned first by `classify_ode()`. See Hints below for more options that you can use for hint.
- **simplify:** simplify enables simplification by `odesimp()`. See its docstring for more information. Turn this off, for example, to disable solving of solutions for func or simplification of arbitrary constants. It will still integrate with this hint. Note that the solution may contain more arbitrary constants than the order of the ODE with this option enabled.
- **xi and eta:** are the infinitesimal functions of an ordinary differential equation. They are the infinitesimals of the Lie group of point transformations for which the differential equation is invariant. The user can specify values for the infinitesimals. If nothing is specified, xi and eta are calculated using `infinitesimals()` with the help of various heuristics.
- **ics:** is the set of initial/boundary conditions for the differential equation. It should be given in the form of `{f(x0): x1, f(x).diff(x).subs(x, x2): x3}` and so on. For power series solutions, if no initial conditions are specified f(0) is assumed to be C0 and the power series solution is calculated about 0.

- **x0:** is the point about which the power series solution of a differential equation is to be evaluated.
- **n:** gives the exponent of the dependent variable up to which the power series solution of a differential equation is to be evaluated. also be much faster than all, because `integrate()` is an expensive routine.
- **Usage:**
 - Solves any kind of ordinary differential equation and system of ordinary differential equations.
 - Usage `dsolve(eq, f(x), hint)` – > Solve ordinary differential equation eq for function `f(x)`, using method `hint`.

2. `odeint()`: The `odeint` (ordinary differential equation integration) library is a collection of advanced numerical algorithms to solve initial-value problems.

```
y = odeint(model, y0, t)
```

Parameters:

- **model:** Function name that returns derivative values at requested `y` and `t` values as `dydt = model(y,t)`
- **y0:** Initial conditions of the differential states
- **t:** Time points at which the solution should be reported.

3. `linspace()`:

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

Parameters

- **start:** It represents the starting value of the sequence.
- **stop:** It represents the ending value of the sequence.
- **num:** It generates a number of samples. The default value of `num` is 50 and it must be a non-negative number. It is of `int` type and can be optional.
- **endpoint:** By default its value is `True`. If we take it as `False` then the value can be excluded from the sequence. It is of `bool` type and can be optional.
- **retstep:** If its `True` then it returns samples and step value where the step is the spacing between the samples.
- **dtype(data type):** It represents the type of the output array. It can also be optional.
- **axis:** The axis is the result to store the samples. It is of `int` type and can be optional.

1. Solve : $\frac{dP(t)}{dt} = r.$

```
from sympy import *
init_printing()

t,r = symbols('t,r') # Define the symbols
P = Function('P')(t) # define function
C1 = Symbol('C1')

print("\nDifferential Equation")
DE1=Derivative(P, t, 1)-r # define the differeentail equation
display(DE1)

# General solution
print("\nGeneral Solution")

GS1=dsolve(DE1) # Solve the differentail equation
display(GS1) # Display the solution

print("\nParticular Solution")
PS1=GS1.subs({C1:2}) # substitute the value of the constant
display(PS1)
```

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

```
#Solve the Differential Equation dy/dx+ytan(x)-y^3sec(x)=0
from sympy import *
x,y=symbols('x,y')
y=Function('y')(x)
y1=diff(y,x)
print("\n Differential Equation is")
DE=Eq(y1+y*tan(x)-y**3*sec(x),0)
display(DE)
print("\n General Solution is")
GS=dsolve(DE)
display(GS)
```

Differential Equation is

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

General Solution is

$$[\text{Eq}(y(x), -\sqrt{1/(C1 - 2*\sin(x))}*\cos(x)), \text{Eq}(y(x), \sqrt{1/(C1 - 2*\sin(x))}*\cos(x))]$$

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

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

3: Solve: $x^3 \frac{dy}{dx} - x^2 y + y^4 \cos x = 0$.

```
from sympy import *
x,y=symbols('x,y')
y=Function("y")(x)
y1=Derivative(y,x)
z1=dsolve(Eq(x**3*y1-x**2*y+y**4*cos(x),0),y)
display(z1)
```

In []: solve:dy/dx-y=x

```
In [1]: from sympy import*
x,y=symbols('x,y')
y=Function('y')(x)
y1=diff(y,x)
print("\n Differential Equation is")
DE=Eq(y1-y,x)
display(DE)
print("\n General Solution is")
GS=dsolve(DE)
display(GS)
```

Differential Equation is

$$-y(x) + \frac{d}{dx} y(x) = x$$

General Solution is

$$y(x) = C_1 e^x - x - 1$$

In []: solve the non-linear equation $p^2+p(x+y)+xy=0$

```
In [3]: from sympy import*
x=Symbol('x')
y=Function('y')(x)
p=diff(y,x)
eq=Eq(p**2+p*(x+y)+x*y,0)
print("\n Differential Equation is")
display(eq)
sol=dsolve(eq)
print("\n General Soluton is")
display(sol)
```

Differential Equation is

$$xy(x) + (x + y(x)) \frac{d}{dx} y(x) + \left(\frac{d}{dx} y(x) \right)^2 = 0$$

General Soluton is

[Eq(y(x), C1 - x**2/2), Eq(y(x), C1*exp(-x))]

In []: $xy(dy/dx)^2 - (x^2+y^2) dy/dx + xy = 0$

```
In [10]: from sympy import*
x,y=symbols('x,y')
y=Function('y')(x)
y1=diff(y,x)
eq=Eq(x*y*y1**2-(x**2+y**2)*y1+x*y,0)
print("\n Differential Equation is")
display(eq)
sol=dsolve(eq)
print("\n General Soluton is")
display(sol)
```

Differential Equation is

$$xy(x) \left(\frac{d}{dx} y(x) \right)^2 + xy(x) - (x^2 + y^2(x)) \frac{d}{dx} y(x) = 0$$

General Soluton is

[Eq(y(x), -sqrt(C1 + x**2)), Eq(y(x), sqrt(C1 + x**2)), Eq(y(x), C1*x)]

Exercise:

Plot the following:

- Solve $y \sin x dx - (1 + y^2 + \cos^2 x) dy = 0$.
Ans: $(1/2)y \cos 2x + (3/2)y + y^3/3 = 0$
- Solve $\frac{dy}{dx} = x + y$ subject to condition $y(0) = 2$.
Ans: $y = 3e^x - x - 1$
- Solve $\frac{dy}{dx} = x^2$ subject to condition $y(0) = 5$.
Ans: $y = x^3/3 + 5$
- Solve $x^2 y' = y \log(y) - y'$.
Ans: $y(x) = e^{C_1 \tan^{-1}(x)}$
- Solve $y' - y - x e^x = 0$.
Ans: $y(x) = \left(C_1 + \frac{x^2}{2} \right) e^x$

LAB 6: Programme to compute area, volume and center of gravity

Objectives:

1. to evaluate double integration.
2. to compute area and volume.
3. to calculate center of gravity of 2D object.

Syntax for the commands used:

1. Data pretty printer in Python:

```
pprint()
```

2. integrate:

```
integrate(function,(variable, min_limit, max_limit))
```

Double and triple integration

Example 1:

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

```
from sympy import*
x,y,z=symbols('x,y,z')
I=integrate(x**2+y**2,(y,0,x),(x,0,1))
print(I)
```

1/3

Example 2:

Evaluate the integral $\int_0^3 \int_0^{3-x} \int_0^{3-x-y} (xyz) dz dy dx$

```
from sympy import*
x,y,z=symbols('x,y,z')
I=integrate((x*y*z),(z,0,3-x-y),(y,0,3-x),(x,0,3))
print(I)
```

81/80

Example 3:

Prove that $\int \int (x^2 + y^2) dy dx = \int \int (x^2 + y^2) dx dy$

```

from sympy import *
x,y=symbols('x,y')
I1=integrate(x**2+y**2,y,x)
pprint(I1)
I2=integrate(x**2+y**2,x,y)
pprint(I2)

```

$$\frac{x^3 \cdot y}{3} + \frac{x \cdot y^3}{3}$$

$$\frac{x^3 \cdot y}{3} + \frac{x \cdot y^3}{3}$$

Area and Volume

Area of the region R in the cartesian form is $\int \int_R dx dy$

Example 4:

Find the area of an ellipse by double integration. $A=4 \int_0^a \int_0^{(b/a)\sqrt{a^2-x^2}} dy dx$

```

from sympy import *
x=Symbol('x')
y=Symbol('y')
#a=Symbol('a')
#b=Symbol('b')
a=4
b=6
w3=4*integrate(1,(y,0,(b/a)*sqrt(a**2-x**2)),(x,0,a))
print(w3)

```

24.0*pi

Area of the region R in the polar form is $\int \int_R r dr d\theta$

Example 5:

Find the area of the cardioid $r = a(1 + \cos\theta)$ by double integration

```

from sympy import *
r,t,a=symbols('r,t,a')
I=2*integrate(r,(r,0,a*(1+cos(t))),(t,0,pi))
print(I)

```

3*pi*a**2/2

Volume of a solid is given by $\int_V dx dy dz$

Example 6:

Find the volume of the tetrahedron bounded by the planes $x=0, y=0$ and $z=0, \frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$

```
from sympy import *
x=Symbol('x')
y=Symbol('y')
z=Symbol('z')
a=Symbol('a')
b=Symbol('b')
c=Symbol('c')
w2=integrate(1,(z,0,c*(1-x/a-y/b)),(y,0,b*(1-x/a)),(x,0,a))
print(w2)
```

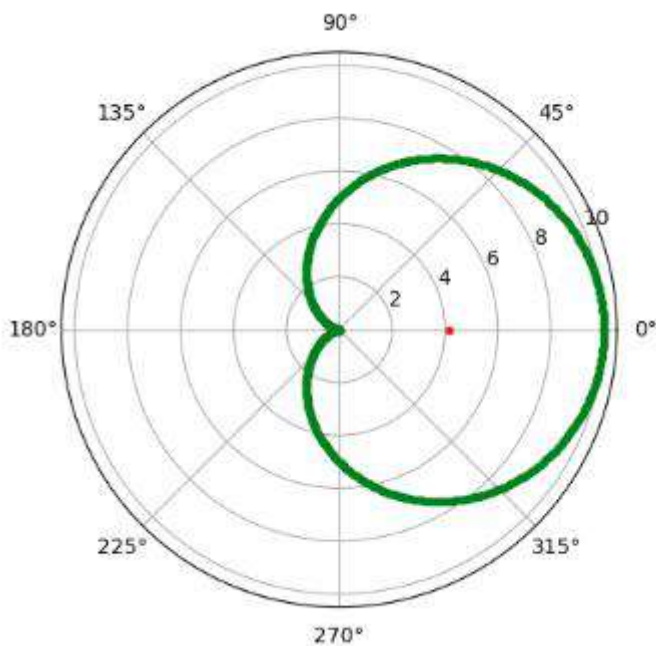
$a*b*c/6$

Center of Gravity

Find the center of gravity of cardioid . Plot the graph of cardioid and mark the center of gravity.

```
import numpy as np
import matplotlib.pyplot as plt
import math
from sympy import *
r=Symbol('r')
t=Symbol('t')
a=Symbol('a')
I1=integrate(cos(t)*r**2,(r,0,a*(1+cos(t))),(t,-pi,pi))
I2=integrate(r,(r,0,a*(1+cos(t))),(t,-pi,pi))
I=I1/I2
print(I)
I=I.subs(a,5)
plt.axes(projection='polar')
a=5
rad=np.arange(0,(2*np.pi),0.01)
for i in rad:
    r=a+(a*np.cos(i))
    plt.polar(i,r,'g.')
plt.polar(0,I,'r.')
plt.show()
```

$5*a/6$



Exercise:

1. Evaluate $\int_0^1 \int_0^x (x+y) dy dx$

Ans: 0.5

2. Find the $\int_0^{\log(2)} \int_0^x \int_0^{x+\log(y)} (e^{x+y+z}) dz dy dx$

Ans: -0.2627

3. Find the area of positive quadrant of the circle $x^2 + y^2 = 16$

Ans: 4π

4. Find the volume of the tetrahedron bounded by the planes $x=0, y=0$ and $z=0,$

$$\frac{x}{2} + \frac{y}{3} + \frac{z}{4} = 1$$

Ans: 4

LAB 7: Evaluation of improper integrals, Beta and Gamma functions

Objectives:

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:

1. gamma

```
math.gamma(x)
```

Parameters :

x : The number whose gamma value needs to be computed.

2. beta

```
math.beta(x,y)
```

Parameters :

x , y: The numbers whose beta value needs to be computed.

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

Example 1:

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

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

1

Gamma function is $\Gamma(n) = \int_0^{\infty} e^{-x} x^{n-1} dx$

Example 2:

Evaluate $\Gamma(5)$ by using definition

```
from sympy import *
x=symbols('x')
w1=integrate(exp(-x)*x**4,(x,0,float('inf')))
print(simplify(w1))
```

24

Example 3:

Evaluate $\int_0^{\infty} e^{-st} \cos(4t) dt$. That is Laplace transform of $\cos(4t)$

```
from sympy import *
t,s=symbols('t,s')
# for infinity in sympy we use oo
w1=integrate(exp(-s*t)*cos(4*t),(t,0,oo))
display(simplify(w1))
```

$$\left. \begin{array}{l} \frac{s}{s^2 + 16} \quad \text{for } |\arg(s)| < \frac{\pi}{2} \\ \int_0^{\infty} e^{-st} \cos(4t) dt \quad \text{otherwise} \end{array} \right\}$$

Example 4:

Find Beta(3,5), Gamma(5)

```
#beta and gamma functions
from sympy import beta, gamma
m=input('m :');
n=input('n :');
m=float(m);
n=float(n);
s=beta(m,n);
t=gamma(n)
print('gamma (' ,n,') is %3.3f'%t)
print('Beta (' ,m,n,') is %3.3f'%s)
```

```
m :3
n :5
gamma ( 5.0 ) is 24.000
Beta ( 3.0 5.0 ) is 0.010
```

Example 5:

Calculate Beta(5/2,7/2) and Gamma(5/2).

```
#beta and gamma functions
# If the number is a fraction give it in decimals. Eg 5/2=2.5
from sympy import beta, gamma
m=float(input('m :'));
n=float(input('n :'));

s=beta(m,n);
t=gamma(n)
print('gamma (' ,n,') is %3.3f'%t)
print('Beta (' ,m,n,') is %3.3f '%s)
```

```
m : 2.5
n :3.5
gamma ( 3.5 ) is 3.323
Beta ( 2.5 3.5 ) is 0.037
```

Example 6:

Verify that $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$ for $m=5$ and $n=7$

```
from sympy import beta, gamma
m=5;
n=7;
m=float(m);
n=float(n);
s=beta(m,n);
t=(gamma(m)*gamma(n))/gamma(m+n);
print(s,t)
if (abs(s-t)<=0.00001):
    print('beta and gamma are related')
else:
    print('given values are wrong')
```

0.000432900432900433 0.000432900432900433
beta and gamma are related

Exercise:

- Evaluate $\int_0^{\infty} e^{-t} \cos(2t) dt$
Ans: $1/5$
- Find the value of $Beta(5/2, 9/2)$
Ans: 0.0214
- Find the value of $Gamma(13)$
Ans: 479001600
- Verify that $Beta(m, n) = Gamma(m)Gamma(n)/Gamma(m + n)$ for $m=7/2$ and $n=11/2$
Ans: True

LAB 8: Numerical solution of system of equations, test for consistency and graphical representation of the solution.

Objectives:

1. to find solution of system of equations numerically.
2. to test for consistency and represent the solution graphically.

Syntax for the commands used:

1. `numpy.matrix(data, dtype = None)`

```
numpy.matrix(data, dtype = None)
```

Returns a matrix from an array-like object, or from a string of data. A matrix is a specialized 2-D array that retains its 2-D nature through operations.

2. `numpy.linalg.matrix_rank(A):`

```
numpy.linalg.matrix_rank(A)
```

Return rank of the array.

3. `numpy.shape(A):`

```
numpy.shape(A)
```

Returns the shape of an array.

4. `sympy.Matrix()`

```
sympy.Matrix()
```

Creates a matrix.

Solution of system of equations

System of homogenous linear equations:

The linear system of equations of the form $AX = 0$ is called system of homogenous linear equations. The n -tuple $(0, 0, \dots, 0)$ is a trivial solution of the system. The homogeneous system of m equations $AX = 0$ in n unknowns has a non trivial solution if and only if the rank of the matrix A is less than n . Further if $\rho(A) = r < n$, then the system possesses $(n - r)$ linearly independent solutions.

Example 1:

Check whether the following system of homogenous linear equation has non-trivial solution. $x_1 + 2x_2 - x_3 = 0$, $2x_1 + x_2 + 4x_3 = 0$, $3x_1 + 3x_2 + 4x_3 = 0$.

```
import numpy as np
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[0],[0],[0]])

r=np.linalg.matrix_rank(A)
n=A.shape[1]

if (r==n):
    print("System has trivial solution")
else:
    print("System has", n-r, "non-trivial solution(s)")
```

System has trivial solution

Example 2:

Check whether the following system of homogenous linear equation has non-trivial solution. $x_1 + 2x_2 - x_3 = 0$, $2x_1 + x_2 + 4x_3 = 0$, $x_1 - x_2 + 5x_3 = 0$.

```
import numpy as np
A=np.matrix([[1,2,-1],[2,1,4],[1,-1,5]])
B=np.matrix([[0],[0],[0]])
r=np.linalg.matrix_rank(A)
n=A.shape[1]
if (r==n):
    print("System has trivial solution")
else:
    print("System has", n-r, "non-trivial solution(s)")
```

System has 1 non-trivial solution(s)

System of Non-homogenous Linear Equations

The linear system of equations of the form $AX = B$ is called system of non-homogenous linear equations if not all elements in B are zeros.

The non homogeneous system of m equations $AX = B$ in n unknowns is

- consistent (has a solution) if and only if, $\rho(A) = \rho([A|B])$
- has unique solution, $\rho(A) = n$
- has infinitely many solutions, $\rho(A) < n$
- system is inconsistent $\rho(A) \neq \rho([A|B])$.

Example 3:

Examine the consistency of the following system of equations and solve if consistent.

$$x_1 + 2x_2 - x_3 = 1, \quad 2x_1 + x_2 + 4x_3 = 2, \quad 3x_1 + 3x_2 + 4x_3 = 1.$$

```
#Examine the consistency of the following system of equation and solve if consistent
#x1+2x2-x3=1, 2x1+x2+4x3=2, 3x1+3x2+4x3=1
```

```
import numpy as np
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B),axis=1)
rA=np.linalg.matrix_rank(A)
print("rank of matrix A",rA)
rAB=np.linalg.matrix_rank(AB)
print("rank of matrix AB",rAB)
if(rA==rAB):
    if(rA==3):
        print("The system has unique solution")
        print(np.linalg.solve(A,B))
    else:
        print("The system has infinitely many solution")
else:
    print("The system of equation is inconsistent")
```

```
rank of matrix A 3
rank of matrix AB 3
The system has unique solution
[[ 7.]
 [-4.]
 [-2.]]
```

```
#Examine the consistency of the following system of equation and solve if consistent
#x1+2x2-x3=1, 2x1+x2+5x3=2, 3x1+3x2+4x3=1
```

```
import numpy as np
A=np.matrix([[1,2,-1],[2,1,5],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B),axis=1)
rA=np.linalg.matrix_rank(A)
print("rank of matrix A",rA)
rAB=np.linalg.matrix_rank(AB)
print("rank of matrix AB",rAB)
if(rA==rAB):
    if(rA==n):
        print("The system has unique solution")
    else:
        print("The system has infinitely many solution")
else:
    print("The system of equation is inconsistent")
```

```
rank of matrix A 2
rank of matrix AB 3
The system of equation is inconsistent
```

Example 4:

Examine the consistency of the following system of equations and solve if consistent.

$$x_1 + 2x_2 - x_3 = 1, \quad 2x_1 + x_2 + 5x_3 = 2, \quad 3x_1 + 3x_2 + 4x_3 = 1.$$

#Examine the consistency of the following system of equation and solve if consistent
 $x_1+2x_2-x_3=1, 2x_1+x_2+5x_3=2, 3x_1+3x_2+4x_3=1$

```
import numpy as np
A=np.matrix([[1,2,-1],[2,1,5],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B),axis=1)
rA=np.linalg.matrix_rank(A)
print("rank of matrix A",rA)
rAB=np.linalg.matrix_rank(AB)
print("rank of matrix AB",rAB)
if(rA==rAB):
    if(rA==n):
        print("The system has unique solution")
    else:
        print("The system has infinitely many solution")
else:
    print("The system of equation is inconsistent")
```

rank of matrix A 2

rank of matrix AB 3

The system of equation is inconsistent

In []: Find the rank of coefficient matrix and augment matrix for following system.
 $x_1+2x_2-x_3=1, 2x_1+x_2+4x_3=2, 3x_1+3x_2+4x_3=1.$

```
In [2]: import numpy as np
A=np.matrix([[1,2,-1],[2,1,4],[3,3,4]])
B=np.matrix([[1],[2],[1]])
AB=np.concatenate((A,B),axis=1)
display(AB)
rA=np.linalg.matrix_rank(A)
print("rank of matrix A",rA)
rAB=np.linalg.matrix_rank(AB)
print("rank of matrix AB",rAB)
```

```
matrix([[ 1,  2, -1,  1],
        [ 2,  1,  4,  2],
        [ 3,  3,  4,  1]])
```

rank of matrix A 3

rank of matrix AB 3



Graphical representation of solution

Example 5:

Obtain the solution of $3x + 5y = 1$; $x + y = 1$ graphically.

```

from sympy import *
import numpy as np
import matplotlib.pyplot as plt

x,y=symbols('x,y')
sol=solve([3*x+5*y-1,x+y-1],[x,y])
p=sol[x]
q=sol[y]

print('Point of intersection is A (', p ,',', q, ')\n')
x = np.arange(-10, 10, 0.001)

y1 = (1-3*x)/5
y2=1-x

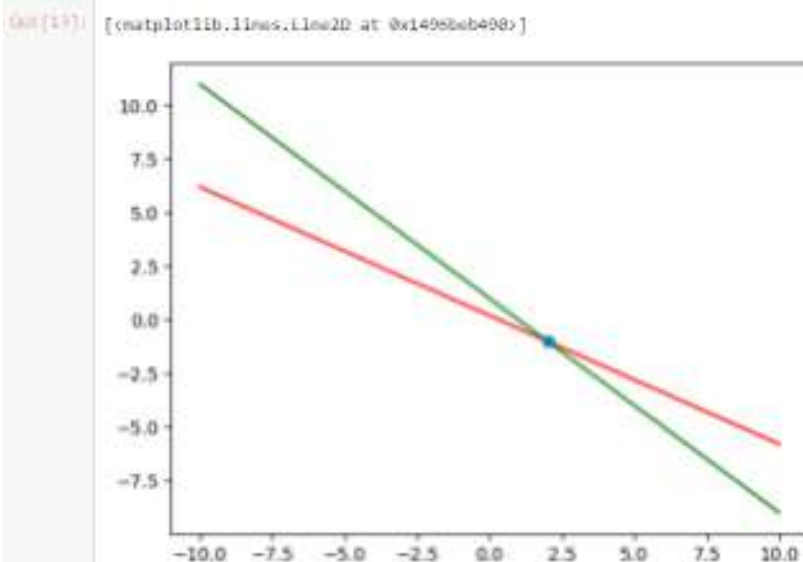
plt.plot(x,y1,x,y2)
plt.plot(p,q,marker = 'o')

plt.annotate('A', xy=(p,q), xytext=(p+0.5, q))
plt.xlim(-5,7)
plt.ylim(-7,7)
plt.axhline(y=0)
plt.axvline(x=0)
plt.title("$3x+5y=1; x+y=1$")
plt.xlabel("Values of x")
plt.ylabel("Values of y ")

plt.legend(['$3x+5y=1$', '$x+y=1$'])
plt.grid()
plt.show()

```

Point of intersection is A (2 , -1)

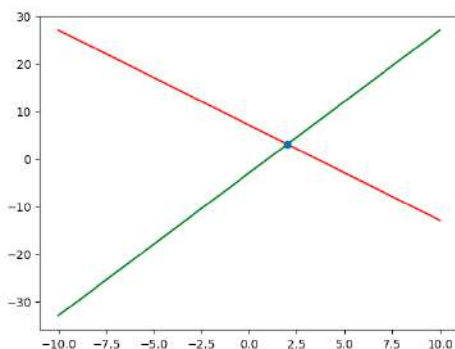


Example 6:

Obtain the solution of $2x + y = 7$; $3x - y = 3$ graphically.

```
#obain the solution of 2x+y=7;3x-y=3 graphically
from sympy import*
import numpy as np
import matplotlib.pyplot as plt
x,y=symbols('x,y')
sol=solve([2*x+y-7,3*x-y-3],[x,y])
p=sol[x]
q=sol[y]
print('point of intersection is A(',p,',',q,')\n')
x=np.arange(-10,10,0.001)
y=np.arange(-30,30,0.001)
y1=7-2*x
y2=3*x-3
plt.plot(x,y1,'r')
plt.plot(x,y2,'g')
plt.plot(p,q,marker='o')
```

point of intersection is A(2 , 3)

**Exercise:**

- Find the solution of the system homogeneous equations $x + y + z = 0$, $2x + y - 3z = 0$ and $4x - 2y - z = 0$.
Ans: The system has trivial solution.
- Find the solution of the system non-homogeneous equations $25x + y + z = 27$, $2x + 10y - 3z = 9$ and $4x - 2y - 12z = -10$.
Ans: [1, 1, 1]
- Find the solution of the system non-homogeneous equations $x + y + z = 2$, $2x + 2y - 2z = 4$ and $x - 2y - z = 5$.
Ans:[3,-1,0]
- Check whether the following system of equations are consistent.
 - $x + y + z = 2$, $2x + 2y - 2z = 6$ and $x - 2y - z = 5$.
 - $2x + y + z = 4$, $4x + 2y - 2z = 8$ and $4x + 22y + 2z = 5$.
 Ans: a. Consistent, b. Inconsistent

LAB 9: Solution of system of linear equations by Gauss-Seidel method.

Objectives:

1. to check whether the given system is diagonally dominant or not.
2. to find the solution if the system is diagonally dominant.

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

Example 1:

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

*#apply gauss seidal iteration method for the following system of equations
#20x+y-2z=17, 3x+20y-z=-18, 2x-3y+20z=25*

```
from sympy import*
from numpy import*
x=0
y=0
z=0
for i in range(8):
    print('iteration number',i+1,)
    x=(17-y+2*z)/20
    y=(-18-3*x+z)/20
    z=(25-2*x+3*y)/20
    x1=x
    x2=y
    x3=z
    print('\t,x=',round(x1,7),'\t,y=',round(x2,7),'\t,z=',round(x3,7))
```

```
iteration number 1
    ,x= 0.85      ,y= -1.0275     ,z= 1.010875
iteration number 2
    ,x= 1.0024625 ,y= -0.9998256 ,z= 0.9997799
iteration number 3
    ,x= 0.9999693 ,y= -1.0000064 ,z= 1.0000021
iteration number 4
    ,x= 1.0000005 ,y= -1.0        ,z= 1.0
iteration number 5
    ,x= 1.0       ,y= -1.0        ,z= 1.0
iteration number 6
    ,x= 1.0       ,y= -1.0        ,z= 1.0
iteration number 7
    ,x= 1.0       ,y= -1.0        ,z= 1.0
iteration number 8
    ,x= 1.0       ,y= -1.0        ,z= 1.0
```

Example 2:

Solve $x + 2y - z = 3$; $3x - y + 2z = 1$; $2x - 2y + 6z = 2$ by Gauss-Seidel Iteration method.

```
#Apply gauss seidal iteration method for the following system of equations
#x+2y-z=3,3x-y+2z=1,2x-2y+6z=2
```

```
from sympy import*
from numpy import*
x=0
y=0
z=0
for i in range(5):
    print('iteration number',i+1,)
    x=(1+y-2*z)/3
    y=(3-x+z)/2
    z=(2-2*x+2*y)/6
    x1=x
    x2=y
    x3=z
    print('\t,x=',round(x1,7),'\t,y=',round(x2,7),'\t,z=',round(x3,7))
```

```
iteration number 1
    ,x= 0.3333333    ,y= 1.3333333    ,z= 0.6666667
iteration number 2
    ,x= 0.3333333    ,y= 1.6666667    ,z= 0.7777778
iteration number 3
    ,x= 0.3703704    ,y= 1.7037037    ,z= 0.7777778
iteration number 4
    ,x= 0.382716    ,y= 1.6975309    ,z= 0.7716049
iteration number 5
    ,x= 0.3847737    ,y= 1.6934156    ,z= 0.7695473
```

```
In [ ]: Apply Gauss seidal iteration metod for the following system of equations
20x+y-2z=95,3x+20y-z=-18,2x-3y+20z=25
```

```
In [8]: from sympy import*
from sympy import*
x=0
y=0
z=0
for i in range(5):
    print('iteration number',i+1,)
    x=(95-y+2*z)/20
    y=(-18-3*x+z)/20
    z=(25-2*x+3*y)/20
    x1=x
    x2=y
    x3=z
    print('\t,x=',round(x1,3),'\t,y=',round(x2,3),'\t,z=',round(x3,3))
```

```
iteration number 1
    ,x= 4.75    ,y= -1.613    ,z= 0.533
iteration number 2
    ,x= 4.884    ,y= -1.606    ,z= 0.521
iteration number 3
    ,x= 4.882    ,y= -1.606    ,z= 0.521
iteration number 4
    ,x= 4.882    ,y= -1.606    ,z= 0.521
iteration number 5
    ,x= 4.882    ,y= -1.606    ,z= 0.521
```

Example 3:

Apply Gauss-Siedel method to solve the system of equations: $20x + y - 2z = 17$; $3x + 20y - z = -18$; $2x - 3y + 20z = 25$.

```

from numpy import *
def seidel(a, x ,b):
    #Finding length of a(3)
    n = len(a)
    # for loop for 3 times as to calculate x, y , z
    for j in range(0, n):
        # temp variable d to store b[j]
        d = b[j]

        # to calculate respective xi, yi, zi
        for i in range(0, n):
            if(j != i):
                d=d-a[j][i] * x[i]
            # updating the value of our solution
            x[j] = d / a[j][j]
        # returning our updated solution
    return x
a=array([[20.0,1.0,-2.0],[ 3.0,20.0,-1.0],[2.0,-3.0,20.0]])
x=array([[0.0],[0.0],[0.0]])
b=array([[17.0],[-18.0],[25.0]])
for i in range(0, 25):
    x = seidel(a, x, b)
print(x)

```

```

[[ 1.]
 [-1.]
 [ 1.]]

```

Exercise:

- Check whether the following system are diagonally dominant or not
 - $25x + y + z = 27$, $2x + 10y - 3z = 9$ and $4x - 2x - 12z = -10$.
 - $x + y + z = 7$, $2x + y - 3z = 3$ and $4x - 2x - z = -1$.

Ans: a. Yes b. No

- Solve the following system of equations using Gauss-Seidel Method.
 - $4x + y + z = 6$, $2x + 5y - 2z = 5$ and $x - 2x - 7z = -8$.
 - $27x + 6y - z = 85$, $6x + 15y + 2z = 72$ and $x + y + 54z = 110$

Ans: a. [1,1,1] b. [2.42, 3.57, 1.92]

LAB 10: Compute eigenvalues and corresponding eigenvectors. Find dominant and corresponding eigenvector by Rayleigh power method.

Objectives:

1. to find eigenvalues and corresponding eigenvectors.
2. to find dominant and corresponding eigenvector by Rayleigh power method.

Syntax for the commands used:

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

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

2. `np.linalg.eigvals(A)`: Computes the eigenvalues of a non-symmetric array.

3. `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

4. `lambda arguments:expression`: Anonymous function or function without a name

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- They are syntactically restricted to a single expression.
- Example: `f=lambda x : x**2 - 3*x + 1` (Mathematically $f(x) = x^2 - 3x + 1$)

5. `np.dot(vector_a, vector_b)`: Returns the dot product of vectors a and b.

Eigenvalues and Eigenvectors

Eigenvector of a matrix A is a vector represented by a matrix X such that when X is multiplied with matrix A, then the direction of the resultant matrix remains same as vector X.

Example 1:

Obtain the eigen values and eigen vectors for the given matrix.

$$\begin{bmatrix} 4 & 3 & 2 \\ 1 & 4 & 1 \\ 3 & 10 & 4 \end{bmatrix}.$$

```
import numpy as np
A=np.array([[4,3,2],[1,4,1],[3,10,4]])
print("\n given matrix :\n",A)
w,v=np.linalg.eig(A)
print("\n Eigen valus :\n",w)
print("\n Eigen vectors :\n",v)
```

Given matrix:

```
[[ 4  3  2]
 [ 1  4  1]
 [ 3 10  4]]
```

Eigen values:

```
[8.98205672  2.12891771  0.88902557]
```

Eigen vectors:

```
[[-0.49247712 -0.82039552 -0.42973429]
 [-0.26523242  0.14250681 -0.14817858]
 [-0.82892584  0.55375355  0.89071407]]
```

Eigen value:

```
8.982056720677654
```

Corresponding Eigen vector : [-0.49247712 -0.26523242 -0.82892584]

Example 2:

Obtain the eigen values and eigen vectors for the given matrix..

$$A = \begin{bmatrix} 1 & -3 & 3 \\ 3 & -5 & 3 \\ 6 & -6 & 4 \end{bmatrix}.$$

```
import numpy as np
I=np.array([[1,-3,3],[3,-5,3],[6,-6,4]])

print("\n Given matrix: \n", I)

w,v = np.linalg.eig(I)

print("\n Eigen values: \n", w)

print("\n Eigen vectors: \n", v)
```

Given matrix:

```
[[ 1 -3  3]
 [ 3 -5  3]
 [ 6 -6  4]]
```

Eigen values:

```
[ 4.+0.00000000e+00j -2.+1.10465796e-15j -2.-1.10465796e-15j]
```

Eigen vectors:

```
[[-0.40824829+0.j          0.24400118-0.40702229j  0.24400118+0.40702229j]
 [-0.40824829+0.j          -0.41621909-0.40702229j -0.41621909+0.40702229j]]
```

Largest eigenvalue and corresponding eigenvector by Rayleigh method

For a given Matrix A and a given initial eigenvector X_0 , the power method goes as follows: Consider AX_0 and take the largest number say λ_1 from the column vector and write $AX_0 = \lambda_1 X_1$. 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.

Example 4:

Compute the numerically largest eigenvalue of $P = \begin{bmatrix} 6 & -2 & 2 \\ -2 & 3 & -1 \\ 2 & -1 & 3 \end{bmatrix}$ by power 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, 1,1])
a = np.array([[6,-2,2 ],
              [-2,3,-1],[2,-1,3]])

for i in range(10):
    x = np.dot(a, x)
    lambda_1, x = normalize(x)

print('Eigenvalue:', lambda_1)
print('Eigenvector:', x)
```

Eigenvalue: 7.999988555930031
Eigenvector: [1. -0.49999785 0.50000072]

Example 5:

Compute the numerically largest eigenvalue of $P = \begin{bmatrix} 1 & 1 & 3 \\ 1 & 5 & 1 \\ 3 & 1 & 1 \end{bmatrix}$ by power 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, 1,1])
a = np.array([[1,1,3 ],
              [1,5,1],[3,1,1]])

for i in range(10):
    x = np.dot(a, x)
    lambda_1, x = normalize(x)

print('Eigenvalue:', lambda_1)
print('Eigenvector:', x)
```

Eigenvalue: 6.001465559355154
Eigenvector: [0.5003663 1. 0.5003663]

```
In [ ]: computer numerical largest eigenvalue and largest eigen vectors by power method for
A=[25 1 2
  1 3 0
  2 0 -4] by taking initial eigen vector [1,0,1]'
```

```
In [1]: import numpy as np
def normalize(x):
    fac=abs(x).max()
    x=x/x.max()
    return fac,x
x=np.array([1,0,1])
A=np.array([[25,1,2],[1,3,0],[2,0,-4]])
for i in range(8):
    x=np.dot(A,x)
    lambdal,x=normalize(x)
print('Eigenvalue:',lambdal)
print('Eigenvector:',x)
```

```
Eigenvalue: 25.182145649683974
Eigenvector: [1.          0.04508129  0.06853551]
```

Exercise:

1. Find the eigenvalues and eigenvectors of the following matrices

a. $P = \begin{bmatrix} 25 & 1 \\ 1 & 3 \end{bmatrix}$

Ans. Eigenvalues are 25.04536102 and 2.95463898; and corresponding eigenvectors are [0.99897277 -0.04531442] and [0.04531442 0.99897277].

b. $P = \begin{bmatrix} 25 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & -4 \end{bmatrix}$

Ans. Eigenvalues are 25.18215138, -4.13794129 and 2.95578991; and corresponding

eigenvectors are [0.9966522 0.06880398 0.04416339], [0.04493037 -0.00963919 -0.99894362] and [0.0683056 -0.99758363 0.01269831].

c. $P = \begin{bmatrix} 11 & 1 & 2 \\ 0 & 10 & 0 \\ 0 & 0 & 12 \end{bmatrix}$

Ans. Eigenvalues are 11., 10. and 12.; and corresponding eigenvectors are [1. -0.70710678 0.89442719], [0. 0.70710678 0.], and [0. 0. 0.4472136].

d. $P = \begin{bmatrix} 3 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 12 \end{bmatrix}$

Ans. Eigenvalues are 12.22971565, 3.39910684 and 1.37117751; and eigenvectors are [-0.11865169 -0.85311963 0.50804396], [-0.10808583 -0.49752078 -0.86069189] and [-0.98703558 0.1570349 0.03317846].

2. Find the dominant eigenvalue of the matrix $P = \begin{bmatrix} 25 & 1 & 2 \\ 1 & 3 & 0 \\ 2 & 0 & -4 \end{bmatrix}$ by power method.

Take $X_0 = (1, 0, 1)^T$.

Ans. 25.182151221680012

3. Find the dominant eigenvalue of the matrix $P = \begin{bmatrix} 6 & 1 & 2 \\ 1 & 10 & -1 \\ 2 & 1 & -4 \end{bmatrix}$ by power method.

Take $X_0 = (1, 1, 1)^T$.

Ans. 10.107545112667367

4. Find the dominant eigenvalue of the matrix $P = \begin{bmatrix} 5 & 1 & 1 \\ 1 & 3 & -1 \\ 2 & -1 & -4 \end{bmatrix}$ by power method.

Take $X_0 = (1, 0, 0)^T$.

Ans. 5.544020973078026

A₁