VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAUM

MATHEMATICS FOR CSE STREAM- 1
(Subject Code: BMATS101)

LABORATORY MANUAL - MASTER COPY

for

FIRST YEAR B.E (CSE)
SEMESTER-I

Prepared by

Dr. Shantha Kumari K
Professor, Department of Mathematics

A J INSTITUTE OF ENGINEERING & TECHNOLOGY

(A unit of Laxmi Memorial Education Trust. (R))
NH - 66, KottaraChowki, Kodical Cross - 575 006

Vision

To produce top-quality engineers who are groomed for attaining excellence in their profession and competitive enough to help in the growth of nation and global society.

Mission

- To offer affordable high-quality graduate program in engineering with value education and make the students socially responsible.
- To support and enhance the institutional environment to attain research excellence in both faculty and students and to inspire them to push the boundaries of knowledge base.
- To identify the common areas of interest amongst the individuals for the effective industry- institute partnership in a sustainable way by systematically working together.
- To promote the entrepreneurial attitude and inculcate innovative ideas among the engineering professionals.

Program Outcomes:

PO1: Engineering knowledge Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

PO2: 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.

PO3: 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.

PO4: 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.

PO5: 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.

PO6: 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.

PO7: 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.

PO8: Ethics Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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

PO11: 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.

PO12: 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.

Mathematics-I for Computer Science and Engineering stream (Effective from the academic year 2022 -2023) SEMESTER – I			
Course Code	**BMATS101**	CIE Marks	**50**
CREDITS	**04**	SEE Marks	**50**
Teaching Hours/Week(L:T:P: S)	**2:2:2:0**	Exam Hours	**3 hours**
Course Type : Integrated			
Total Hours of Pedagogy 40 hours Theory + 10 to12 Lab			

Course Objectives :

The goal of the course Mathematics-I for Computer Science and Engineering stream (22MATS11) is to

- Familiarize the importance of calculus associated with one variable and multivariable for computer science and engineering.
- Analyze Computer science and engineering problems by applying Ordinary Differential Equations.
- Apply the knowledge of modular arithmetic to computer algorithms.
- Develop the knowledge of Linear Algebra to solve the system of equations.

Module-1: Calculus (8 hours)

Introduction to polar coordinates and curvature relating to Computer Science and Engineering. Polar coordinates, Polar curves, angle between the radius vector and the tangent, angle between two curves. Pedal equations. Curvature and Radius of curvature - Cartesian, Parametric, Polar and Pedal forms. Problems. Self-study: Center and circle of curvature, evolutes and involutes. Applications: Computer graphics, Image processing.

Module-2: Series Expansion and Multivariable Calculus (8 hours)

Introduction of series expansion and partial differentiation in Computer Science & Engineering applications. Taylor's and Maclaurin's series expansion for one variable (Statement only) – problems. Indeterminate forms - L'Hospital's rule-Problems. Partial differentiation, total derivative - differentiation of composite functions. Jacobian and problems. Maxima and minima for a function of two variables. Problems.

Module-3: Ordinary Differential Equations (ODEs) of First Order (8 hours)

Introduction to first-order ordinary differential equations pertaining to the applications for Computer Science & Engineering. Linear and Bernoulli's differential equations. Exact and reducible to exact differential equations - Integrating factors on $\frac{1}{N}\left(\frac{\partial M}{\partial y}-\frac{\partial N}{\partial x}\right)$ and $\frac{1}{M}\left(\frac{\partial N}{\partial x}-\frac{\partial M}{\partial y}\right)$. Orthogonal trajectories, L-R & C-R circuits. Problems. Non-linear differential equations: Introduction to general and singular solutions, Solvable for p only, Clairaut's equations, reducible to Clairaut's equations. Problems.

Module-4: Modular Arithmetic (8 hours)

Introduction of modular arithmetic and its applications in Computer Science and Engineering. Introduction to Congruences, Linear Congruences, The Remainder theorem, Solving Polynomials, Linear Diophantine Equation, System of Linear Congruences, Euler's Theorem, Wilson Theorem and Fermat's little theorem. Applications of Congruences-RSA algorithm.

Module-5: Linear Algebra (8 hours)

Introduction of linear algebra related to Computer Science & Engineering. Elementary row transformation of a matrix, Rank of a matrix. Consistency and Solution of system of linear equations - Gauss-elimination method, Gauss-Jordan method and approximate solution by Gauss-Seidel method. Eigenvalues and Eigenvectors, Rayleigh's power method to find the dominant Eigenvalue and Eigenvector.

Course outcomes :

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

CO1	Apply the knowledge of calculus to solve problems related to polar curves
CO2	Learn the notion of partial differentiation to compute rate of change of multivariate functions
CO3	Analyze the solution of linear and nonlinear ordinary differential equations
CO4	Get acquainted and to apply modular arithmetic to computer algorithms
CO5	Make use of matrix theory for solving the system of linear equations and compute eigenvalues and eigenvectors
CO6	Familiarize with modern mathematical tool - PYTHON

Weblinks and Video Lectures (e-Resources):

1. http://nptel.ac.in/courses.php?disciplineID=111
2. http://www.class-central.com/subject/math(MOOCs)
3. http://academicearth.org/
4. VTU e-Shikshana Program
5. VTU EDUSAT Program

Assessment Details (both CIE and SEE) :

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50). The minimum passing mark for the SEE is 35% of the maximum marks (18 marks out of 50). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures not less than 35% (18 Marks out of 50) in the semester end examination(SEE), and a minimum of 40% (40 marks out of 100) in the total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation(CIE):

The CIE marks for the theory component of the IC shall be 30 marks and for the laboratory component 20 Marks. CIE for the theory component of the IC

CIE for the theory component of the IC :

Three Tests each of 20 Marks; after the completion of the syllabus of 35-40%, 65-70%, and 90- 100% respectively.

- Two Assignments/two quizzes/ seminars/one field survey and report presentation/one-course project totalling 20 marks. Total Marks scored (test + assignments) out of 80 shall be scaled down to **30 marks.**

CIE for the practical component of the IC

- On completion of every experiment/program in the laboratory, the students shall be evaluated and marks shall be awarded on the same day. The 15 marks are for conducting the experiment and preparation of the laboratory record, the other 05 marks shall be for the test conducted at the end of the semester.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to 15 marks.
- The laboratory test (**duration 03 hours**) at the end of the 15th week of the semester/after completion of all the experiments (whichever is early) shall be conducted for 50 marks and scaled down to **05 marks**.
  Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IC/IPCC for **20 marks**.
- The minimum marks to be secured in CIE to appear for SEE shall be 12 (40% of maximum marks) in the theory component and 08 (40% of maximum marks) in the practical component. The laboratory component of the IC/IPCC shall be for CIE only. However, in SEE, the questions from the laboratory component shall be included. The maximum of 05 questions is to be set from the practical component of IC/IPCC, the total marks of all questions should not be more than 25 marks. The theory component of the IC shall be for both CIE and SEE.

Semester End Examination(SEE): Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the subject (**duration 03 hours**)

- The question paper shall be set for 100 marks. The medium of the question paper shall be English/Kannada). The duration of SEE is 03 hours.
- The question paper will have 10 questions. Two questions per module. Each question is set for 20 marks. The students have to answer 5 full questions, selecting one full question from each module. The student has to answer for 100 marks and **marks scored out of 100 shall be proportionally reduced to 50 marks.**
- There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics under that module**.

Suggested Learning Resources:

Books (Title of the Book/Name of the author/Name of the publisher/Edition and Year)

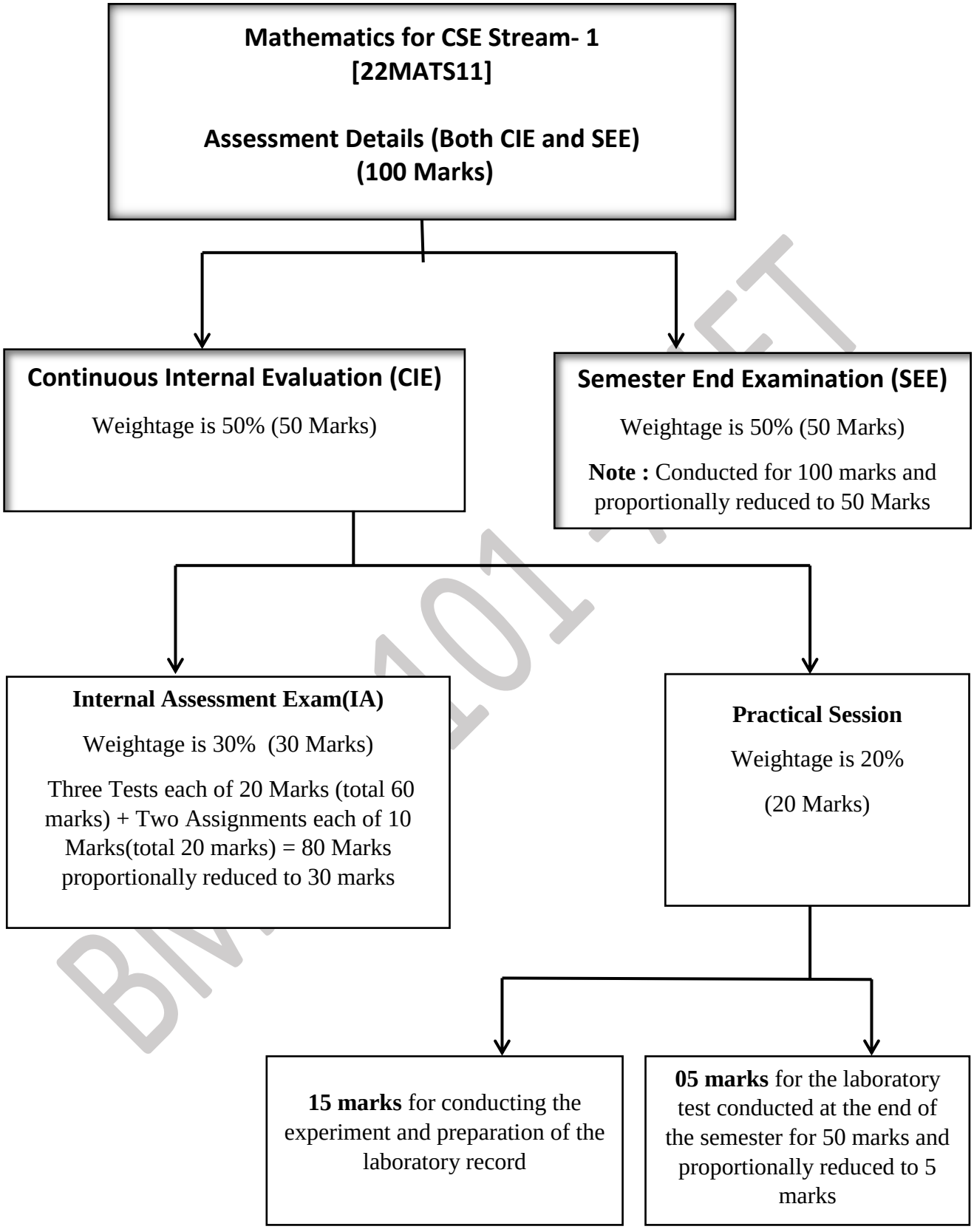
Text Books :

1. **B. S. Grewal**: "Higher Engineering Mathematics", Khanna Publishers, 44th Ed., 2021.
2. **E. Kreyszig**: "Advanced Engineering Mathematics", John Wiley & Sons, 10th Ed., 2018.
3. **David M Burton**: "Elementary Number Theory" McGraw Hill, 7th Ed.,2017.

Reference Books :

4. **V. Ramana**: "Higher Engineering Mathematics" McGraw-Hill Education, 11th Ed., 2017
5. **Srimanta Pal &Subodh C. Bhunia**: "Engineering Mathematics" Oxford University Press, 3 rd Ed., 2016.
6. **N.P Bali and Manish Goyal**: "A Textbook of Engineering Mathematics" Laxmi 26.10.2022 5 Publications, 10th Ed., 2022.
7. **C. Ray Wylie, Louis C. Barrett**: "Advanced Engineering Mathematics" McGraw – Hill Book Co., New York, 6th Ed., 2017.
8. **Gupta C.B, Sing S.R and Mukesh Kumar**: "Engineering Mathematic for Semester I and II", Mc-Graw Hill Education(India) Pvt. Ltd 2015.
9. **H. K. Dass and Er. RajnishVerma**: "Higher Engineering Mathematics" S. Chand Publication, 3rd Ed., 2014.
10. **James Stewart**: "Calculus" Cengage Publications, 7th Ed., 2019.
11. **David C Lay**: "Linear Algebra and its Applications", Pearson Publishers, 4th Ed., 2018.
12. **Gareth Williams:** "Linear Algebra with Applications", Jones Bartlett Publishers Inc., 6th Ed., 2017.
13. **Gilbert Strang:** "Linear Algebra and its Applications", Cengage Publications, 4th Ed. 2022.
14. **William Stallings**: "Cryptography and Network Security" Pearson Prentice Hall, 6th Ed., 2013.
15. **Kenneth H Rosen**: "Discrete Mathematics and its Applications" McGraw-Hill, 8th Ed. 2019.
16. **Ajay Kumar Chaudhuri**: "Introduction to Number Theory" NCBA Publications, 2nd Ed., 2009.
17. **Thomas Koshy**: "Elementary Number Theory with Applications" Harcourt Academic Press, 2 nd Ed., 2008.

Assessment Process for Mathematics for CSE- Stream – 1 Course



List of Laboratory experiments (2 hours/week per batch/ batch strength 15) 10 lab sessions + 1 repetition class + 1 Lab Assessment	
1)	2D plots for Cartesian and polar curves
2)	Finding angle between polar curves, curvature and radius of curvature of a given curve
3)	Finding partial derivatives and Jacobian
4)	Applications to Maxima and Minima of two variables
5)	Solution of first-order ordinary differential equation and plotting the solution curves
6)	Finding GCD using Euclid's Algorithm
7)	Solving linear congruences $ax \equiv b(modm)$
8)	Numerical solution of system of linear equations, test for consistency and graphical representation
9)	Solution of system of linear equations using Gauss-Seidel iteration
10)	Compute eigenvalues and eigenvectors and find the largest and smallest eigenvalue by Rayleigh power method.

Instructions to students

- Students should report to the concerned labs as per the given timetable.
- Students should make an entry in the log book whenever they enter the labs during practical.
- Students must bring Observation book, record and manual along with pen, pencil, and eraser etc., no borrowing from others.
- Students must use the computers carefully, as they are expensive. Each person may only use one computer at a time
- Any damage to the lab computers will be viewed seriously.
- Students may not install software on lab computers. If you have a question regarding specific software that you need to use, contact the concerned Faculty and Labs support team.
- When the experiment is completed, students should shut down the computers and make the counter entry in the logbook
- Wear your College ID card
- Avoid unnecessary talking while doing the experiment
- Do not panic if you do not get the output
- Students should not leave the lab without concerned faculty's permission.
- Before leaving the lab, students should check whether they have switched off the computers and the power supplies and kept their chairs properly.
- In each Lab student have to show the record of previous Lab.
- Each Lab will be evaluated for 15 marks and finally average will be taken for 15 marks.
- Viva questions shall be asked in labs and attendance also can be considered for everyday Lab evaluation.
- Tests shall be considered for 5 marks and final Lab assessment is for 20 marks.
- Student has to score minimum 8 marks out of 20 to pass Lab component.

Rules for Maintaining Record

- Write your name, USN and name of the subject on the outside front cover of the record. Write the same information in the first page inside the cover page.
- Update Table of Contents every time you start each new experiment or topic.
- Always use pen and write neatly and clearly
- Start each new topic (experiment number, experiment name, etc.) on a right-side (odd numbered) page.
- Obvious care should be taken to make it readable, even if you have bad handwriting
- Date to be written on every page on the top right-side corner
- Attach printouts and plots of data as needed.
- Strictly observe the instructions given by the Teacher/ Lab Instructor.

INTRODUCTION

WHAT IS PYTHON?

Python is a computer programming language. It is a high-level general purpose scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Beginner-level language that supports development of wide range of applications.

History of Python

- Python was developed by ***Guido van Rossum*** in the ***late eighties*** and ***early nineties*** at the ***National Research Institute*** for Mathematics and Computer Science in the ***Netherlands***. Guido van Rossum (1987) named it after the BBC television show 'Monty Python's Flying Circus.' (Comedy show).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

PYTHON USED FOR...

- Web and Internet Development
- Desktop GUI Applications
- Scientific and Numeric
- Software Development
- Education
- Business Applications
- Games and 3D Graphics
- Network programming
- Database Access
- One of the recent growing field of expertise is 'data science'. Many data scientists use

Python for their day-to-day work.

PYTHON'S MAJOR FEATURES

Python's features include −

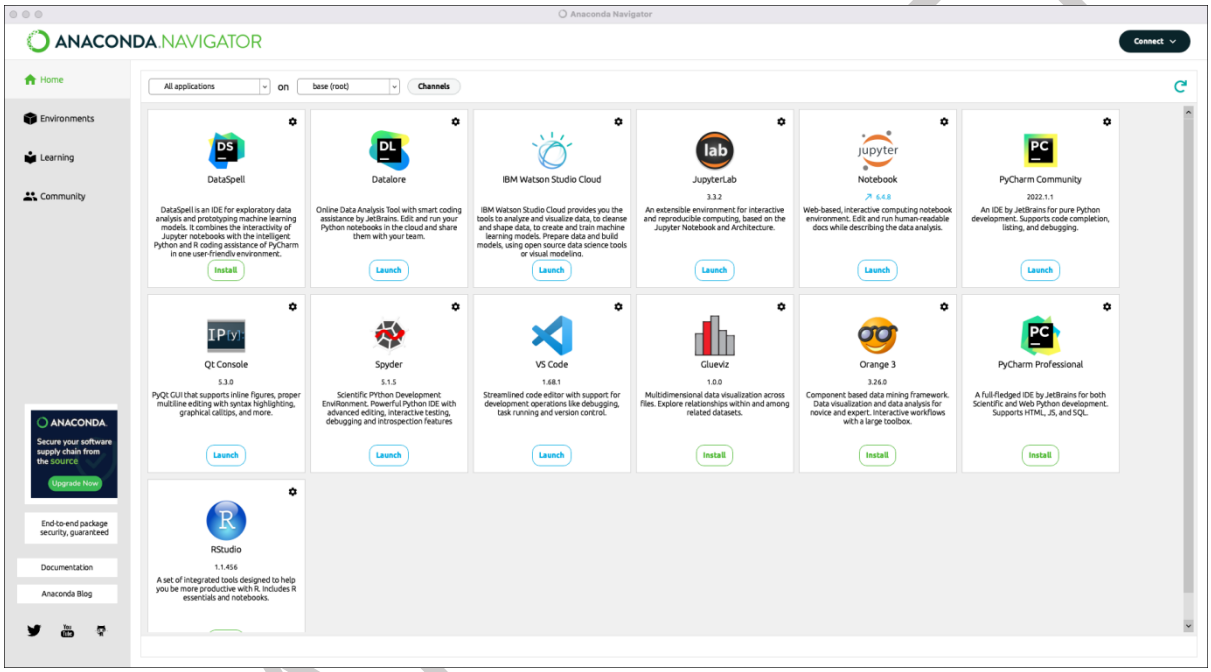
- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- Python is portable: it runs on many Unix variants including Linux and macOS, and on Windows.
- Python includes a comprehensive base library.

INTRODUCTION TO ANACONDA

Anaconda is a package manager, environment manager, and Python distribution with a collection of 1,500+ open source packages with free community support. Anaconda is free and easy to install and can be used on Windows, macOS, or Linux. Anaconda can be downloaded from https://www.anaconda.com/products/individual

After installing Anaconda, we use Anaconda Navigator to launch applications and easily manage packages, environments and channels without using command-line commands.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in the terminal window.



APPLICATIONS IN NAVIGATOR

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Glueviz
- Orange 3 App
- RStudio
- Anaconda Prompt (Windows only)
- Anaconda PowerShell (Windows only)

JUPYTER NOTEBOOK :

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the wholecomputation process: developing, documenting, and executing code, as well as communicating the results.

Keywords and Identifiers :

Keywords are the reserved words in Python used by Python interpreter to recognize the structure of the program. **Identifiers** are the name given to entities like class, functions, variables etc. It helps to differentiate one entity from another.Identifiers can be a combination of letters from "a" to "z", from "A" to "Z" and digits from "0" to "9" or special character _.

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language.
- Keywords cannot be used as identifiers.
- Only special character used in identifiers is _.
- Identifier can be of any length.

Reserved Words

- The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	**exec**	**Not**
assert	**finally**	**or**
break	**for**	**pass**
class	**from**	**print**
continue	**global**	**raise**
def	**If**	**return**
del	**import**	**try**
elif	**in**	**while**
else	**is**	**with**
except	**lambda**	**yield**
False	**True**	**None**

DATA TYPES

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type : **str**

Numeric Types **: int, float, complex**

Sequence Types **: list, tuple, range**

Mapping Type **: dict**

Set Types **: set, frozenset**

Boolean Type **: bool**

Binary Types **: bytes, bytearray, memoryview**

Quotation in python

Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
# First comment

print ("Hello, Python!")# second comment
```

This produces the following result −

```
Hello, Python!
```

You can comment multiple lines as follows −

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

PYTHON - VARIABLE TYPES

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example −

```
counter=100   # An integer assignment
miles=1000.0   # A floating point
name="John"   # A string
print (counter)
print (miles)
print (name)
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result −

```
100
1000.0
John
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example −

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

EXAMPLES :

a=2.4

x=3

y = ’a ’

z=" Hello "

print (type (a)) #type ( ) gives the data type of the v a r i a b l e .

print (type (x))

print (type (y))

print (type (z))

EXAMPLE FOR NUMERIC TYPES:

x = 1 # int

y = 2.8 # f l o a t

z = 1 j # complex

X = 35e3 # e i n d i c a t e s the power of 10

Y = 12E4

Z = −87.7e100

OPERATORS

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

All the common algebraic operators presented in the following table are available in Python.

Addition : +

Subtraction : −

Multiplication : *

Division : /

Integer Division : //

Power: **

Remainder: %

Comparison operators :

< less than

> greater than

<= less than or equal to

>= greater than or equal to

== equal to

! = not equal to

Logical operators:

and logical and

or logical or

not logical not

ASSIGNMENT OPERATOR

= is used as assignment operator.

For example x = 5, means 5 is stored to variable x.

+ = The expression a+ = 3, give the result of a = a + 3.

− = The expression x− = 3, give the result of x = x − 3.

∗ = The expression a∗ = 3, give the result of a = a ∗ 3.

/ = The expression a/ = 3, give the result of a = a/3.

// = The expression a// = 3, give the result of a = a//3.

% = The expression a% = 3, give the result of a = a%3.

∗∗ = The expression a∗ = 3, give the result of a = a ∗∗3.

IDENTITY OPERATORS

is Returns True if both variables are the same object

is not Returns True if both variables are not the same object

MEMBERSHIP OPERATORS

in Returns True if a sequence with the specified value is present in the object

not in Returns True if a sequence with the specified value is not present in the object

PYTHON LISTS

Python Lists are used to group related data together. They are collection of data items. Lists are used to store multiple items in a single variable.We enclose the items with in square brackets[ ]and separate them by commas.
Lists are created using square brackets: for example
a=[1,2,3]

```
Ex: list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

PYTHON TUPLES

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
Creating a tuple is as simple as putting different comma-separated values.

Ex:

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

Tuples are created using round brackets: for example

a=(1,2,3)

items are ordered, allow duplicate values.

List items are changeable, whereas; Tuple items are unchangeable.

INPUT STATEMENT

The input() function allows user input.

SYNTAX:

input(prompt)

prompt– A String, representing a default message before the input.

For example:

```
x = input ( ' Enter your name : ' )
print ( ' Hello , ' +x )
print ( ' Hello , ' , x )
print ( type ( x ) )
```

To read integer then the following to be used.

x = int (input (' Enter an integer : '))

print (type (x))

To read float then the following to be used.

x = float (input ('Enter an float : '))

print (type (x))

OUTPUT STATEMENTS

Python provides the **print()** function to display output to the standard output devices.

SYNTAX:

print(value(s), sep= ' ', end = '\n', file=file, flush=flush)

value(s) – Any value, and as many as you like. Will be converted to string before printed

sep='separator' – (Optional) Specify how to separate the objects, if there is more than one. Default :' '

end='end' – (Optional) Specify what to print at the end. Default : '\n'

file – (Optional) An object with a write method. Default : sys.stdout

flush – (Optional) A Boolean, specifying if the output is flushed (True) or buffered (False). Default: False

Programming Structures :

Conditional structure

What is conditioning in Python?

• Based on certain conditions, the flow of execution of the program is determined using proper syntax.

How to use if conditions?

- if statement — for implementing one-way branching
- if-else statements —for implementing two-way branching
- nested if statements —for implementing multiple branching
- if-elifladder — for implementing multiple branching

if statement :

```
# Syntax:
If condition :
statements
```

Example: #Check if the given number is positive

```
a=int (input (" Enter an integer: "))
if a>0 :
print(" 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 .
```

.

LAB 1: 2D plots of Cartesian and polar curves

1.1 Objectives:
Use python
1. to plot Cartesian curves.
2. to plot polar curves.
3. to plot implicit functions.

Syntax for the commands used:

1. 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)

2. 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)

3. 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)
4. Return evenly spaced values within a given interval. arangecan be called with a varying number of positional arguments.
numpy .arange([start ,]stop , [step ,] dtype=None , *, like = None)
https://matplotlib.org/stable/api/pyplot_summary.html#modulematplotlib.pyplot

1.2 Example: Plotting points (Scattered plot)

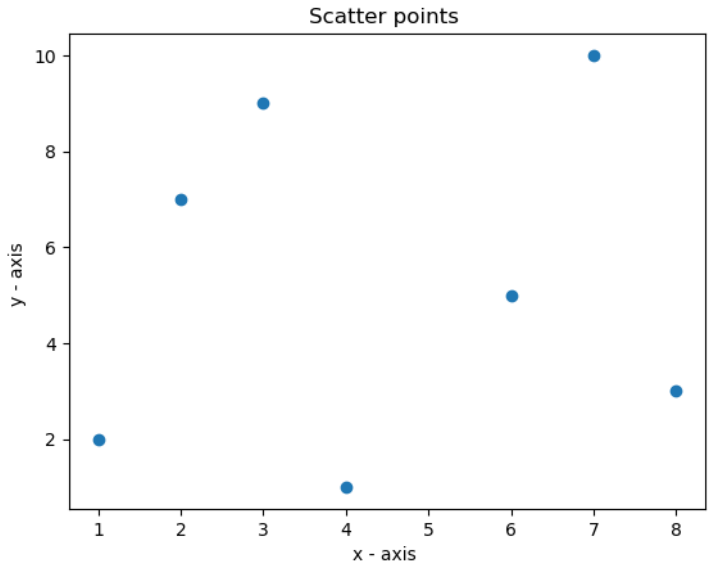
```
#Syntax
# importing the required module
#x axis values
# corresponding y axis values
# plotting the points
# naming the x axis
# naming the y axis
# giving a title to my graph
# function to show the plot
```

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 :

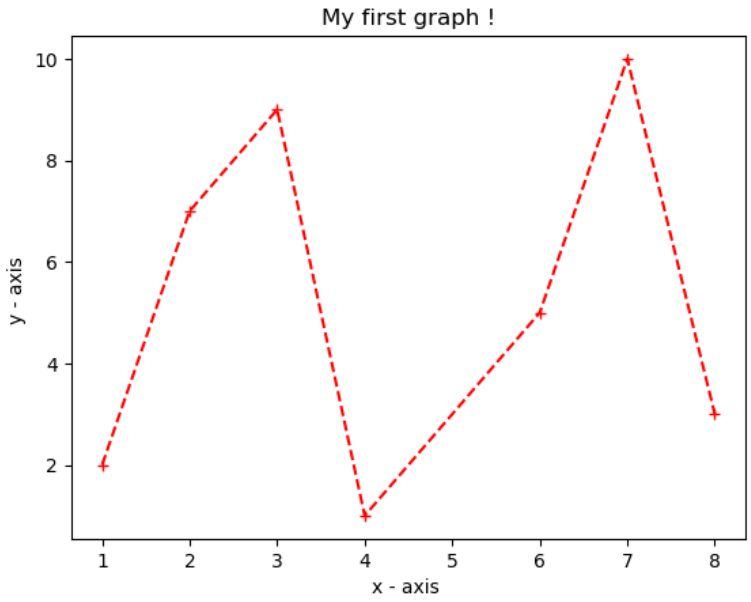


Example: Plotting a line(Dashed Lines)

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
```

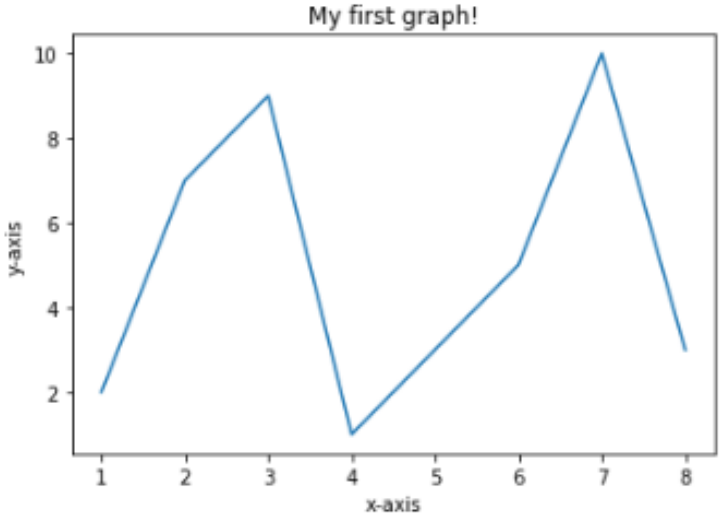
Output :



Example: Plotting a line(Line plot)

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

Output :

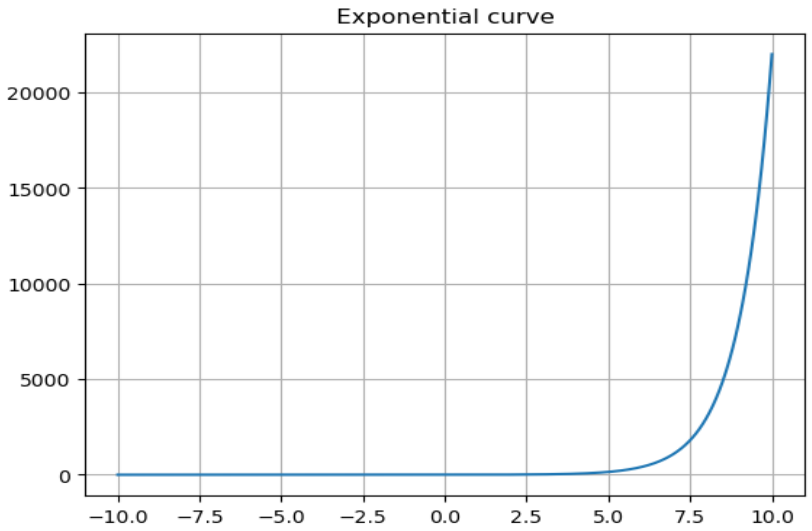


Example : Plotting the Exponential curve $y = e^x$

Code :

```
# importing the required modules
import numpy as np
import matplotlib . pyplot as plt
x = np. arange (-10 , 10 , 0.001)        # x takes the values between -10 and 10
with a step length of 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 :

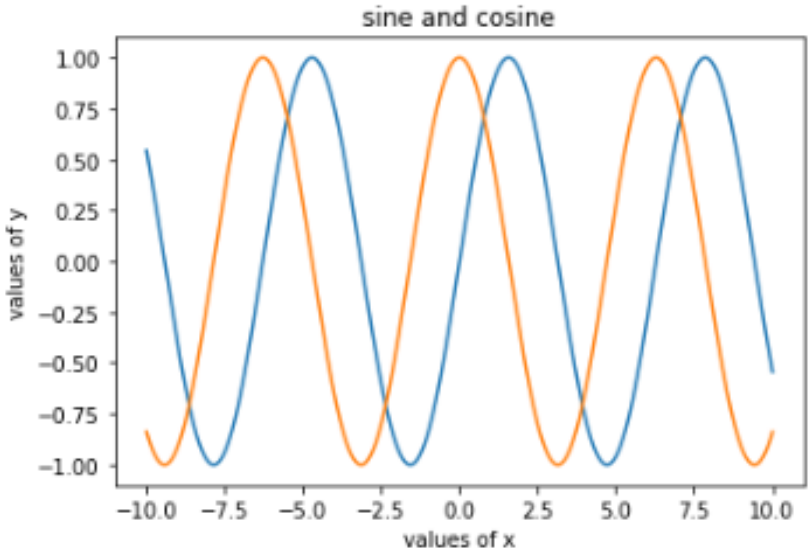


Example : Plotting Sine and Cosine curves

Code :

```
import numpy as np                  # if we are calling numpy no need to call math
import matplotlib.pyplot as plt
x=np.arange(-10,10,0.001)
                    # x takes the values between -10 and 10 with step length 0.001
y1=np.sin(x)
y2=np.cos(x)
plt.plot(x,y1,x,y2)                 # ploting the points
plt.title("sine and cosine")          # Title
plt.xlabel('values of x')           # naming x axis
plt.ylabel('values of y')           # naming y axis
plt.show()                          # shows the plot
```

Output :



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)

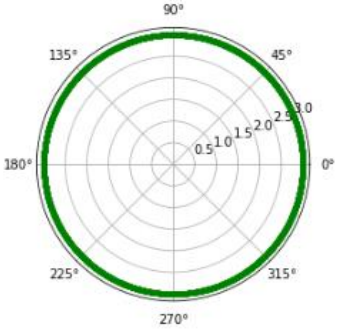
- Theta: This is the angle at which we want to draw the curve.
- r: It is the distance.
- kwargs means key word arguments

Example : Plot a Circle: r = p,where p is the radius of the circle

Code :

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

output :



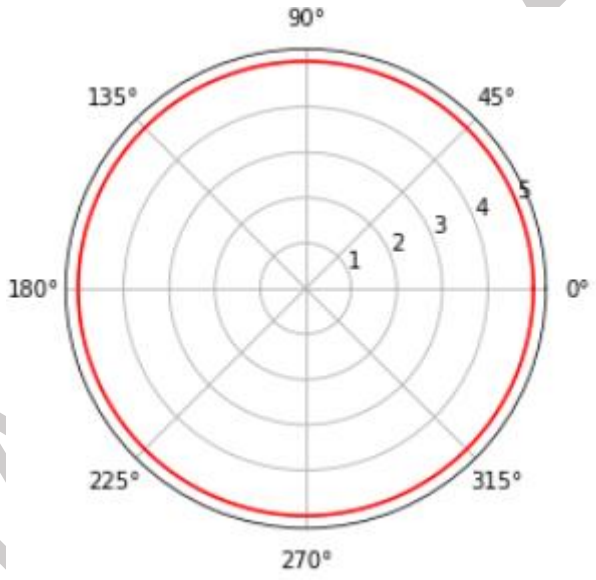
Example : Plot a Circle $x^2 + y^2 = r$, Using **linespace** where r is the radius of the circle.

Here we use **PyLab** which is a convenience module that bulk imports **matplotlib.pyplot** (for plotting) and **NumPy** (for Mathematics and working with arrays) in a single name space
The **linspace** function produces an array or matrix filled with a requested number of numbers starting at a given value and ending at a given value.

Code :

```
from pylab import *
theta=linspace(0,2*numpy.pi, 100000)
x=5*cos(theta)                  # radius = 5
y=5*sin(theta)
r= sqrt(x**2+y**2)
polar (theta, r, 'r-')
show
```

Output :

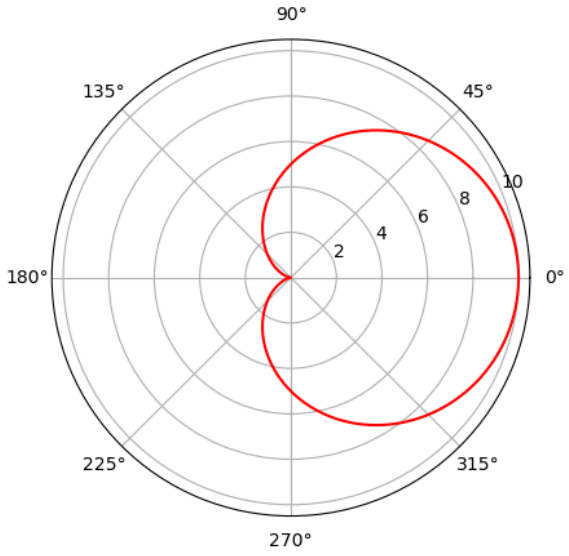


Example : Plot the Cardioid: r = 5(1 + cosθ)

Code :

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

Output :

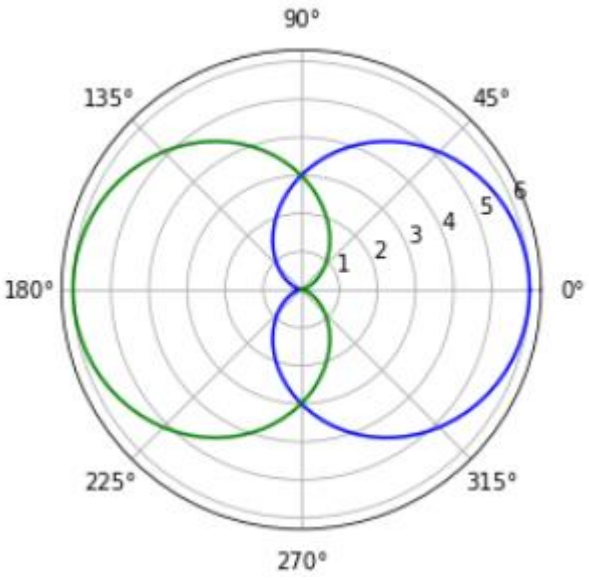


Example : Plot the Cardioids: r = a(1 + cosθ) and r = a(1 + cosθ), where a=3

Code :

```
from pylab import *
theta = linspace (0,2*np.pi , 100)
a=3
r=a + a*cos(theta)
r1=a - a*cos ( theta )
polar (theta , r ,'b')
polar (theta , r1 ,'g')
show ( )
```

output :

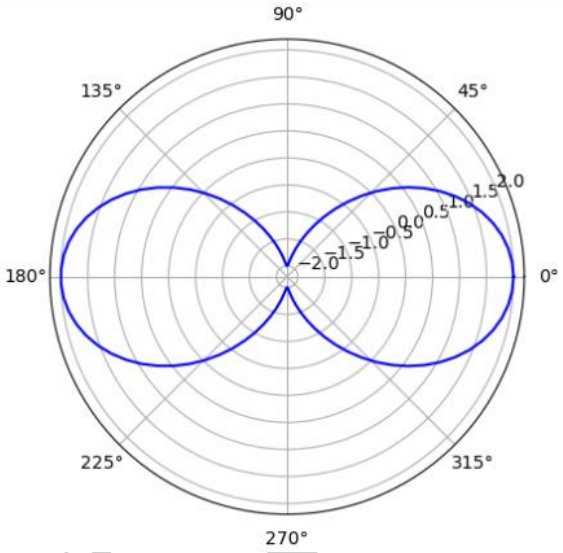


Example : Plot the curve r = 2(cos 2θ)

Code :

```
from pylab import *
theta = linspace (0,2*np.pi , 100)
a=2
r=a*cos(2*theta)
polar (theta , r ,'b')
show ( )
```

output :



Practice problems :

1. **Plot the curve r = 2(cos 4θ)**
2. **Plot the curve r = 2(cos 6θ)**
3. **Plot the curve r = a(sin 2θ), with a=3**
4. **Plot the curve r = a(sin 3θ), with a=3**

Viva Questions :

1. Which is the 2D plotting Library in Python?
   Answer : Matplotlib
2. What is Matplotlib?
   Answer : Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of formats.
3. Which character is used in Python to make a single line comment?
   Answer : : #
4. The power operator in python is
   Answer : **

5. Which module in python contains a function polar( )
   Answer : matplotlib.pyplot
6. What is the use of linspace function in python?
   Answer : linspace is an in-built function in Python's NumPy library. It is used to create an evenly spaced sequence in a specified interval.
7. Name the package that combines NumPy, and Matplotlib into a single namespace
8. **Answer :** Pylab
9. Which function in pyplot module of matplotlib library is used to display all figures.
   Answer : The show() function
10. What is the use of grid() function?
    Answer : The **grid()** function in the **Pyplot** module of the **Matplotlib** library is used to configure the grid lines

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

2.1 Objectives:

Use python

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

Syntax for the commands used:

1. diff( )

diff(function , variable)

2. 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.

3. doit( )

doit (x)

4. Return : evaluated object

5. simplify( )

simplify (expression)

6. expression– It is the mathematical expression which needs to be simplified.

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

8. display( )

display (expression)

9. expression– It is the mathematical expression which needs to be simplified.

10. Returns: Displays the expression.

11. syntax of Substitute : subs

( ) **math expression . subs (variable , substitute)**

12. variable – It is the variable or expression which will be substituted.

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

14. Returns: Returns the expression after the substitution.

2.2 1. Angle between two polar curves

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

If tan ϕ1 and tan ϕ2 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.

```
#Syntax
# Define the variables required as symbols
# Input first polar curve
# Input first polar curve
# Find the derivative of first function
# Find the derivative of second function
# solve r1 == r2, to find the point of intersection between curves
# Substitute the value of "t" in t2
# substitute the value of "t" in t1
# To find the inverse tan of w1
# To find the inverse tan of w2
# Angle between two curves is abs (w1 -w2)
```

Example : Find the angle between the curves r = a(1 + cos theta) and r = b(1 − cos theta) where a=4 and b=5

Code :

```
from sympy import *
import math
r, theta = symbols ('r,theta')    # Define the variables required as symbols
r1=4*(1+cos (theta));             # Input first polar curve
r2=5*(1-cos (theta));             # Input first polar curve
dr1 = diff (r1 ,theta)            # find the derivative of first function
dr2 = diff (r2 ,theta)            # find the derivative of secodn function
tanphi1=r1/dr1
tanphi2=r2/dr2
q= solve (r1-r2,theta)            # solve r1= r2, to find the point of intersection between curves
w1=tanphi1. subs ({theta: float (q[1])})      # substitute the value of "theta" in tanphi1
w2=tanphi2. subs ({theta: float (q[1])})      # substitute the value of "theta" in tanphi2
phi1= atan (w1)                               # to find the inverse tan of w1
phi2= atan (w2)                               # to find the inverse tan of w2
w=abs(phi1-phi2)                  # angle between two curves is abs(phi1-phi2)=|phi1-phi2|
print ('Angle between curves in radians is %0.3f ' %(w))
print('Angle between curves in degrees is', math. degrees(w))
```

Example : Find the angle between the curves r = a log(theta) and r= a/log(theta) where a=4

Code :

```
from sympy import *
import math                       #  we can also use import math as mt
r, theta = symbols ('r,theta')    # Define the variables required as symbols
r1=4* log(theta);                 # Input first polar curve
```

```
r2=4/ log(theta);                    # Input first polar curve
dr1 = diff (r1 ,theta)               # find the derivative of first function
dr2 = diff (r2 ,theta)               # find the derivative of second function
tanphi1=r1/dr1
tanphi2=r2/dr2
q= solve (r1-r2,theta)               #  to find the value of theta
w1=tanphi1. subs ({theta: float (q[1])})     # substitute the value of "theta" in tanphi1
w2=tanphi2. subs ({theta: float (q[1])})     # substitute the value of "theta" in tanphi2
phi1= atan (w1)                       # to find the inverse tan of w1
phi2= atan (w2)                       # to find the inverse tan of w2
angle=abs(phi1-phi2)          # angle between two curves is abs(phi1-phi2)=|phi1-phi2|
print ('Angle between curves in radians is %0.3f ' %(angle))
print('Angle between curves in degrees is %0.3f' %(math. degrees(angle)))
```

Output:

Angle between curves in radians is 0.705
Angle between curves in degrees is 40.395

Practice Problems :

1. Find the angle between the curves r = 4(1 + cost) and r = 5(1 − cost).
2. Find the angle between the curves r = 4 cost and r = 5 sin t.

2.3 2. Radius of curvature

Formula to calculate Radius of curvature in polar form is $\rho = \frac{(r^2+r_1^2)^{\frac{3}{2}}}{r^2+2r_1^2-rr_2}$

```
#Syntax
# Define the variables required as symbols
# Input first polar curve
# Input first polar curve
# Find the first derivative of r w.r.t "t"
# Find the second derivative of r w.r.t "t"
# Substitute r1 and r2 in formula
# Substitute t in rho
```

Example : Find the radius of curvature, r = 4(1 + cos t) at t=π/2.

Code :

```
from sympy import *
theta= Symbol ('theta') # define t as symbol
r= Symbol ('r')
r=4*(1+cos(theta))
r1 = diff (r ,theta)
```

```
r2 = diff (r1 ,theta)
rho =(r**2+r1**2)**(1.5)/(r**2+2*r1**2-r*r2);
rho1 = rho. subs (theta, pi/2) # substitute the theta=pi/2  in rho
print ('The radius of curvature is %3.4f units' %(rho1) )
display (simplify (rho ))
```

output :

The radius of curvature is 3.7712 units

$$3.77123616632825(\cos(\theta)+1)^{0.5}$$

Example : Find the radius of curvature for r = asin(nt) at t = pi/2 and n = 1.

Code :

```
from sympy import *
theta,r,a,n= symbols ('theta, r, a, n')
r=a*sin(n*theta)
r1 = diff (r ,theta)                    # or  r1= Derivative (r,t). doit ()
r2 = diff (r1 ,theta)                   # r2= Derivative (r1 ,t). doit ()
rho = (r ** 2+r1 ** 2) ** 1.5/(r ** 2+2*r1 ** 2-r*r2);
rho = (r ** 2+r1 ** 2) ** 1.5/(r ** 2+2*r1 ** 2-r*r2);
rho1 = rho . subs (theta, pi/2)
rho2 = rho1 . subs (n,1)
print ("The radius of curvature is", rho2)
display (simplify (rho2 ))
```

output :

```
The radius of curvature is (a**2)**1.5/(2*a**2)
```

$$\frac{(a^2)^{1.5}}{2a^2}$$

Practice Problems:

1. Find the radius of curvature, r = 4(1 + cost) at t=π/2.
2. Find the radius of curvature for r = $a\sin(nt)$ at t = pi/2 and n = 1.

Viva Questions :

1. What is the use of %0.3f in print statement?
   Answer : We use %0.3f as format placeholders, Using this we are able to contain the number of decimal places to 3. "f" stands for floating point. The integer (here 3) represents the number of decimals after the point.

2. What is the use of Sympy in python?
   Answer : SymPy is an open-source Python library **for symbolic computation. We can use** symbols **to solve equations.**
3. What is the expression for angle between two polar curves?
   Answer : $|\boldsymbol{\phi_1 - \phi_2}|$ **where** $\boldsymbol{\phi_1 = tan^{-1}\left(\frac{r_1}{r_1'}\right)}$ **and** $\boldsymbol{\phi_1 = tan^{-1}\left(\frac{r_2}{r_2'}\right)}$
4. What is the expression for the radius of curvature in polar form.?
   Answer : $\boldsymbol{\rho = \frac{(r^2+r_1^2)^{\frac{3}{2}}}{r^2+2r_1^2-rr_2}}$
5. What is the use of simplify() function?
   Answer : simplify() is there that attempts to arrive at the simplest form of mathematical expressions

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

3.1 Objectives:

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:

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 variables then it differentiates the function partially w.r.t variable.

If u = u(x, y) then,

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

```
#Syntax
# Define the variables required as symbols
# input mutivariable function u=u(x,y)
# Differentate u w.r.t x
# Differentate u w.r.t. y
#uxy = diff (ux,y)
#uyx = diff (uy,x)
# Check the condition uxy=uyx
# If condition 1 is True - Statements 1 will be executed .
# else if condition 2 is True - Statements 2 will be executed and so on
```

Example : Prove that mixed partial derivatives, uxy= uyx for
u = exp(x)(xcos(y) ysin(y)).

Code :

```
from sympy import *
x,y = symbols ('x y')
u=exp(x)*(x*cos(y)-y*sin(y)) # input mutivariable function u=u(x,y)
ux = diff (u,x) # Differentate u w.r.t x
uy = diff (u,y) # Differentate u w.r.t. y
uxy = diff (ux ,y) # or duxy = diff (u,x,y)
uyx = diff (uy ,x) # or duyx = diff (u,y,x)
# Check the condtion uxy=uyx
if uxy == uyx :
    print ('Mixed partial derivatives are equal ')
else :
   print ('Mixed partial derivatives are not equal ')
```

Output: Mixed partial derivatives are equal

Example : Prove that if u = exp(x)(x cos(y) − y sin(y)) then uxx+ uyy= 0.

```
#Syntax
# Define the variables required as symbols
# input mutivariable function u=u(x,y)
# Differentate u w.r.t x
# Differentate u w.r.t. y
# uxy = diff (ux,y)
#uyx = diff (uy,x)
# Add uxx and uyy
# Simply the w to get actual result
```

Code :

```
from sympy import *
x,y = symbols ('x y')
u=exp(x)*(x*cos(y)-y*sin(y))
print('The value of u   is u=')
display (u)
ux = diff (u,x)
uy = diff (u,y)
uxx = diff (ux ,x)                  # or uxx= diff (u,x,x) second derivative of u w.r.t x
uyy = diff (uy ,y)                  # or uyy= diff (u,y,y) second derivative of u w.r.t y
w=uxx+uyy                           # Add uxx and uyy
print('The value of w   is w=')
display(w)
w1= simplify (w)                    # Simply the w to get actual result
print ('Ans :',float (w1))
```

Output:

```
The value of u   is u=
```

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

```
The value of w   is w=
```

$$(x\cos(y) - y\sin(y))\,e^x + (-x\cos(y) + y\sin(y) - 2\cos(y))\,e^x + 2e^x\cos(y)$$

```
Ans : 0.0
```

Jacobians :

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

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

```
#Syntax
# Define the variables required as symbols
# Input multivariable function
# Find the all first order partial derivates
# Construct the Jacobian matrix
#Find the determinant of Jacobian Matrix
```

Example : If u = xy/z, v = yz/x,w= 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
# find the all first order partial derivates
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 )
M= Matrix ([[ux , uy , uz],[vx , vy , vz],[wx , wy , wz]]) ;      # construct the Jacobian matrix
print ("The Jacobian matrix is \n")
```

```
display ( M )
J =det ( M )                                  # Find the determinat of Jacobian Matrix

print ('\n\n J = ', J)
```

Output:

The Jacobian matrix is

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

J = 4

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

```
#Syntax
# Define the variables required as symbols
# Input multivariable function
# Find the all first order partial derivates
# Construct the Jacobian matrix
# Find the determinant of Jacobian Matrix
```

Code :

```
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
# find the all first order partial derivates
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 )
M= Matrix ([[ux , uy , uz],[vx , vy , vz],[wx , wy , wz]]) ;   # construct the Jacobian matrix
print ("The Jacobian matrix is M= \n")
display ( M )
J =det ( M )                                  # Find the determinat of Jacobian Matrix
print ("The Jacobian   is J= \n", )
display(J)
J1=J . subs ([(x , 1 ) , (y , -1 ) , (z , 0 )])
print ("Ans :The Jacobian at (1, −1, 0)  is J1=\n", float(J1) )
```

Output :

```
The Jacobian matrix is M=
```

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

```
The Jacobian   is J=
```

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

```
Ans :The Jacobian at (1, -1, 0)  is J1=
 20.0
```

Viva Questions :

1. If $f(x,y) = x^3 + y^3$, then value of $\frac{\partial f}{\partial x}$
   Answer : $3x^2$
2. Expression for Jacobian of a transformation $\boldsymbol{u = f(x,y)}$, $\boldsymbol{v = g(x,y)}$ **is**
   Answer : $\boldsymbol{J = \begin{bmatrix} u_x & u_y \\ v_x & v_y \end{bmatrix}}$
3. Jacobian is always non-zero : True or False?
   Answer : False
4. If u and v are functions of the two independent variables x and y , and are not independent then the Jacobian is ------
   Answer : 0,

LAB 4 : Applications of Maxima and Minima of functions of two variables

4.1 Objectives:

Use python

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

Returns the limit of the mathematical expression under given conditions.

Example: Find the Maxima and minima of $f(x,y) = x^3 + 3xy^2 - 15x^2 - 15y^2 + 72x$.

Code :

```
import sympy
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 (a,b)=", sol)
fxx=diff(fx, x)
```

```
fxy= diff(fx,y)
fyy=diff(fy, y)
for x1, y1 in sol :                    # Here (x1,y1) is a stationary point
   r= fxx.subs({x:x1, y:y1})
   s= fxy.subs({x:x1, y:y1})
   t= fyy.subs({x:x1, y:y1})
   delta= r*t -s*2
   if (delta>0 and r>0):
      print("The function has minimum at (",x1,",",y1,")")
      q1=f.subs({x:x1, y:y1})
      print("Minimum value:", q1)
   elif (delta>0 and r<0):
      print("The function has maximum at (",x1,",",y1,")")
      q2=f.subs({x:x1, y:y1})
      print("Minimum value:", q2)
   elif(delta<0):
      print("(",x1,",",y1,") is a saddle point")
   elif(delta==0):
      print("it needs further investigations")
```

Output :

```
The critical points are (a,b)= [(4, 0), (5, -1), (5, 1), (6, 0)]
The function has maximum at ( 4 , 0 )
Minimum value: 112
( 5 , 1 ) is a saddle point
The function has minimum at ( 6 , 0 )
Minimum value: 108
```

Example: Find the Maxima and minima of $f(x,y) = 2(x^2 - y^2) - x^4 + y^4$

Code :

```
import sympy
from sympy import *
x,y= symbols('x,y')
f=2*(x**2-y**2)-x**4+y**4
fx=diff(f, x)
fy=diff(f,y)
sol=solve([Eq(fx,0), Eq(fy,0)], (x,y))
print ("The critical points are (a,b)=", sol)
fxx= diff(fx, x)
fxy= diff(fx,y)
fyy=diff(fy, y)
for x1, y1 in sol:     # Here (x1,y1) is a stationary point
   r= fxx.subs({x:x1, y:y1})
   s= fxy.subs({x:x1, y:y1})
   t= fyy.subs({x:x1, y:y1})
```

```
delta= r*t-s*2
if (delta>0 and r>0):
   print("The function has minimum at (",x1,",",y1,")")
   q1=f.subs({x:x1, y:y1})
   print("Minimum value:", q1)
elif (delta>0 and r<0):
   print("The function has maximum at (",x1,",",y1,")")
   q2=f.subs({x:x1, y:y1})
   print("Minimum value:", q2)
elif(delta<0):
   print("(",x1,",",y1,") is a saddle point")
elif(delta==0):
   print("it needs further investigations")
```

Output :

```
The critical points are (a,b)= [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)]
( -1 , -1 ) is a saddle point
The function has maximum at ( -1 , 0 )
Minimum value: 1
( -1 , 1 ) is a saddle point
The function has minimum at ( 0 , -1 )
Minimum value: -1
( 0 , 0 ) is a saddle point
The function has minimum at ( 0 , 1 )
Minimum value: -1
( 1 , -1 ) is a saddle point
The function has maximum at ( 1 , 0 )
Minimum value: 1
( 1 , 1 ) is a saddle point
```

Practice problems :

1. Find the Maxima and minima of $f(x,y) = x^3 + y^3 - 3x - 12y + 20$
2. Find the Maxima and minima of $xy(a - x - y)$ with $a = 2$

Viva Questions :

1. What is a saddle point?
   Answer : Saddle point is a point where function have neither maximum nor minimum value.
2. At at (a,b) if $rt - s^2 > 0$ and $r < 0$ then the function has maximum at (a,b). True or false?
   Answer : True
3. At at (a,b) if $rt - s^2 < 0$ then the function has maximum at (a,b). True or false?
   Answer : False
4. The necessary conditions for a function $f(x,y)$ to attain a maximum or minimum value are ------
   Answer : $\frac{\partial f}{\partial x} = 0$ ***and*** $\frac{\partial f}{\partial y} = 0$

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

5.1 Objectives :

Use python

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 forfunc 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 ,axix=0)
Prameters
• 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.

Example : Solve the Differential Equation $\frac{dy}{dx} + y\,tan(x) - y^3\,sec(x) = 0$

```
 # Syntax
#from sympy import *
# init_printing ()
# Define the symbols
# define function
# define the differential equation
# General solution
# Solve the differential equation
# Display the solution
```

Code :

```
from sympy import *
init_printing ()                          # initialize the printing
x  = Symbol ('x')                         # Define the symbols
y = Function ('y') ( x)                   # define function
C1 = Symbol ('C1 ')
y1=diff(y,x)
print ("\n Differential Equation is ")
DE = y1 + y*tan(x) - y**3*sec(x)          # define the differential equation
display ( DE )
print ("\n General Solution is ")
GS = dsolve ( DE )                        # Solve the differential equation
display ( GS ) # Display the solution
```

output :

```
Differential Equation is
```

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

```
General Solution is
```

$$\left[y(x) = -\sqrt{\frac{1}{C_1 - 2\sin(x)}}\cos(x),\ y(x) = \sqrt{\frac{1}{C_1 - 2\sin(x)}}\cos(x)\right]$$

Example : Solve the non-linear equation $p^2 + p(x+y) + xy = 0$

Code :

```
init_printing()  # initialize the printing
x = Symbol('x')
y = Function('y')(x)
C1 = Symbol ('C1 ')
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 Solution  is")
display(sol)
```

output :

Differential Equation is

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

General Solution is

$$\left[y(x) = C_1 - \frac{x^2}{2},\ y(x) = C_1 e^{-x}\right]$$

Example : Solve $xy\left(\frac{dy}{dx}\right)^2 - \frac{(x2+y2)\,dy}{dx} + xy = 0$

Code :

```
from sympy import *
init_printing()  # initialize the printing
x = Symbol('x')
y = Function('y')(x)
C1 = Symbol ('C1 ')
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 Solution  is")
display(sol)
```

output :

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

General Solution is

$$\left[y(x) = -\sqrt{C_1 + x^2},\ y(x) = \sqrt{C_1 + x^2},\ y(x) = C_1 x\right]$$

Viva Questions :

1. What is the order of a differential equation?

   Answer : The order of a differential equation is defined to be that of the highest order derivative it contains

2. What is Numpy?

   Answer : NumPy stands for **"Numerical Python"**. It is used for efficient and general numeric computations on numerical data saved in arrays.

3. What is the use of init_printing in Sympy Package?

   Answer : This tells sympy to display expressions in a nicer format.

4. The order of a dirrerential equation is equal to the number of arbitrary constants in its general solution. True or False?

   Answer : True

LAB 6: Finding GCD using Euclid's algorithm

6.1 Objectives :

Use python

1. to find the GCD of two given integers by Euclid's algorithm

2. to check whether given two integers are relatively prime or not.

Euclidean algorithm : Euclidean algorithm is useful to find GCD of two numbers. The algorithm is as follows:

The two numbers a andb can be assumed positive such that a < b. Let r1 be the remainder when b is divided by a. Then $0 \leq r1 < a$. That is $b = ak1 + r1$.

Now let r2 be the remainder when a is divided by r1. That is $a = r1k2 + r2$. Where $0 \leq r2 < r1$. Continue this process of dividing each divisor by the next remainder. At some stage we obtain remainder 0. The last non-zero remainder is the GCD of a andb. This is known as Euclid's algorithm.

Algorithm analysis:

1. Recursive process - operations are repeated till stopping criterion is reached

2. The output of one step is used as the input of the next step.

Example : Write a program to find the GCD of any two numbers using Euclidian Algorithm

Code :

```
import math
a=int(input("Enter the first number: a="))
b=int(input("Enter the second number: b="))
def gcd(a,b):
    if a< b:
      (a,b) = (b,a)
    r=a%b
    if r == 0:
       return b
    else:
       return (gcd(b, r))          # recursion taking place
print('GCD of', a, 'and', b, 'is', gcd(a, b))
```

output :

```
Enter the first number: a=33
Enter the second number: b=44
GCD of 33 and 44 is 11
```

Relatively prime : Two numbers a and b are called relatively prime or co-prime if their GCD (also known as HCF) is equal to 1.
For example: 2 and 19 are relatively prime, because 1 is the largest natural number that divides both 2 and 19.

Divides
If GCD of a and b is a, then a divides b.
Note that when GCD (a, b) = a is equivalent to the statement a is that the largest natural number that divides both a and b.
For example: The GCD of 4 and 8 is 4, as 4 is the largest number that divides both 4 and 8. Since 4 is one of the given numbers, 4 divides 8.

Example : Write a program to check whether any two given numbers are relatively prime or not

Code :

```
a=int(input("Enter the first number: a :"))
b=int(input("Enter the second number: b :"))
def gcd(a,b):
    if a< b:
        (a,b) = (b,a)
    r=a%b
    if r == 0:
        return b
    else:
        return (gcd(b, r)) # recursion taking place
print('GCD of', a, 'and', b, 'is', gcd(a, b))
ans=gcd(a,b)
# calling function with parameters and printing it out
if (ans==1):
    print("The numbers", a, "and", b, "are relatively prime")
else:
    print("The numbers", a, "and", b, "are not relatively prime")
```

output :

```
Enter the first number: a :33
Enter the second number: b :45
GCD of 33 and 45 is 3
The numbers 33 and 45 are not relatively prime
```

Another output :

```
Enter the first number: a :13
Enter the second number: b :15
GCD of 13 and 15 is 1
The numbers 13 and 15 are relatively prime
```

Example : Prove that 8 divides 128.

Code :

```
a=int(input("Enter the first number: a :"))
b=int(input("Enter the second number: b:"))
if a <= b:
   r=b%a
   if r == 0:
      print("The number", a, "divides", b)
   else :
       print("The number", a, "does not divide", b)
else :
      print("The number", a, "does not divide", b)
```

output :
Enter the first number: a :8
Enter the second number: b:128
The number 8 divides 128

Practice problems :

1. Show that the numbers 14 and 19 are relatively prime
2. Show that 3 divides 96
3. Find gcd of 146 and 295 using Euclidean Algorithm

Viva Questions :

1. Which is the symbol for Modulus In Python?
   Answer : Tthe modulus symbol is represented as the percentage (%) symbol. Hence, it is called the remainder operator.
2. What is the use of return statement ?
   Answer : A return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller.
3. If return statement is not used inside the function, the function will return None. True or False?
   Answer : True
4. When you say that two integers a and b are relatively prime?**Answer :** Two numbers a and b are called relatively prime or co-prime if their GCD (also known as HCF) is equal to 1.
5. Which method is used to accept data from the user?
   Answer : input () method
6. Name three numeric data types in python
   Answer : integer, floating point and complex
7. Which module is to be imported for sqrt( ) ?
   Answer : math module

LAB 7: Solving linear congruence of the form ax≡b (mod m)

7.1 Objectives:

Use python

1. to find solution of linear congruence.
2. to find multiplicative inverse of a mod p.

Example : Show that the linear congruence 6x ≡ 5(mod 15) has no solution.

```
# from sympy import *
# Linear congruence
# Consider ax=b(mod m), x is called the solution of the    congruence
```

Code :

```
from sympy import *
from math import *
a=int( input ('enter integer a '));
b=int( input ('enter integer b '));
n=int( input ('enter integer n '));
d=gcd(a,n)
r=b%d                                   # Remainder calculation
if (r !=0) :
   print ('the congruence has no integer solution ');
else :
   for k in range (1,n-1):
      x=(n*k)/a+ (b/a)
      if(x // 1==x):   # floor function   check whether x is an integer   for eg :  //  = 3
         print ('the solution of the congruence is ', x)
         break
```

output :

```
enter integer a 6
enter integer b 5
enter integer n 15
the congruence has no integer solution
```

Example : Find the solution of the congruence 5x ≡ 3(mod 13).

Code :

```
from math import *
a=int( input ('enter integer a '));
b=int( input ('enter integer b '));
n=int( input ('enter integer n '));
d=gcd(a,n)
r=b%d         # Remainder calculation
if (r !=0) :
```

```
   print ('the congruence has no integer solution ');
else :
   for k in range (1,n-1):
      x=(n*k)/a+ (b/a)
      if(x // 1==x):                    # to   check whether x is an integer
         print ('the solution of the congruence is ', x)
         break
```

output :

```
enter integer a 5
enter integer b 3
enter integer n 13
the solution of the congruence is  11.0
```

E**xample : Find the inverse of 5 mod 13.**

Code :

```
from math import *
a=5
b=1
n=13
d=gcd(a,n)
r=b%d                              # Remainder calculation
if (r !=0) :
   print ('the inverse does not exist ');
else :
   for k in range (1,n-1):
      x=(n*k)/a+ (b/a)
      if(x // 1==x):
         print ('the inverse of 5 under mod 13 is x= ', x)
         break
```

output :

```
The inverse of 5 under mod 13 is x= 8.0
```

Viva Questions :

1. What is the purpose of the \\ operator?
   Answer : \\ operator is used to find the integer pat of the quotient when one number is divided by other
2. What is the output of 2 % 5 ?
   Answer : 2
3. What is the use of break statement?
   Answer : In Python, break is used **to terminate the execution of the loop**
4. break statement is used to terminate a for loop without completing its iteration. True or false?
   Answer : True
5. What is the use of range( ) Function in Python and What Does It Do?
   Answer : range( ) Function is used to generate a sequence of numbers within a given range.

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

8.1 Objectives :

Use python

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.

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

Code :

```
from sympy import *
import numpy as np
A= np. matrix ([[1 ,2 ,-1],[2 ,1 , 4],[3 ,3 , 4]]) # numpy . matrix ( data )
B= np.matrix ([[0],[0],[0]])
r= np. linalg . matrix_rank ( A ) # Return rank of the array.
n= A. shape [1] # Returns the shape of an array.
if ( r==n ):
  print (" System has trivial solution ")
else :
  print (" System has", n-r , "non - trivial solution (s)")
```

output :

System has trivial solution

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

Code :

```
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)")
```

output :

System has 1 non - trivial solution (s)

Example : Examine the consistency of the following system of equations and solve if consistent. $x_1 + 2x_2 - x_3 = 1,\ 2x_1 + x_2 + 4x_3 = 2,\ 3x_1 + 3x_2 + 4x_3 = 1$

Code :

```
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 ("The Rank of the   matrix A is rA= ")
print(rA)
rAB =np . linalg . matrix_rank ( AB )
print ("The Rank of the Augmented matrix AB is ")
print(rAB)
n=A . shape [1]
if ( rA==rAB ):
   if ( rA==n ):
        print ("The system has unique solution ")
        print ( np . linalg . solve (A , B ) )
   else :
      print ("The system has infinitely many solutions ")
else :
    print ("The system of equations is inconsistent ")
```

output :

```
The Rank of the   matrix A is rA=
3
The Rank of the Augmented matrix AB is
3
The system has unique solution
[[ 7.]
 [-4.]
 [-2.]]
```

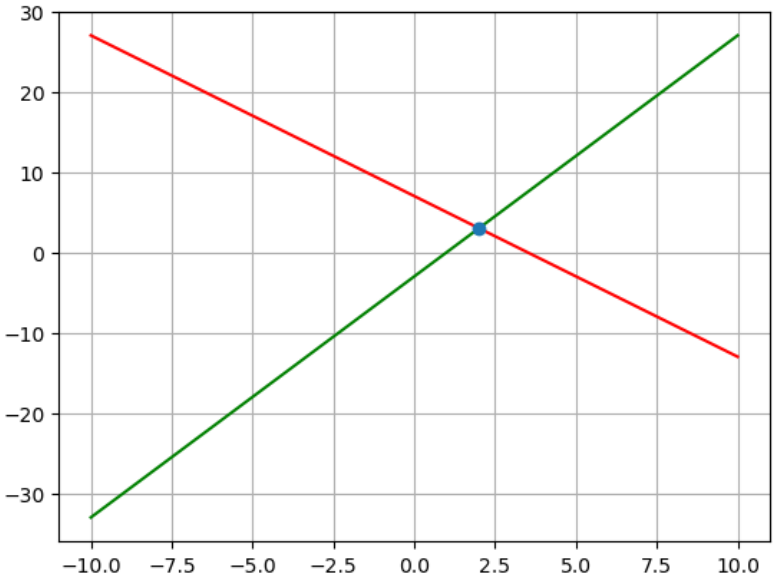
Example : Obain the solution of 2x + y = 7; 3x - y = 3 graphically

Code :

```
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)
y1=7-2*x
y2=3*x-3
plt.plot(x,y1,'r')
plt.plot(x,y2,'g')
plt.plot(p,q,marker='o')
plt . grid ()
plt . show ()
```

output :

```
point of intersection is A( 2 , 3 )
```



Practice problems

1. Check whether the following system of homogenous linear equation has non-trivial solution $x_1 + 2x_2 - x_3 = 0, 2x_1 + x_2 + 4x_3 = 0, 3x_1 + 3x_2 + 4x_3 = 0$
2. Check whether the following system of homogenous linear equation has non-trivial solution $x_1 + 2x_2 - x_3 = 0, 2x_1 + x_2 + 4x_3 = 0, x_1 - x_2 + 5x_3 = 0$.
3. Obtain the solution of 3x + 5y = 1; x + y = 1 graphically.

Viva Questions :

1. The System of equations AX=B is consistent if rank of A < rank of (A,B). True or false ?
   Answer : False
2. A homogeneous system of equations is always consistent. True or False?
   Answer : True
3. If a homogeneous linear system AX=0 has more variables than equations, then it has a nontrivial solution. True or False?
   Answer : True
4. What is meant by trivial solution of a linear homogeneous system of equations?
   Answer : Since the zero solution is the "obvious" solution, hence it is called a trivial solution. Any solution which has at least one component non-zero (thereby making it a non-obvious solution) is termed as a "non-trivial" solution
5. In Sympy which function is used to **solve** equations and expressions that contain symbolic math variables
   Answer : SymPy's **solve**() function can be used to solve equations and expressions that contain symbolic math variables
6. What is the use of numpy.arrange( ) function?
   Answer : By using **Numpy. arange(start, stop, step)** values are generated within the half-open interval [start, stop), with spacing between values given by step.

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

9.1 Objectives:

Use python

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

Example 1: Solve the system of equations using Gauss-Seidel method: 20x+y−2z = 17; 3x+20y−z =−18; 2x− 3y + 20z = 25.

```
#Syntax
# Gauss Seidel Iteration
# Defining equations to be solved
# In diagonally dominant form
# Initial setup
# Reading tolerable error
# Implementation of Gauss Seidel Iteration
```

Code :

```
from sympy import *
x,y,z=symbols('x,y,z')
f1 =  ( 17-y+2*z)/20
f2 =  (-18-3*x+z)/20
f3 =  (25-2*x+3*y)/20
x0 = 0                    # Initial setup
y0 = 0
z0 = 0
count = 1
e = float (input ('Enter acceptable error : ') )          # Reading tolerable error
print ('\n Count \tx\ty\tz \n')
condition = True
while condition :                                  # Implementation of Gauss Seidel Iteration
        x1 = f1.subs ({x:x0, y:y0, z:z0} )
        y1 = f2.subs ({ x:x1, y:y0, z:z0} )
        z1 = f3.subs ({x:x1, y:y1, z: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 is : x=%0.3f , y=%0.3f and z = %0.3f\n'% ( x1 , y1 , z1 ) )
```

output :

```
Enter acceptable error : 0.0001

 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

4       1.0000  -1.0000 1.0000


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

Example : Solve the system of equations using Gauss-Seidel method: $10x + y + z = 12$, $x + 10y + z = 12$, $x + y + 10z = 12$

Code :

```
from sympy import *
x,y,z=symbols('x,y,z')
f1 =  ( 12-y-z)/10
f2 =  (12-x-z)/10
f3 =  (12-x-y)/10
x0 = 0                  # Initial setup
y0 = 0
z0 = 0
count = 1
e = float (input ('Enter acceptable error : ') )          # Reading tolerable error
print ('\n Count \tx\ty\tz \n')
condition = True
while condition :                                  # Implementation of Gauss Seidel Iteration
        x1 = f1.subs ({x:x0, y:y0, z:z0} )
        y1 = f2.subs ({ x:x1, y:y0, z:z0} )
        z1 = f3.subs ({x:x1, y:y1, z: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 is : x=%0.3f , y=%0.3f and z = %0.3f\n'% ( x1 , y1 , z1 ) )
```

output :

```
Enter acceptable error : 0.001

 Count  x       y       z

1       1.2000  1.0800  0.9720

2       0.9948  1.0033  1.0002

3       0.9996  1.0000  1.0000


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

Practice problem : Solve x+2y−z = 3; 3x−y+2z = 1; 2x−2y+6z = 2 by Gauss-Seidel Iteration method.

Viva Questions :

1. What is the use of symbols() method in python?
   Answer : Using symbols() method, we can declare some variables for the use of mathematical expression and polynomials
2. What is the use of while loop in Python?
   Answer : Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.
3. What is meant by tolerance?
   Answer : Tolerance is the level of error that is acceptable for an engineering application
4. What is the use of \t in Python ?
   Answer : It is used in representing certain whitespace characters. When you type \t in a program, it will insert a tab into the text. This can be useful for formatting text or creating tables.
5. What is the use of \n in Python ?
   Answer : In Python, \n is a special character used to represent a newline or line break in a string. . It is also known as an escape character and is used to create a new line in the output when printing a string
6. What does %d do in Python?
   Answer : The %d operator is used as a placeholder to specify integer values, decimals, or numbers. It allows us to print numbers within strings or other values. The %d operator is put where the integer is to be specified. Floating-point numbers are converted automatically to decimal values.

LAB 10 : Compute eigenvalues and eigenvectors and find the largest and smallest eigenvalue by Rayleigh power method.

10.1 Objectives:

Use python

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)

Returns the following:

• w(. . . , M) array

The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be of complex type, unless the imaginary part is zero in which case it will be cast to a real type. When a is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs.

• v(. . . , M, M) array

The normalized (unit "length") eigenvectors, such that the column v[:,i] is the eigenvector corresponding to the eigenvalue w[i].

2. np.linalg.eigvals(A): Computes th 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. lambdaarguments: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 are syntactically restricted to a single expression.

• Example: f=lambda x :x ∗∗2−3 ∗x+1 (Mathematically f(x) = x2 −3x+1)

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

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

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

Code :

```
import numpy as np
A=np . array ([[4 ,3 , 2],[1 ,4 , 1],[3 , 10 , 4]])
print ("\n Given matrix is : \n", A )
eigvalue, eigvector= np . linalg .eig ( A )
print ("\n Eigen values  are: \n", eigvalue )
print ("\n Eigen vectors are : \n",  eigvector )
```

output :

```
Given matrix is :
[[ 4  3  2]
[ 1  4  1]
[ 3 10  4]]

Eigen values  are:
[8.98205672 2.12891771 0.88902557]

Eigen vectors are :
[[-0.49247712 -0.82039552 -0.42973429]
[-0.26523242  0.14250681 -0.14817858]
[-0.82892584  0.55375355  0.89071407]]
```

Practice problem : Obtain the eigen values and eigen vectors for the given matrix.

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

10.2. Largest eigenvalue and corresponding eigenvector by Rayleigh method :

For a given Matrix Aand a given initial eigenvector X0, 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 by taking [1 1 1]' as the initial eigen vector.**

Code :

```
import numpy as np
A=np.array([[6,-2,2],[-2,3,-1],[2,-1,3]])
X=np.array([1,1,1])
count=15
oldeigenvalue=0
for i in range(count):
    X=np.dot(A,X)
    eigenvalue=max(abs(X))
    X=X/eigenvalue
    oldeigenvalue=eigenvalue
print ("\n Given matrix is : \n", A )
print("Numerically Largest Eigen Value is %0.3f" %(eigenvalue))
print("Corresponding Eigen Vector is", X)
```

output :

```
 Given matrix is :
 [[ 6 -2  2]
 [-2  3 -1]
 [ 2 -1  3]]
Numerically Largest Eigen Value is 8.000
Corresponding Eigen Vector is [ 1.  -0.5  0.5]
```

Example 2 : Compute the numerically largest eigenvalue of P = $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & 1 & 2 \end{bmatrix}$ **by power method**

Code :

```
import numpy as np
A=np.array([[2,-1,0],[-1,2,-1],[0,-1,2]])
X=np.array([1,0,0])
count=15
oldeigenvalue=0
for i in range(count):
    X=np.dot(A,X)
    eigenvalue=max(abs(X))
    X=X/eigenvalue
    oldeigenvalue=eigenvalue
print ("\n Given matrix is : \n", A )
```

```
print("Numerically Largest Eigen Value is %0.3f" %(eigenvalue))
print("Corresponding Eigen Vector is", X)
```

output :

```
 Given matrix is :
 [[ 2 -1  0]
 [-1  2 -1]
 [ 0 -1  2]]
Numerically Largest Eigen Value is 3.414
Corresponding Eigen Vector is [ 0.70757087 -1.          0.70664269]
```

Viva Questions :

1. If A is a triangular matrix, then the diagonal elements of A are the eigenvalues of A. True or False?
   Answer : True
2. A and A^T have the same eigenvectors. True or False?
   Answer : True
3. Which function is used to solve the eigenvalue/eigenvector problem for a square matrix in python?
   Answer : The main built-in function in Python to solve the eigenvalue/eigenvector problem for a square array is the eig function in numpy.linalg.
4. What is Numpy? What is its use ?
   Answer : NumPy is an open source numerical Python library. NumPy contains a multi-dimentional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays such as trigonometric, statistical and algebraic routines.
5. How to find Dot product of two arrays a and b using Numpy?
   Answer : numpy.dot(a, b) returns the dot product of vectors a and b.
6. what is use of max () function in python?
   Answer : max() is used to compute the maximum value in the list given
7. What is the use of **max(abs(X))** where X is a vector?
   Answer : max(abs(X)) gives the maximum among the absolute values of the elements of X. i.e. the numerically largest value in X.
8. The product of the eigenvalues of A is the equal to det(A), the determinant of A. True or False?
   Answer : True