Mathematics Lab (Calculus, Linear Algebra & Differential equations)

B.E. II-Semester
2021-22

LAB MANUAL

[MASTER MANUAL]

Prepared by

Dr. N. Anil Kumar (Assoc Prof), IT

DEPARTMENT OF INFORMATION TECHNOLOGY

VASAVI COLLEGE OF ENGINEERING
(AUTONOMOUS)

INDEX

INSTITUTE VISION AND MISSION

Vision

Striving for a symbiosis of technological excellence and human values.

Mission

To arm young brains with competitive technology and nurture holistic development of the individuals for a better tomorrow.

Our Quality Policy

Education without quality is like a flower without fragrance. It is our earnest resolve to strive towards imparting high standards of teaching, training and developing human resources.

DEPARTMENT VISION, MISSION, PEOs & PSOs

Vision

To be a center of excellence in the emerging areas of Information Technology.

Mission

- Provide a comprehensive learning experience on the latest technologies and applications.
- Equip the stakeholders with latest technical knowledge and leadership skills with collaboration to become competent professionals.
- Motivate innovation and contribute to the societal issues with human values and professional ethics.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's)

A Graduate of Information Technology will be able to:

PEO1: Pursue higher studies in multidisciplinary areas with research orientation.

PEO2: Develop core IT competencies aligned with emerging industry trends to become global leaders with ethical values.

PEO3: Engage in continuous learning and address the societal problems with sustainable solutions.

PROGRAM SPECIFIC OUTCOMES (PSO's)

Our students, upon graduation from the program, will be able to

PSO1: Identify and develop software solutions using programming languages, tools and AI/ML concepts.

PSO2: Design, develop and maintain secure stand-alone, embedded and networked systems.

PSO3: Analyze the architectures of autonomous or semi-autonomous intelligent systems and apply to real-time scenarios.

PROGRAM OUTCOMES (POs)

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 analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, 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 modeling to complex engineering activities with an understanding of the limitations.
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 and sustainability:** Understand the impact of the professional engineering solutions in 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 teamwork:** 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 one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

COURSE OUTCOMES:

Course Name: Mathematics Lab (Calculus, Linear Algebra & Differential equations)

S.No	Course Outcomes	BTL
1	Perform basic operations on vectors and matrices using Linear Algebra module of NumPy and SciPy.	2
2	Use matplotlib library for numerical analysis and visualization.	2
3	Perform matrix decompositions and solve system of linear equations.	3
4	Use SymPy library to solve calculus problems.	3
5	Solve Differential equations using SymPy library.	2

MAPPING LEVELS:

Correlation levels 1, 2 or 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High)

CO/PO/PSO MAPPING

CO/ PO /PSO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	1			3							2	3		
CO2	3	1			2							1	3		
CO3	3	3	1		3							2	3	2	
CO4	3	2			2							1	3		
CO5	3	1			1								3		

CO/PO/PSO JUSTIFICATION

MAPPING	CORRELATION LEVELS	JUSTIFICATION
CO1-PO1	**3**	Student will learn fundamental knowledge of Linear algebra and programming
CO1-PO2	**1**	With numerical analysis and visualization student will be able to formulate problems.
CO1-PO5	**3**	Numpy, scipy, mathplotlib are modern libraries which can be used to solve problems.
CO1-PO12	**2**	Numpy, scipy, mathplotlib will be used throughout the career of the students.
CO1-PSO1	**3**	Apart from basic mathematical knowledge students will understand how to implement them using various libraries in python
CO2-PO1	**3**	Student will learn fundamental knowledge of matrix operations
CO2-PO2	**1**	With the knowledge of matrix analysis student will be able analyse problems.
CO2-PO5	**2**	scipy are modern libraries which can be used to solve problems.
CO2-PO12	**1**	Matrix analysis will be extensively used in machine learning and image processing.
CO2-PSO1	**3**	Students will understand how to implement matrix operations using various libraries in python
CO3-PO1	**3**	Student will learn fundamental knowledge of matrix decomposition.
CO3-PO2	**3**	With the knowledge of solving linear equations student will be able analyse and solve problems.
CO3-PO3	**1**	various matrix decomposition can be used in signal processing, pseuodinverse, least square minimization, etc
CO3-PO5	**3**	Libraries like scipy which are modern libraries, can be used to solve problems.
CO3-PO12	**2**	Matrix decomposition will be extensively used in machine learning
CO3-PSO1	**3**	Students will understand how to implement matrix decomposition using various libraries in python.
CO3-PSO2	**2**	various matrix decomposition can be used in signal processing, pseudoinverse, least square minimization, etc can be used to design software solutions.
CO4-PO1	**3**	Student will learn fundamental knowledge of calculus
CO4-PO2	**2**	With the analysis of extreme values student should be able to solve the problems
CO4-PO5	**2**	Libraries like scipy which are modern libraries, can be used to solve problems in calculus.
CO4-PO12	**1**	saddle point, extreme points will be used find optimum values of the equations.
CO4-PSO1	**3**	Students will understand how to find extreme values using various libraries in python.
CO5-PO1	**3**	Student will learn fundamental knowledge of differential equations and

		solving.
CO5-PO2	**1**	With the understanding of differential equations, students will be able to formulate complex problems.
CO5-PO5	**1**	Libraries like scipy, which are modern libraries, can be used to solve differential equations.
CO5-PSO1	**3**	Students will understand how to solve differential equations using various libraries in python.

DEPARTMENT OF INFORMATION TECHNOLOGY

Syllabus for B.E II- SEMESTER

Mathematics Lab (Calculus, Linear Algebra & Differential equations)

Instruction:2Hrs/ week	SEE Marks : 50	Course Code : U21BS211MA
Credits : 1	CIE Marks :30	Duration of SEE : 3 Hours

Course Objective:	**Course Outcomes:**
The course will enable the students to:	**At the end of the course student will be able to:**
1. Demonstrate the linear algebra, calculus and differential equation concepts using SciPy and Numpy. 2. Provide hands on experience on plotting and symbolic mathematics.	1.Perform basic operations on vectors and matrices using Linear Algebra module of NumPy and SciPy. 2. Use matplotlib library for numerical analysis and visualization. 3. Perform matrix decompositions and solve system of linear equations. 4. Use SymPy library to solve calculus problems. 5. Solve Differential equations using SymPy library.

List of Experiments

1. Introduction to Anaconda &Jupyter Notebook setup and evaluating elementary functions.
2. Basic operations on Matrix & Vector.
3. Matrix analysis: Rank, Determinant, Trace, Orthogonal basis & Inverse of matrices.
4. Eigen values and Eigenvectors of Matrix.
5. Matrix decompositions: SVD, QR, LU, Pseudo Inverse
6. Solve system of linear equations.
7. Data plotting (2D,3D) of various mathematical functions.
8. Test the convergence of infinite series i.e., power, Taylor.
9. Introduction to calculus and examine minima, maxima and saddle points of a given function.
10. Application of definite integrals to area & volume calculations.
11. Solving differential equations.

Learning Resources:

1. Kong, Qingkai, Timmy Siauw, and Alexandre Bayen. Python Programming and Numerical Methods: A Guide for Engineers and Scientists. Academic Press, 2020.
2. https://numpy.org/doc/1.21/user/tutorialsindex.html
3. https://personal.math.ubc.ca/—pwalls/math-python/linear-alqebra/linear-algebra-scipy/

System requirements

- Anaconda/Jupyter (software that you are required to install)

LIST OF EXPERIMENTS

S.No	Name of the Experiment	CO mapping	PO/ PSO mapping
1	Introduction to Anaconda & Jupyter Notebook setup and evaluating elementary functions.	1	1, 5, 12
2	Basic operations on Matrix & Vector.	2	1,5
3	Matrix analysis: Rank, Determinant, Trace, Orthogonal basis & Inverse of matrices.	2	1,2,5
4	Eigen values and Eigenvectors of Matrix.	2	1,5,12
5	Matrix decompositions: SVD, QR, LU, Pseudo Inverse	3	1,2,5,12
6	Solve system of linear equations.	3	1,2,5
7	Data plotting (2D,3D) of various mathematical functions.	1	1,5
8	Test the convergence of infinite series i.e., power, Taylor.	4	1,5
9	Intro to calculus and examine minima, maxima and saddle points of a given function.	4	1,2,5,12
10	Application of definite integrals to area & volume calculations.	4	1,2,5
11	Solving differential equations.	5	1,2,5

Introduction to Lab

The mathematics laboratory is a place where student can experiment and explore mathematical concepts. Mathematical truths are accepted only on the basis of proofs, and not through experimentation. The mathematics laboratory is a place to enjoy mathematics through informal exploration. Mathematics lab is carried out using python programming language.

1. Introduction to Python

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions: **Python 2 and Python 3**. Both are quite different.

Beginning with Python programming:

1.1) Finding an Interpreter:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like **https://ide.geeksforgeeks.org/** that can be used to run Python programs without installing an interpreter.

Windows: There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) that comes bundled with the Python software downloaded from **http://python.org/**.

Linux: Python comes preinstalled with popular Linux distros such as Ubuntu and Fedora. To check which version of Python you're running, type "python" in the terminal emulator. The interpreter should start and print the version number.

macOS: Generally, Python 2.7 comes bundled with macOS. You'll have to manually install Python 3 from **http://python.org/**.

1.2) Writing our first program:

Just type in the following code after you start the interpreter.

```
# Script Begins
```

```
print("GeeksQuiz")


# Scripts Ends
```

Output:

```
GeeksQuiz
```

Let's analyze the script line by line.

Line 1: [*# Script Begins*] In Python, comments begin with a #. This statement is ignored by the interpreter and serves as documentation for our code.

Line 2: [print("GeeksQuiz")] To print something on the console, print() function is used. This function also adds a newline after our message is printed(unlike in C). Note that in Python 2, "print" is not a function but a keyword and therefore can be used without parentheses. However, in Python 3, it is a function and must be invoked with parentheses.

Line 3: [*# Script Ends*] This is just another comment like in Line 1.

Python designed by Guido van Rossum at CWI has become a widely used general-purpose, high-level programming language.

Prerequisites:

Knowledge of any programming language can be a plus.

Reason for increasing popularity

1. Emphasis on **code readability, shorter codes**, ease of writing
2. Programmers can express logical concepts in **fewer lines** of code in comparison to languages such as C++ or Java.
3. Python supports **multiple** programming paradigms, like object-oriented, imperative and functional programming or procedural.
4. There exists inbuilt functions for almost all of the frequently used concepts.
5. Philosophy is "Simplicity is the best".

LANGUAGE FEATURES

- **Interpreted**
    - There are no separate compilation and execution steps like C and C++.
    - Directly *run* the program from the source code.
    - Internally, Python converts the source code into an intermediate form called bytecodes which is then translated into native language of specific computer to run it.
    - No need to worry about linking and loading with libraries, etc.

- **Platform Independent**
    - Python programs can be developed and executed on multiple operating system platforms.
    - Python can be used on Linux, Windows, Macintosh, Solaris and many more.
- **Free and Open Source;** Redistributable
- **High-level Language**
    - In Python, no need to take care about low-level details such as managing the memory used by the program.
- **Simple**
    - Closer to English language;Easy to Learn
    - More emphasis on the solution to the problem rather than the syntax
- **Embeddable**
    - Python can be used within C/C++ program to give scripting capabilities for the program's users.
- **Robust**:
    - Exceptional handling features
    - Memory management techniques in built
- **Rich Library Support**
    - The Python Standard Library is very vast.
    - Known as the **"batteries included"** philosophy of Python ;It can help do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, email, XML, HTML, WAV files, cryptography, GUI and many more.
    - Besides the standard library, there are various other high-quality libraries such as the Python Imaging Library which is an amazingly simple image manipulation library.

Python vs C

Python	C
Dynamically Typed • No need to declare anything. An assignment statement binds a name to an object, and the object can be of any type. • No type casting is required when using container objects	**Statically Typed** • All variable names (along with their types) must be explicitly declared. • Type casting is required
Concise Express much in limited words	**Verbose** Contains more words
Compact	**Less Compact**

Uses Indentation for structuring code	**Uses braces for structuring code**

The classical **Hello World program** illustrating the **relative verbosity** of a C Program and Python Program C Code

```
#include<stdio.h>
Int main()
{
printf("Hello, world!");
return 1;
}
```

Python Code

```
print("Hello,
world!")
```

Softwares making use of Python

Python has been successfully embedded in a number of software products as a scripting language.

1. GNU Debugger uses Python as a **pretty printer** to show complex structures such as C++ containers.
2. Python has also been used in artificial intelligence
3. Python is often used for **natural language processing** tasks.

Current Applications of Python

1. Several Linux distributions use installers written in Python example in Ubuntu we have the **Ubiquity**
2. Python has seen extensive use in the **information security industry**, including in exploit development.
3. Raspberry Pi– single board computer uses Python as its principal user-programming language.
4. Python is now being used in **Game Development** areas also.

Pros:

1. Ease of use
2. Multi-paradigm Approach

Cons:

1. Slow speed of execution compared to C, C++
2. Absence from mobile computing and browsers
3. For the C, C++ programmers switching to python can be irritating as the language requires proper indentation of code. Certain variable names commonly used like sum are functions in python. So, C, C++ programmers must look out for these.

Industrial Importance

Most of the companies are now looking for candidates who know about Python Programming. Those having the knowledge of python may have more chances of impressing the interviewing panel. So I would suggest that beginners should start learning python and excel in it.

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-to-use language. In addition, the community is very active there. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

Advantages :

1. Presence of third-party modules
2. Extensive support libraries(NumPy for numerical calculations, Pandas for data analytics etc)
3. Open source and community development
4. Versatile, Easy to read, learn and write
5. User-friendly data structures
6. High-level language
7. Dynamically typed language(No need to mention data type based on the value assigned, it takes data type)
8. Object-oriented language
9. Portable and Interactive
10. Ideal for prototypes – provide more functionality with less coding
11. Highly Efficient(Python’s clean object-oriented design provides enhanced process control, and the language is equipped with excellent text processing and integration capabilities, as well as its own unit testing framework, which makes it more efficient.)
12. (IoT)Internet of Things Opportunities
13. Interpreted Language
14. Portable across Operating systems

Applications :

1. GUI based desktop applications
2. Graphic design, image processing applications, Games, and Scientific/ computational Applications
3. Web frameworks and applications
4. Enterprise and Business applications
5. Operating Systems

6. Education
7. Database Access
8. Language Development
9. Prototyping
10. Software Development

Organizations using Python :

1. Google(Components of Google spider and Search Engine)
2. Yahoo(Maps)
3. YouTube
4. Mozilla
5. Dropbox
6. Microsoft
7. Cisco
8. Spotify
9. Quora

Now moving on further **Lets start our basics of Python** . I will be covering the basics in some small sections. Just go through them and trust me you'll learn the basics of Python very easily.

2 Introduction and Setup:

If you are on **Windows OS** download Python by https://www.python.org/downloads/windows/ and now install from the setup and in the start menu type IDLE.IDLE, you can think it as an Python's IDE to run the Python Scripts.

If you are on **Linux/Unix-like** just open the terminal and on 99% linux OS Python comes preinstalled with the OS.Just type 'python3' in terminal and you are ready to go. It will look like this :

The ">>> " represents the python shell and its ready to take python commands and code.

2.1 Variables and Data Structures

In other programming languages like C, C++, and Java, you will need to declare the type of variables but in Python you don't need to do that. Just type in the variable and when values will be given to it, then it will automatically know whether the value given would be an int, float, or char or even a String.

```
# Python program to declare variables

myNumber=3

print(myNumber)
```

```
myNumber2 =4.5

print(myNumber2)


myNumber="helloworld"

print(myNumber)
```

Output:

```
3

4.5

helloworld
```

3. Libraries in python

3.1 numpy Library:

Standard Python distribution doesn't come bundled with NumPy module. To install NumPy using popular Python package installer, **pip**.

pip install numpy

The best way to enable NumPy is to use an installable binary package specific to your operating system. These binaries contain full SciPy stack (inclusive of NumPy, SciPy, matplotlib, IPython, SymPy and nose packages along with core Python).

Windows
Anaconda (from https://www.continuum.io) is a free Python distribution for SciPy stack. It is also available for Linux and Mac.

Canopy (https://www.enthought.com/products/canopy/) is available as free as well as commercial distribution with full SciPy stack for Windows, Linux and Mac.

Python (x,y): It is a free Python distribution with SciPy stack and Spyder IDE for Windows OS. (Downloadable from https://www.python-xy.github.io/)

Linux

Package managers of respective Linux distributions are used to install one or more packages in SciPy stack.

For Ubuntu

```
sudo apt-get install python-numpy
python-scipy python-matplotlibipythonipythonnotebook python-pandas
python-sympy python-nose
```

For Fedora

```
sudo yum install numpyscipy python-matplotlibipython
python-pandas sympy python-nose atlas-devel
```

To test whether NumPy module is properly installed, try to import it from Python prompt.

```
import numpy
```

If it is not installed, the following error message will be displayed.

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import numpy
ImportError: No module named 'numpy'
```

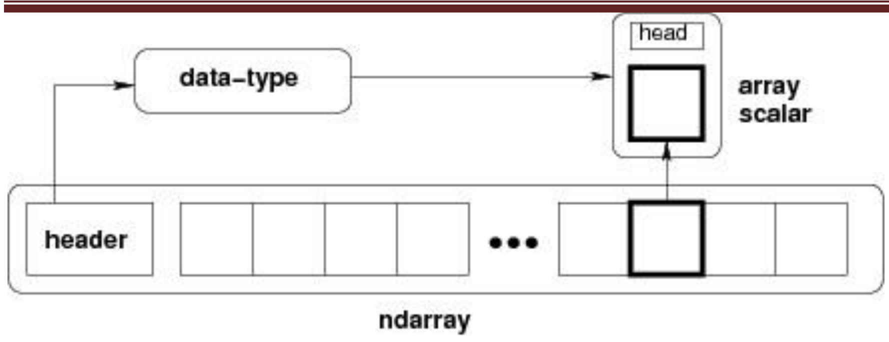
Alternatively, NumPy package is imported using the following syntax −

```
import numpy as np
```

The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.

Every item in anndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**).

Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types. The following diagram shows a relationship between ndarray, data type object (dtype) and array scalar type −



An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows −

numpy.array

It creates anndarray from any object exposing array interface, or from any method that returns an array.

numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

The above constructor takes the following parameters −

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional
3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through

6	**ndmin** Specifies minimum dimensions of resultant array

Look at the following examples to understand better.

Example

```
importnumpyas np
a =np.array([1,2,3])
print a
```

The output is as follows −

```
[1, 2, 3]
```

Example

```
# more than one dimensions
importnumpyas np
a =np.array([[1,2],[3,4]])
print a
```

The output is as follows −

```
[[1, 2]
 [3, 4]]
```

Example

```
# minimum dimensions
importnumpyas np
a =np.array([1,2,3,4,5],ndmin=2)
print a
```

The output is as follows −

```
[[1, 2, 3, 4, 5]]
```

Example

```
# dtype parameter
import numpy as np
```

```
a =np.array([1,2,3],dtype= complex)
print (a)
```

The output is as follows −

```
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

The **ndarray** object consists of contiguous one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block. The memory block holds the elements in a row-major order (C style) or a column-major order (FORTRAN or MatLab style).

ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example

```
Import numpy as np
a =np.array([[1,2,3], [4,5,6]])
print a.shape
```

The output is as follows −

```
(2, 3)
```

Example

```
# this resizes the ndarray
importnumpyas np

a =np.array([[1,2,3],[4,5,6]])
a.shape=(3,2)
print a
```

The output is as follows −

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

Example

NumPy also provides a reshape function to resize an array.

```
importnumpyas np
a =np.array([[1,2,3],[4,5,6]])
b =a.reshape(3,2)
print b
```

The output is as follows −

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

ndarray.ndim
This array attribute returns the number of array dimensions.

Example 1

```
# an array of evenly spaced numbers
importnumpyas np
a =np.arange(24)
print a
```

The output is as follows −

```
[0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 17 18 19 20 21 22 23]
```

Example

```
# this is one dimensional array
importnumpyas np
a =np.arange(24)
a.ndim

# now reshape it
b =a.reshape(2,4,3)
print b
# b is having three dimensions
```

The output is as follows −

```
[[[ 0,  1,  2]
  [ 3,  4,  5]
  [ 6,  7,  8]
  [ 9, 10, 11]]
```

```
 [[12, 13, 14]
  [15, 16, 17]
  [18, 19, 20]
  [21, 22, 23]]]
```

numpy.arange

This function returns an **ndarray** object containing evenly spaced values within a given range. The format of the function is as follows −

```
numpy.arange(start, stop, step, dtype)
```

The constructor takes the following parameters.

Sr.No.	Parameter & Description
1	**start** The start of an interval. If omitted, defaults to 0
2	**stop** The end of an interval (not including this number)
3	**step** Spacing between values, default is 1
4	**dtype** Data type of resulting ndarray. If not given, data type of input is used

The following examples show how you can use this function.

Example

```
Import numpy as np
x =np.arange(5)
print x
```

Its output would be as follows −

```
[0  12  3  4]
```

numpy.linspace

This function is like **arange()** function. In this function, instead of step size, the number of evenly spaced values between the interval is specified. The usage of this function is as follows −

numpy.linspace(start, stop, num, endpoint, retstep, dtype)

```
import numpy as np
x =np.linspace(10,20,5)
print x
```

Contents of ndarray object can be accessed and modified by indexing or slicing, just like Python's in-built container objects.

As mentioned earlier, items in ndarray object follows zero-based index. Three types of indexing methods are available − **field access, basic slicing** and **advanced indexing**.

Basic slicing is an extension of Python's basic concept of slicing to n dimensions. A Python slice object is constructed by giving **start, stop**, and **step** parameters to the built-in **slice** function. This slice object is passed to the array to extract a part of array.

Example

```
Import numpy as np
a =np.arange(10)
s =slice(2,7,2)
print a[s]
```

3.2 scipy library:

SciPy, a scientific library for Python is an open source, BSD-licensed library for mathematics, science and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that, it should work with NumPy arrays.

Installation of SciPy

If you have Python and PIP already installed on a system, then installation of SciPy is very easy.

Install it using this command:

```
pip install scipy
```

The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays and provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install and are free of charge.

NumPy and SciPy are easy to use, but powerful enough to depend on by some of the world's leading scientists and engineers.

SciPy Sub-packages

SciPy is organized into sub-packages covering different scientific computing domains. These are summarized in the following table −

scipy.linalg	Linear algebra routines
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.stats	Statistics

Data Structure

The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for Linear Algebra, Fourier Transforms and Random Number Generation, but not with the generality of the equivalent functions in SciPy.

To start with, let us compare the ‘pi’ value by considering the following example.

```
#Import pi constant from both the packages
fromscipy.constantsimport pi
from math import pi

print("sciPy - pi = %.16f"%scipy.constants.pi)
print("math - pi = %.16f"%math.pi)
```

The above program will generate the following output.

```
sciPy - pi = 3.1415926535897931
math - pi = 3.1415926535897931
```

scipy.linalg

inv(a[, overwrite_a, check_finite])	Compute the inverse of a matrix.
solve(a, b[, sym_pos, lower, overwrite_a, ...])	Solves the linear equation set a @ x == b for the unknown x for square *a* matrix.
solve_banded(l_and_u, ab, b[, overwrite_ab, ...])	Solve the equation a x = b for x, assuming a is banded matrix.

solveh_banded(ab, b[, overwrite_ab, ...])	Solve equation a x = b.
solve_circulant(c, b[, singular, tol, ...])	Solve C x = b for x, where C is a circulant matrix.
solve_triangular(a, b[, trans, lower, ...])	Solve the equation *a x = b* for *x*, assuming a is a triangular matrix.
solve_toeplitz(c_or_cr, b[, check_finite])	Solve a Toeplitz system using Levinson Recursion
matmul_toeplitz(c_or_cr, x[, check_finite, ...])	Efficient Toeplitz Matrix-Matrix Multiplication using FFT
det(a[, overwrite_a, check_finite])	Compute the determinant of a matrix
norm(a[, ord, axis, keepdims, check_finite])	Matrix or vector norm.
lstsq(a, b[, cond, overwrite_a, ...])	Compute least-squares solution to equation Ax = b.
pinv(a[, atol, rtol, return_rank, ...])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
pinvh(a[, atol, rtol, lower, return_rank, ...])	Compute the (Moore-Penrose) pseudo-inverse of a Hermitian matrix.
kron(a, b)	Kronecker product.
khatri_rao(a, b)	Khatri-rao product
tril(m[, k])	Make a copy of a matrix with elements above the kth diagonal zeroed.
triu(m[, k])	Make a copy of a matrix with elements below the kth diagonal zeroed.
orthogonal_procrustes(A, B[, check_finite])	Compute the matrix solution of the orthogonal Procrustes problem.
matrix_balance(A[, permute, scale, ...])	Compute a diagonal similarity transformation for row/column balancing.
subspace_angles(A, B)	Compute the subspace angles between two matrices.

bandwidth(a)	Return the lower and upper bandwidth of a 2D numeric array.
issymmetric(a[, atol, rtol])	Check if a square 2D array is symmetric.
ishermitian(a[, atol, rtol])	Check if a square 2D array is Hermitian.
LinAlgError	Generic Python-exception-derived object raised by linalg functions.
LinAlgWarning	The warning emitted when a linear algebra related operation is close to fail conditions of the algorithm or loss of accuracy is expected.

Eigenvalue Problems

eig(a[, b, left, right, overwrite_a, ...])	Solve an ordinary or generalized eigenvalue problem of a square matrix.
eigvals(a[, b, overwrite_a, check_finite, ...])	Compute eigenvalues from an ordinary or generalized eigenvalue problem.
eigh(a[, b, lower, eigvals_only, ...])	Solve a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eigvalsh(a[, b, lower, overwrite_a, ...])	Solves a standard or generalized eigenvalue problem for a complex Hermitian or real symmetric matrix.
eig_banded(a_band[, lower, eigvals_only, ...])	Solve real symmetric or complex Hermitian band matrix eigenvalue problem.
eigvals_banded(a_band[, lower, ...])	Solve real symmetric or complex Hermitian band matrix eigenvalue problem.
eigh_tridiagonal(d, e[, eigvals_only, ...])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.
eigvalsh_tridiagonal(d, e[, select, ...])	Solve eigenvalue problem for a real symmetric tridiagonal matrix.

Decompositions

lu(a[, permute_l, overwrite_a, check_finite])	Compute pivoted LU decomposition of a

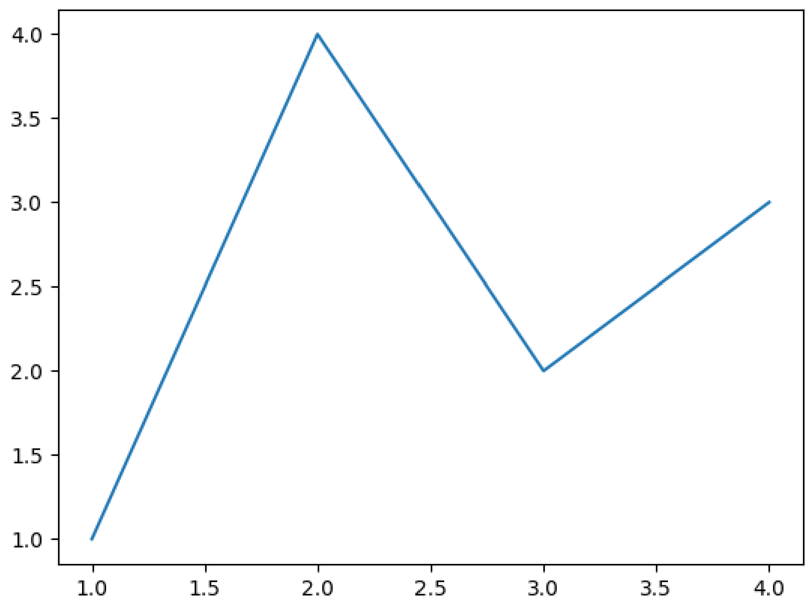
	matrix.
lu_factor(a[, overwrite_a, check_finite])	Compute pivoted LU decomposition of a matrix.
lu_solve(lu_and_piv, b[, trans, ...])	Solve an equation system, a x = b, given the LU factorization of a
svd(a[, full_matrices, compute_uv, ...])	Singular Value Decomposition.

3.3 matplotlib library

```
import matplotlib as mpl

import matplotlib.pyplot as plt

import numpy as np
```

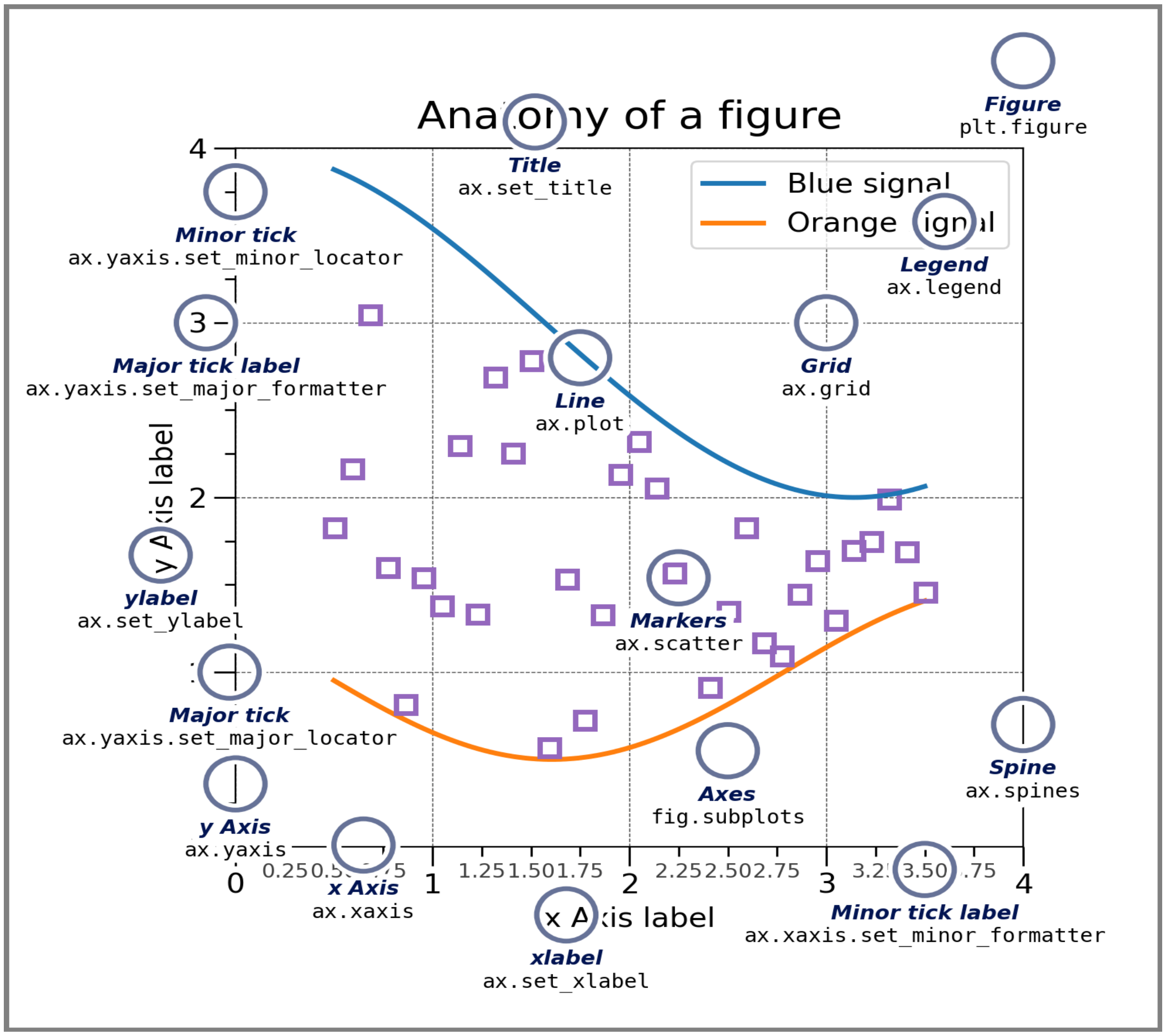
Matplotlib graphs your data on **Figure**s (e.g., windows, Jupyter widgets, etc.), each of which can contain one or more **Axes**, an area where points can be specified in terms of x-y coordinates (or theta-r in a polar plot, x-y-z in a 3D plot, etc.). The simplest way of creating a Figure with an Axes is using **pyplot.subplots**. We can then use **Axes.plot** to draw some data on the Axes:

```
fig,ax=plt.subplots()# Create a figure containing a single axes.
ax.plot([1,2,3,4],[1,4,2,3])# Plot some data on the axes.
```


4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Parts of a Figure

Here are the components of a Matplotlib Figure.



Figure

The **whole** figure. The Figure keeps track of all the child **Axes**, a group of 'special' Artists (titles, figure legends, colorbars, etc), and even nested subfigures.

The easiest way to create a new Figure is with pyplot:

```
fig=plt.figure()# an empty figure with no Axes
fig,ax=plt.subplots()# a figure with a single Axes
fig,axs=plt.subplots(2,2) # a figure with a 2x2 grid of Axes
```

It is often convenient to create the Axes together with the Figure, but you can also manually add Axes later on. Note that many Matplotlib backends support zooming and panning on figure windows.

Axes

An Axes is an Artist attached to a Figure that contains a region for plotting data, and usually includes two (or three in the case of 3D) **Axis** objects (be aware of the difference between **Axes** and **Axis**) that provide ticks and tick labels to provide scales for the data in the Axes. Each **Axes** also has a title (set via **set_title()**), an x-label (set via **set_xlabel()**), and a y-label set via **set_ylabel()**).

The **Axes** class and its member functions are the primary entry point to working with the OOP interface, and have most of the plotting methods defined on them (e.g.ax.plot(), shown above, uses the **plot** method)

Axis

These objects set the scale and limits and generate ticks (the marks on the Axis) and tick labels (strings labeling the ticks). The location of the ticks is determined by a **Locator** object and the ticklabel strings are formatted by a **Formatter**. The combination of the correct **Locator** and **Formatter** gives very fine control over the tick locations and labels.

Artist

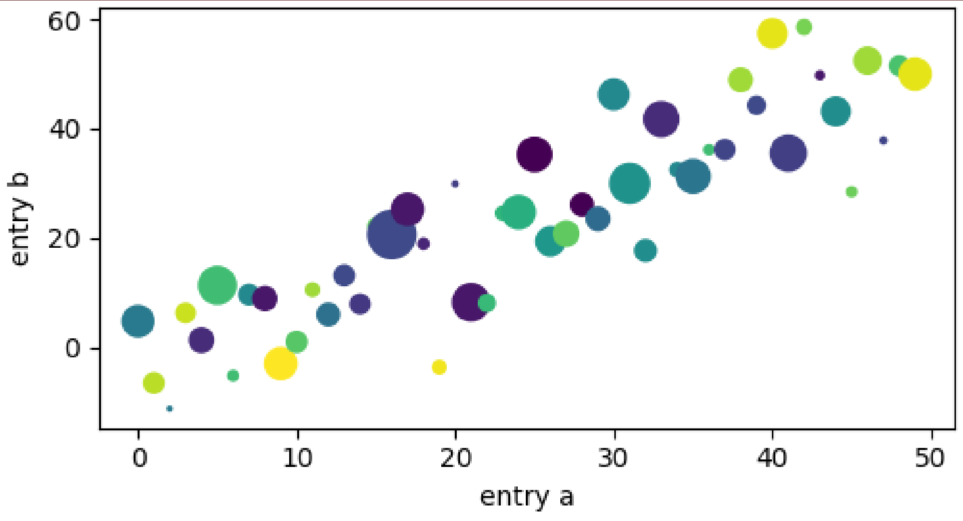
Basically, everything visible on the Figure is an Artist (even **Figure**, **Axes**, and **Axis** objects). This includes **Text** objects, **Line2D** objects, **collections** objects, **Patch** objects, etc. When the Figure is rendered, all of the Artists are drawn to the **canvas**. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes or moved from one to another.

Types of inputs to plotting functions
Plotting functions expect **numpy.array** or **numpy.ma.masked_array** as input, or objects that can be passed to **numpy.asarray**. Classes that are like arrays ('array-like') such as **pandas** data objects and **numpy.matrix** may not work as intended. Common convention is to convert these to **numpy.array** objects prior to plotting. For example, to convert a **numpy.matrix**

```
b=np.matrix([[1,2],[3,4]])
b_asarray=np.asarray(b)
```

Most methods will also parse an addressable object like a *dict*, a **numpy.recarray**, or a **pandas.DataFrame**. Matplotlib allows you to provide the data keyword argument and generate plots passing the strings corresponding to the *x* and *y* variables.

```
np.random.seed(19680801) # seed the random number generator.
data={'a':np.arange(50),'c':np.random.randint(0,50,50),'d':np.random.randn(50)}
data['b']=data['a']+10*np.random.randn(50)
data['d']=np.abs(data['d']) *100

fig,ax=plt.subplots(figsize=(5,2.7),layout='constrained')
ax.scatter('a','b',c='c',s='d',data=data)
ax.set_xlabel('entry a')
ax.set_ylabel('entry b')
```



Coding styles

The explicit and the implicit interfaces

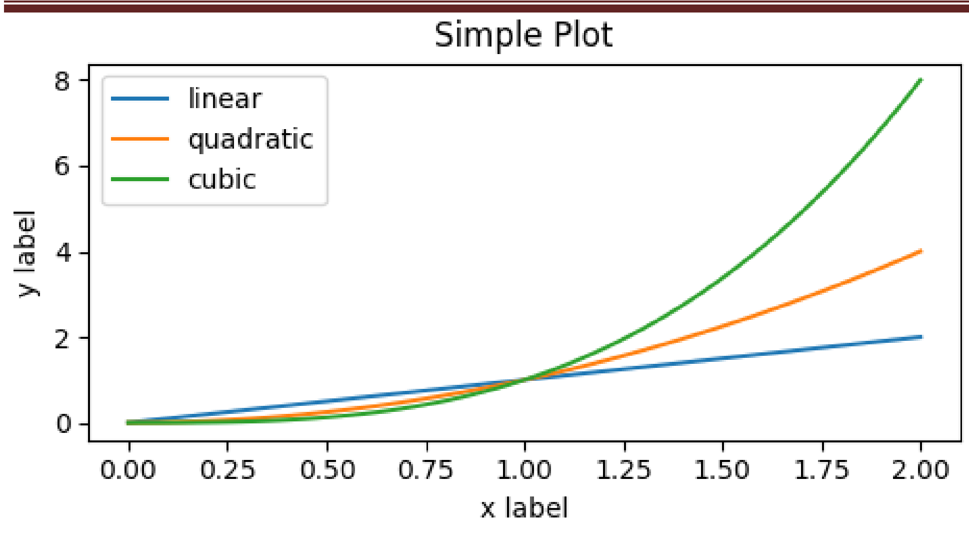
As noted above, there are essentially two ways to use Matplotlib:

- Explicitly create Figures and Axes, and call methods on them (the "object-oriented (OO) style").
- Rely on pyplot to implicitly create and manage the Figures and Axes, and use pyplot functions for plotting.

See Matplotlib Application Interfaces (APIs) for an explanation of the tradeoffs between the implicit and explicit interfaces.

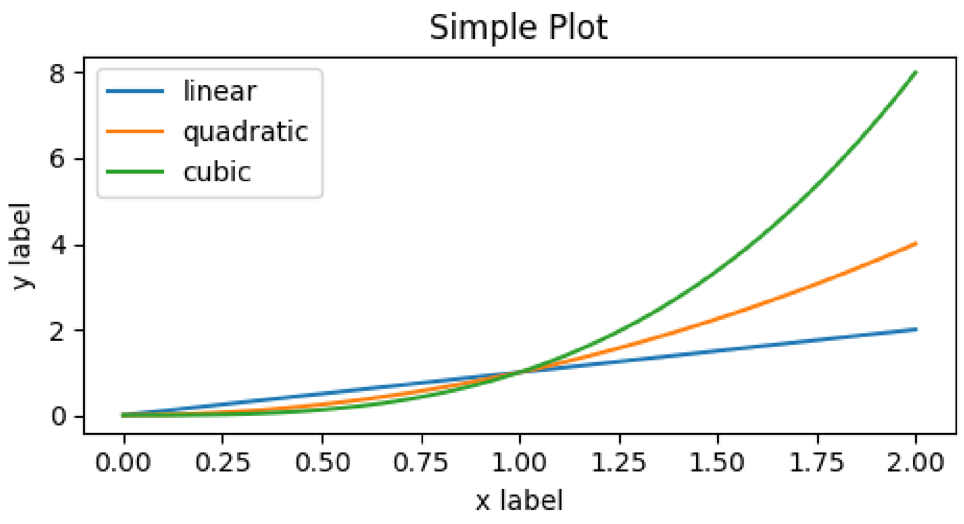
So one can use the OO-style

```
x=np.linspace(0,2,100)# Sample data.

# Note that even in the OO-style, we use `.pyplot.figure` to create the Figure.
fig,ax=plt.subplots(figsize=(5,2.7), layout='constrained')
ax.plot(x,x,label='linear')# Plot some data on the axes.
ax.plot(x,x**2, label='quadratic')# Plot more data on the axes...
ax.plot(x,x**3,label='cubic')# ... and some more.
ax.set_xlabel('x label')# Add an x-label to the axes.
ax.set_ylabel('y label')# Add a y-label to the axes.
ax.set_title("Simple Plot")# Add a title to the axes.
ax.legend()# Add a legend.
```



or the pyplot-style:

```
x=np.linspace(0,2,100)# Sample data.

plt.figure(figsize=(5,2.7),layout='constrained')
plt.plot(x,x,label='linear')# Plot some data on the (implicit) axes.
plt.plot(x,x**2,label='quadratic')# etc.
plt.plot(x,x**3,label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
```



(In addition, there is a third approach, for the case when embedding Matplotlib in a GUI application, which completely drops pyplot, even for figure creation.)

Matplotlib's documentation and examples use both the OO and the pyplot styles. In general, we suggest using the OO style, particularly for complicated plots, and functions and scripts that are intended to be reused as part of a larger project. However, the pyplot style can be very convenient for quick interactive work.

Note

You may find older examples that use the pylab interface, via from pylab import *. This approach is strongly deprecated.

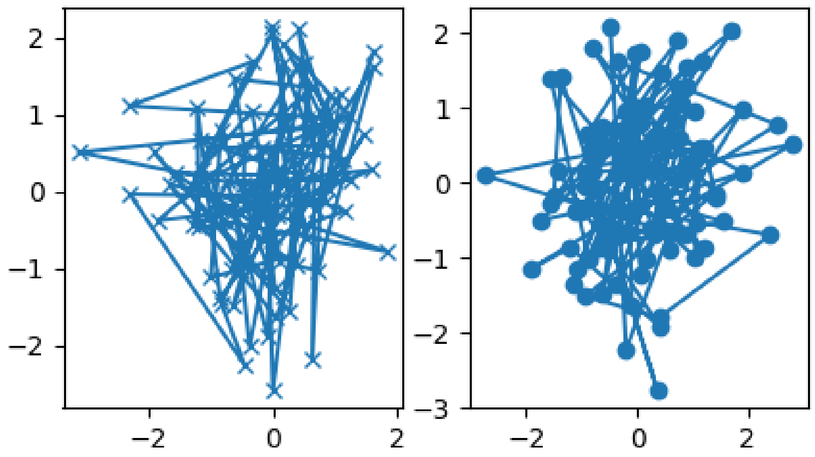
Making a helper functions

If you need to make the same plots repeatedly with different data sets, or want to easily wrap Matplotlib methods, use the recommended signature function below.

```
def my_plotter(ax,data1,data2,param_dict):
"""
   A helper function to make a graph.
   """
out=ax.plot(data1,data2,**param_dict)
returnout
```

which you would then use twice to populate two subplots:

```
data1,data2,data3,data4=np.random.randn(4,100)# make 4 random data sets
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(5,2.7))
my_plotter(ax1,data1,data2,{'marker':'x'})
my_plotter(ax2,data3,data4,{'marker':'o'})
```

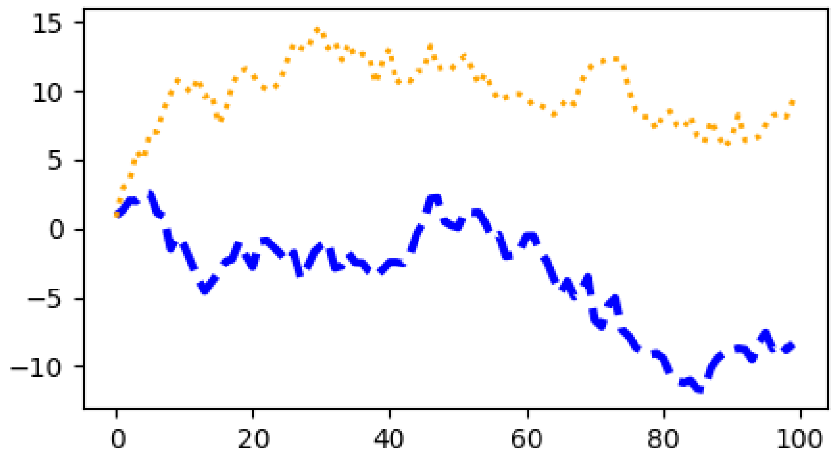


Note that if you want to install these as a python package, or any other customizations you could use one of the many templates on the web; Matplotlib has one at mpl-cookiecutter

Styling Artists

Most plotting methods have styling options for the Artists, accessible either when a plotting method is called, or from a "setter" on the Artist. In the plot below we manually set the *color*, *linewidth*, and *linestyle* of the Artists created by **plot**, and we set the linestyle of the second line after the fact with **set_linestyle**.

```
fig,ax=plt.subplots(figsize=(5,2.7))
x=np.arange(len(data1))
ax.plot(x,np.cumsum(data1),color='blue',linewidth=3,linestyle='--')
l,=ax.plot(x,np.cumsum(data2),color='orange',linewidth=2)
l.set_linestyle(':')
```

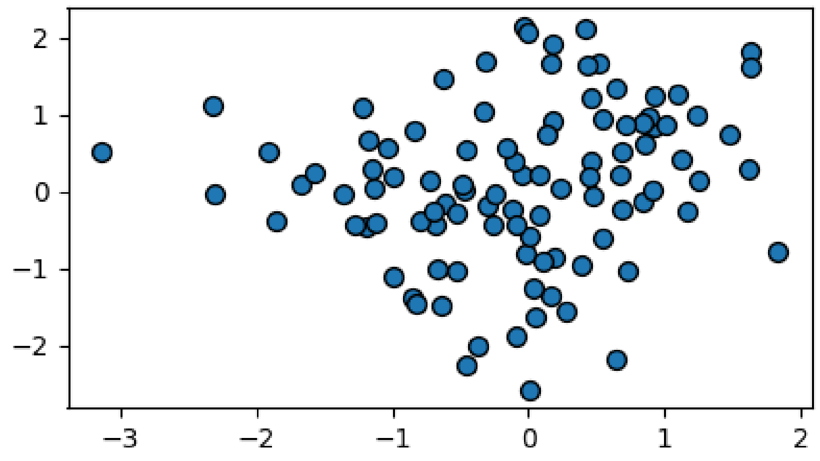


Colors

Matplotlib has a very flexible array of colors that are accepted for most Artists;

Some Artists will take multiple colors. i.e. for a **scatter** plot, the edge of the markers can be different colors from the interior:

```
fig,ax=plt.subplots(figsize=(5,2.7))
ax.scatter(data1,data2,s=50,facecolor='C0',edgecolor='k')
```

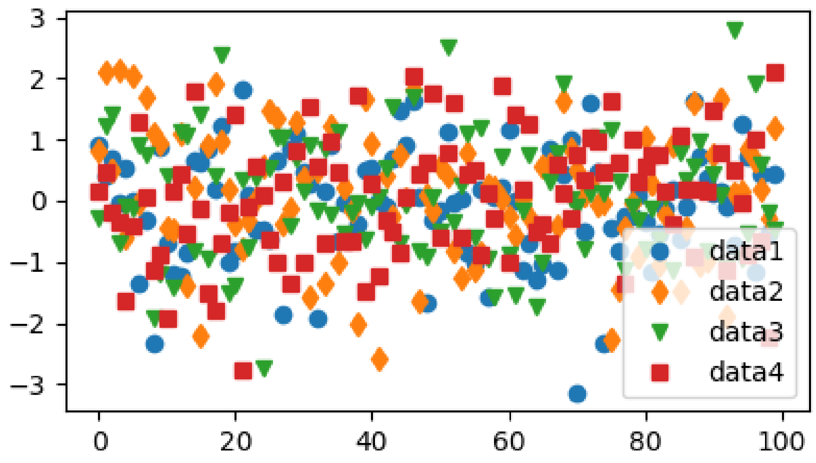


Linewidths, linestyles, and markersizes

Line widths are typically in typographic points (1 pt = 1/72 inch) and available for Artists that have stroked lines. Similarly, stroked lines can have a linestyle.

Marker size depends on the method being used. **plot** specifies markersize in points, and is generally the "diameter" or width of the marker. **scatter** specifies markersize as approximately proportional to the visual area of the marker. There is an array of markerstyles available as string codes, or users can define their own **MarkerStyle**:

```
fig,ax=plt.subplots(figsize=(5,2.7))
ax.plot(data1,'o',label='data1')
ax.plot(data2,'d',label='data2')
ax.plot(data3,'v',label='data3')
ax.plot(data4,'s',label='data4')
ax.legend()
```

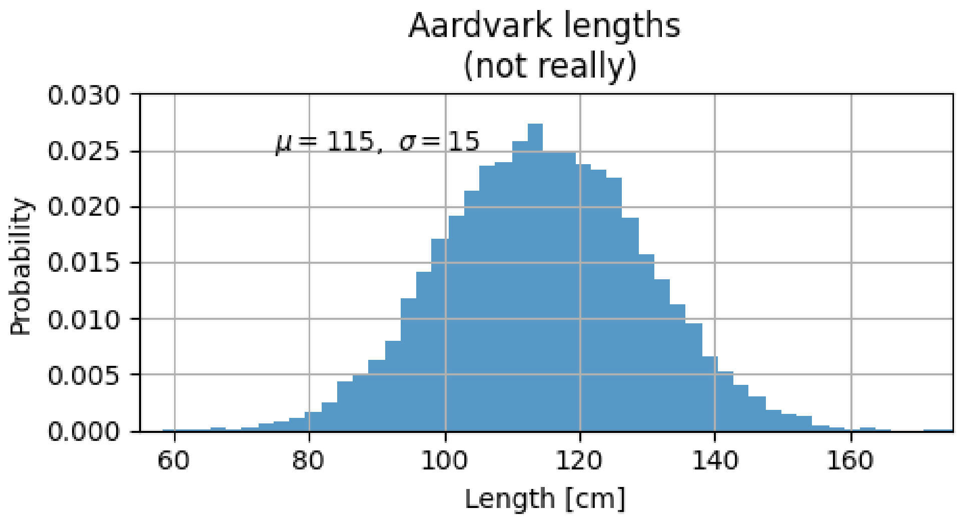


Labelling plots

Axes labels and text

set_xlabel, **set_ylabel**, and **set_title**are used to add text in the indicated locations. Text can also be directly added to plots using **text**:

```
mu,sigma=115,15
x=mu+sigma*np.random.randn(10000)
fig,ax=plt.subplots(figsize=(5,2.7),layout='constrained')
# the histogram of the data
n,bins,patches=ax.hist(x,50,density=True,facecolor='C0',alpha=0.75)

ax.set_xlabel('Length [cm]')
ax.set_ylabel('Probability')
ax.set_title('Aardvark lengths\n (not really)')
ax.text(75,.025,r'$\mu=115,\ \sigma=15$')
ax.axis([55,175,0,0.03])
ax.grid(True)
```



All of the **text** functions return a **matplotlib.text.Text** instance. Just as with lines above, you can customize the properties by passing keyword arguments into the text functions:

```
t=ax.set_xlabel('my data',fontsize=14,color='red')
```

Using mathematical expressions in text

Matplotlib accepts TeX equation expressions in any text expression. For example to write the expression *σi=15* in the title, you can write a TeX expression surrounded by dollar signs:

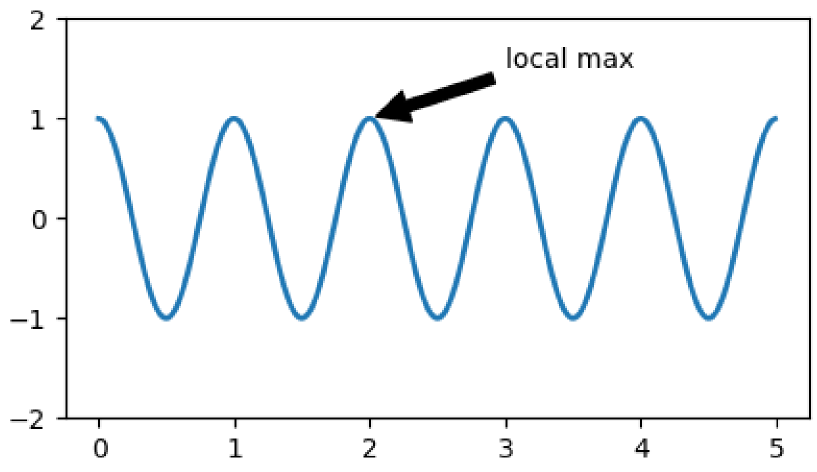
```
ax.set_title(r'$\sigma_i=15$')
```

where the r preceding the title string signifies that the string is a *raw* string and not to treat backslashes as python escapes. Matplotlib has a built-in TeX expression parser and layout engine, and ships its own math fonts

Annotations

We can also annotate points on a plot, often by connecting an arrow pointing to *xy*, to a piece of text at *xytext*:

```
fig,ax=plt.subplots(figsize=(5,2.7))

t=np.arange(0.0,5.0,0.01)
s=np.cos(2*np.pi*t)
line,=ax.plot(t,s,lw=2)

ax.annotate('local max',xy=(2,1),xytext=(3,1.5),
arrowprops=dict(facecolor='black',shrink=0.05))

ax.set_ylim(-2,2)
```

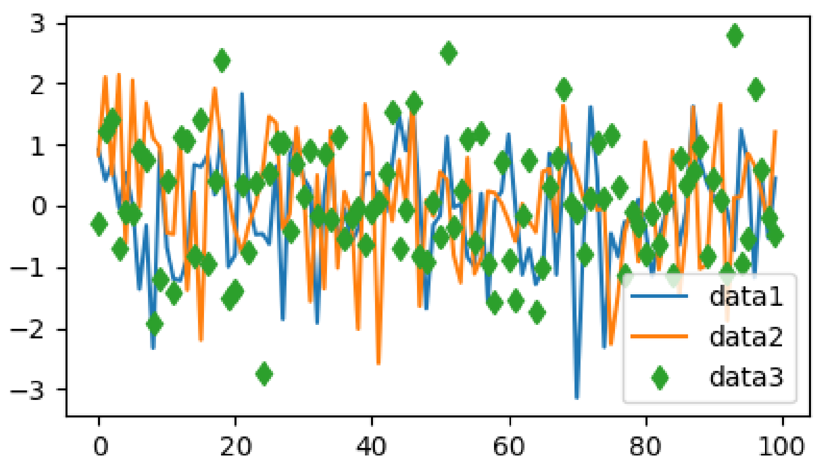


In this basic example, both *xy* and *xytext* are in data coordinates. There are a variety of other coordinate systems one can choose.

Legends

Often we want to identify lines or markers with a **Axes.legend**:

```
fig,ax=plt.subplots(figsize=(5,2.7))
ax.plot(np.arange(len(data1)),data1,label='data1')
ax.plot(np.arange(len(data2)),data2,label='data2')
ax.plot(np.arange(len(data3)),data3,'d',label='data3')
ax.legend()
```



Legends in Matplotlib are quite flexible in layout, placement, and what Artists they can represent.

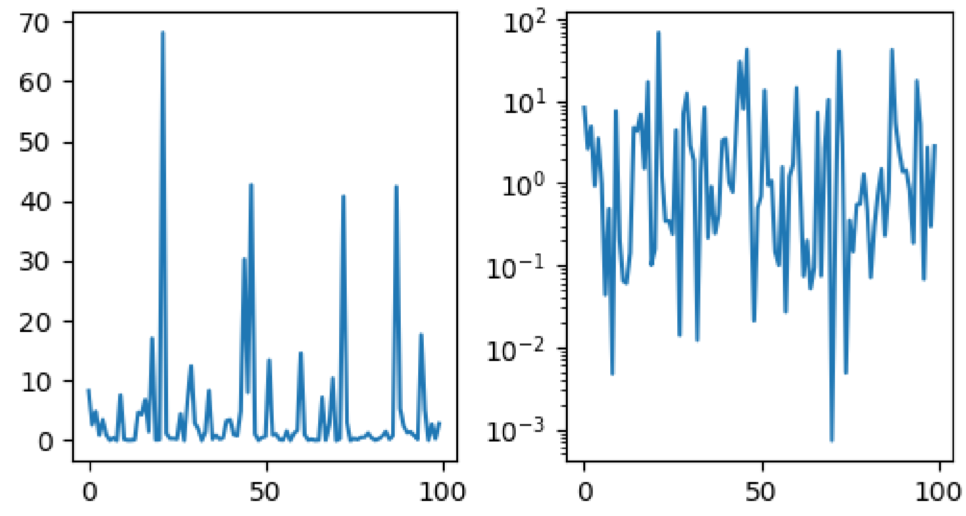
Axis scales and ticks

Each Axes has two (or three) **Axis** objects representing the x- and y-axis. These control the *scale* of the Axis, the tick *locators* and the tick *formatters*. Additional Axes can be attached to display further Axis objects.

Scales

In addition to the linear scale, Matplotlib supplies non-linear scales, such as a log-scale. Since log-scales are used so much there are also direct methods like **loglog**, **semilogx**, and **semilogy**. There are several scales. Here we set the scale manually:

```
fig,axs=plt.subplots(1,2,figsize=(5,2.7),layout='constrained')
xdata=np.arange(len(data1))# make an ordinal for this
data=10**data1
axs[0].plot(xdata,data)

axs[1].set_yscale('log')
axs[1].plot(xdata,data)
```

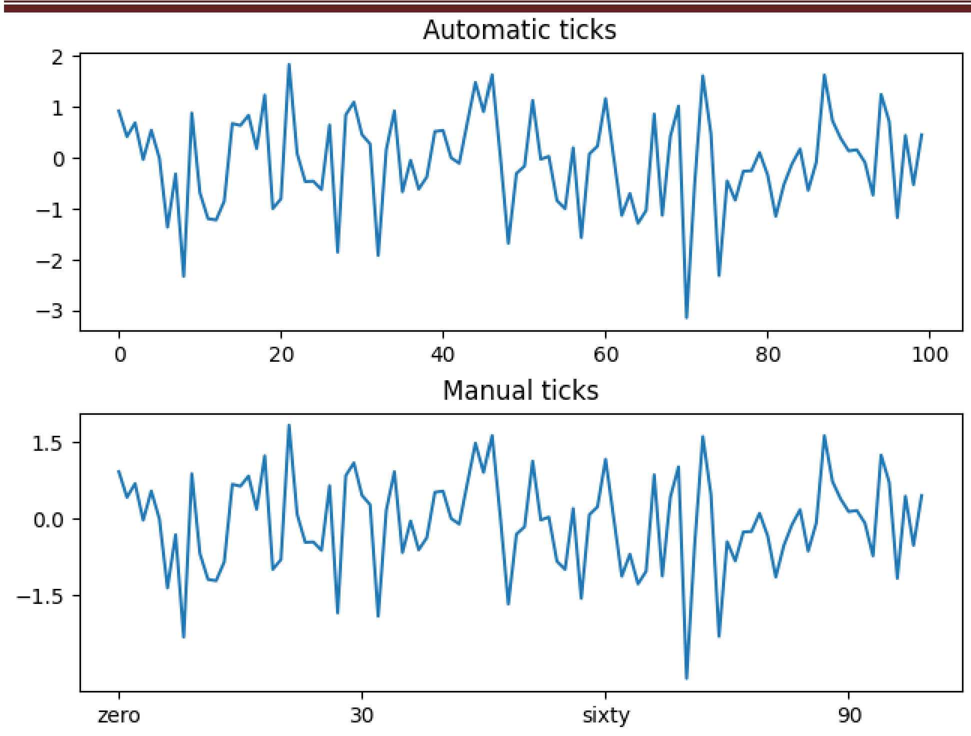


The scale sets the mapping from data values to spacing along the Axis. This happens in both directions, and gets combined into a *transform*, which is the way that Matplotlib maps from data coordinates to Axes, Figure, or screen coordinates.

Tick locators and formatters

Each Axis has a tick *locator* and *formatter* that choose where along the Axis objects to put tick marks. A simple interface to this is **set_xticks**:

```
fig,axs=plt.subplots(2,1,layout='constrained')
axs[0].plot(xdata,data1)
axs[0].set_title('Automatic ticks')

axs[1].plot(xdata,data1)
axs[1].set_xticks(np.arange(0,100,30),['zero','30','sixty','90'])
axs[1].set_yticks([-1.5,0,1.5])# note that we don't need to specify labels
axs[1].set_title('Manual ticks')
```

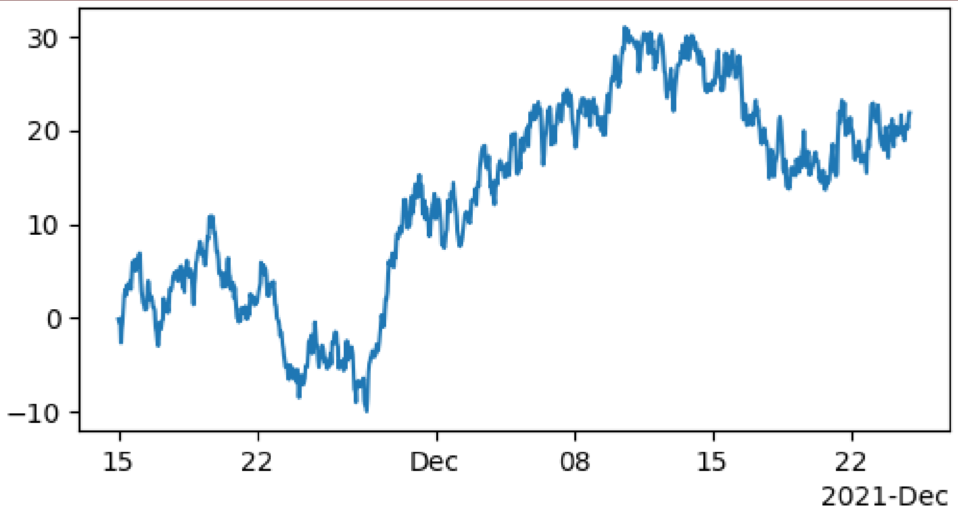


Different scales can have different locators and formatters; for instance, the log-scale above uses **LogLocator** and **LogFormatter**.

Plotting dates and strings

Matplotlib can handle plotting arrays of dates and arrays of strings, as well as floating point numbers. These get special locators and formatters as appropriate. For dates:

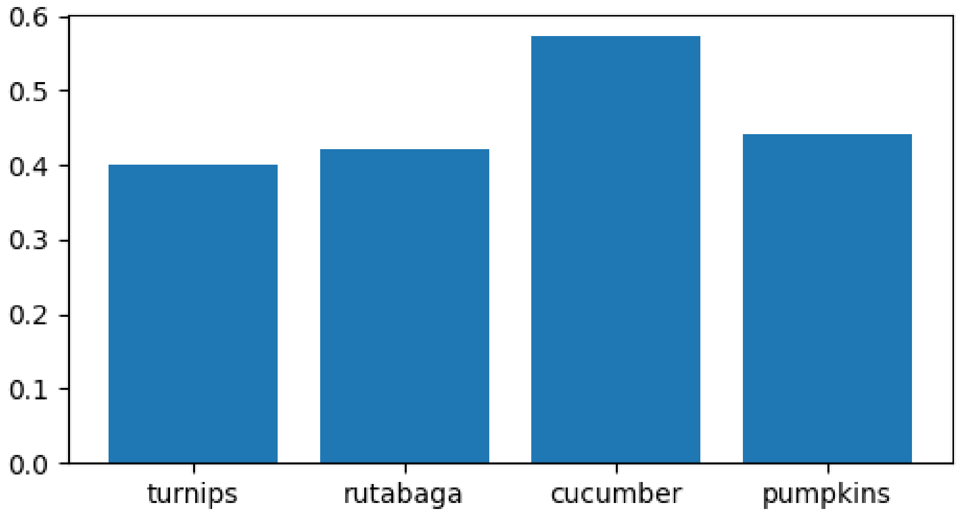
```
fig,ax=plt.subplots(figsize=(5,2.7),layout='constrained')
dates=np.arange(np.datetime64('2021-11-15'),np.datetime64('2021-12-25'),
np.timedelta64(1,'h'))
data=np.cumsum(np.random.randn(len(dates)))
ax.plot(dates,data)
cdf=mpl.dates.ConciseDateFormatter(ax.xaxis.get_major_locator())
ax.xaxis.set_major_formatter(cdf)
```



For more information see the date examples (e.g. Date tick labels)

For strings, we get categorical plotting

```
fig, ax=plt.subplots(figsize=(5,2.7),layout='constrained')
categories=['turnips','rutabaga','cucumber','pumpkins']

ax.bar(categories,np.random.rand(len(categories)))
```



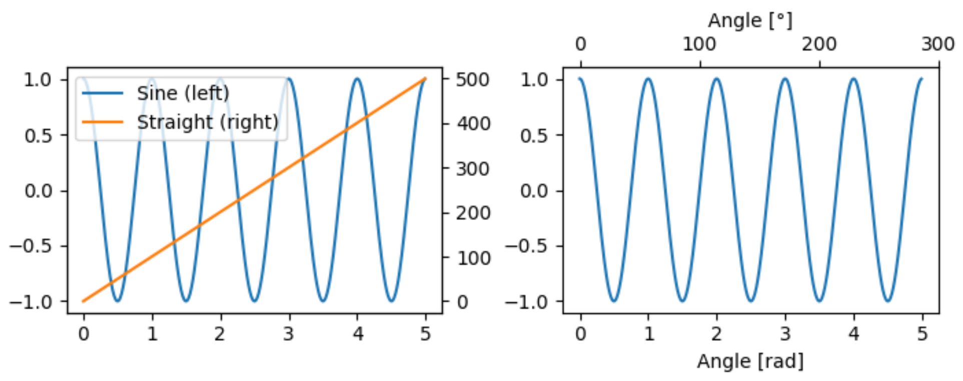
One caveat about categorical plotting is that some methods of parsing text files return a list of strings, even if the strings all represent numbers or dates. If you pass 1000 strings, Matplotlib will think you meant 1000 categories and will add 1000 ticks to your plot!

Additional Axis objects

Plotting data of different magnitude in one chart may require an additional y-axis. Such an Axis can be created by using **twinx** to add a new Axes with an invisible x-axis and a y-axis positioned at the right (analogously for **twiny**).

Similarly, you can add a **secondary_xaxis** or **secondary_yaxis** having a different scale than the main Axis to represent the data in different scales or units.

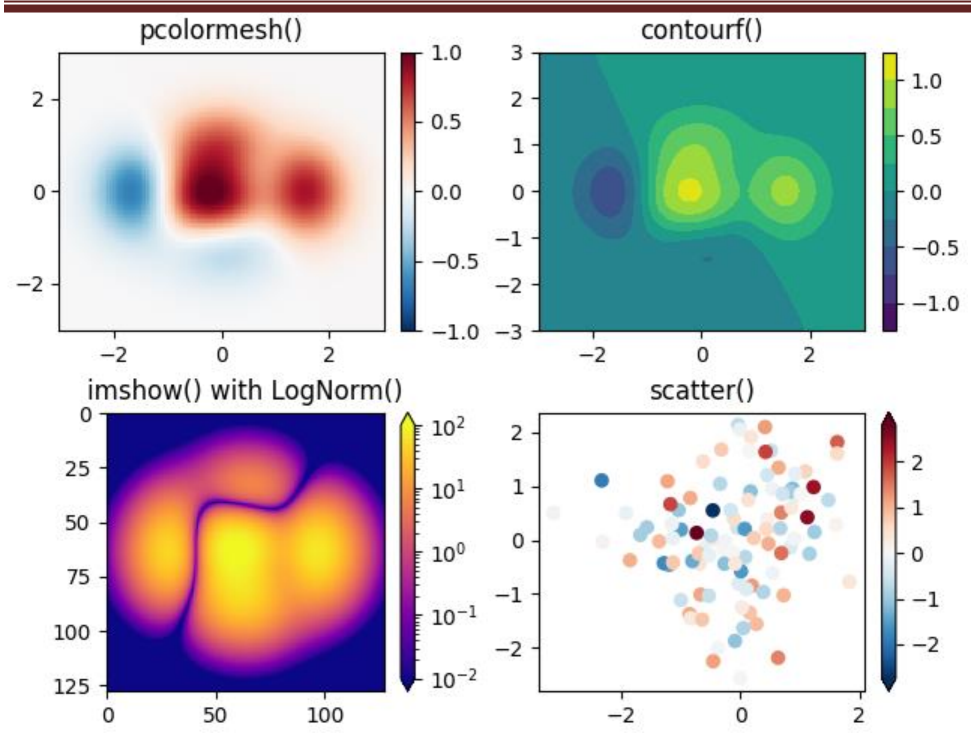
```
fig,(ax1,ax3)=plt.subplots(1,2, figsize=(7,2.7),layout='constrained')
l1,=ax1.plot(t,s)
ax2=ax1.twinx()
l2,=ax2.plot(t,range(len(t)),'C1')
ax2.legend([l1,l2],['Sine (left)','Straight (right)'])

ax3.plot(t, s)
ax3.set_xlabel('Angle [rad]')
ax4=ax3.secondary_xaxis('top',functions=(np.rad2deg,np.deg2rad))
ax4.set_xlabel('Angle [°]')
```



Color mapped data

Often we want to have a third dimension in a plot represented by a colors in a colormap. Matplotlib has several plot types that do this:

```
X,Y=np.meshgrid(np.linspace(-3,3,128),np.linspace(-3,3,128))
Z=(1-X/2+X**5+Y**3)*np.exp(-X**2-Y**2)

fig,axs=plt.subplots(2,2,layout='constrained')
pc=axs[0,0].pcolormesh(X,Y,Z,vmin=-1,vmax=1,cmap='RdBu_r')
fig.colorbar(pc,ax=axs[0,0])
axs[0,0].set_title('pcolormesh()')

co=axs[0,1].contourf(X,Y,Z,levels=np.linspace(-1.25,1.25,11))
fig.colorbar(co,ax=axs[0,1])
axs[0,1].set_title('contourf()')

pc=axs[1,0].imshow(Z**2*100,cmap='plasma',
norm=mpl.colors.LogNorm(vmin=0.01,vmax=100))
fig.colorbar(pc,ax=axs[1,0],extend='both')
axs[1,0].set_title('imshow() with LogNorm()')

pc=axs[1,1].scatter(data1,data2,c=data3,cmap='RdBu_r')
fig.colorbar(pc,ax=axs[1,1], extend='both')
axs[1,1].set_title('scatter()')
```



Colormaps

These are all examples of Artists that derive from **ScalarMappable** objects. They all can set a linear mapping between *vmin* and *vmax* into the colormap specified by *cmap*. Matplotlib has many colormaps to choose from (Choosing Colormaps in Matplotlib) you can make your own Normalizations

Sometimes we want a non-linear mapping of the data to the colormap, as in the LogNorm example above. We do this by supplying the ScalarMappable with the *norm* argument instead of *vmin* and *vmax*.

Colorbars

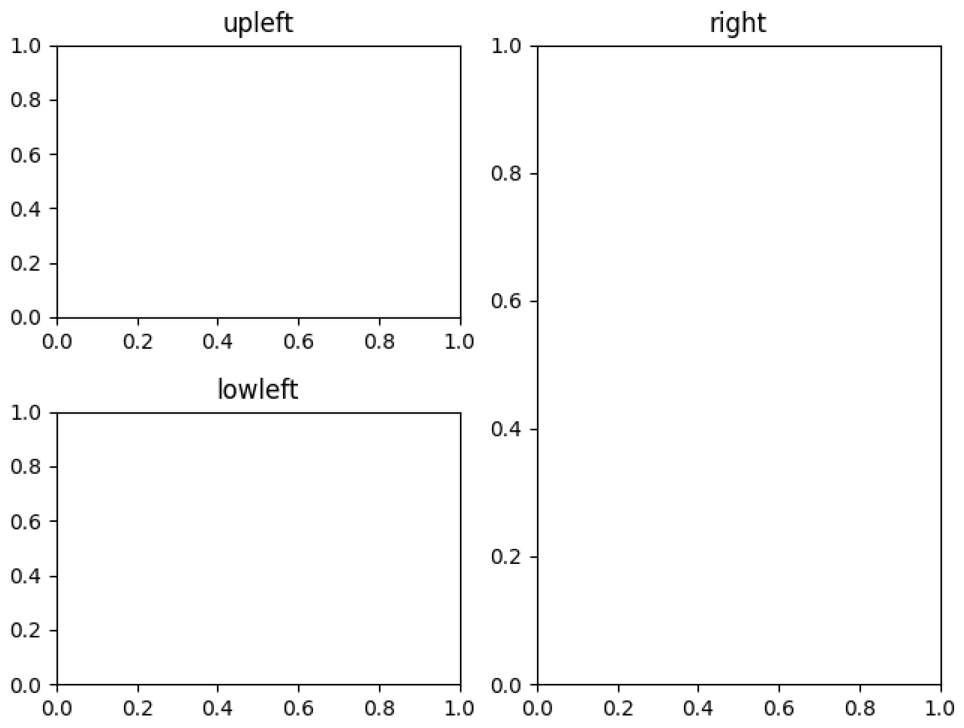
Adding a **colorbar** gives a key to relate the color back to the underlying data. Colorbars are figure-level Artists, and are attached to a ScalarMappable (where they get their information about the norm and colormap) and usually steal space from a parent Axes. You can also change the appearance of colorbars with the *extend* keyword to add arrows to the ends, and*shrink* and *aspect* to control the size. Finally, the colorbar will have default locators and formatters appropriate to the norm. These can be changed as for other Axis objects.

Working with multiple Figures and Axes

You can open multiple Figures with multiple calls to fig = plt.figure() or fig2, ax = plt.subplots(). By keeping the object references you can add Artists to either Figure.

Multiple Axes can be added a number of ways, but the most basic is plt.subplots() as used above. One can achieve more complex layouts, with Axes objects spanning columns or rows, using **subplot_mosaic**.

```
fig,axd=plt.subplot_mosaic([['upleft','right'],
['lowleft','right']],layout='constrained')
axd['upleft'].set_title('upleft')
axd['lowleft'].set_title('lowleft')
axd['right'].set_title('right')
```



3.4 Introduction to sympy library

Run SymPy

After installation, it is best to verify that your freshly-installedSymPy works. To do this, start up Python and import the SymPy libraries:

```
$ python
>>> from sympy import *
```

From here, execute some simple SymPy statements like the ones below:

```
>>>x = Symbol('x')
>>> limit(sin(x)/x, x, 0)
1
```

```
>>>integrate(1/x, x)
log(x)
```

Symbolic computation deals with the computation of mathematical objects symbolically. This means that the mathematical objects are represented exactly, not approximately, and mathematical expressions with unevaluated variables are left in symbolic form.

Let’s take an example. Say we wanted to use the built-in Python functions to compute square roots. We might do something like this

```
>>>importmath
>>>math.sqrt(9)
3.0
```

9 is a perfect square, so we got the exact answer, 3. But suppose we computed the square root of a number that isn’t a perfect square

```
>>>math.sqrt(8)
2.82842712475
```

Here we got an approximate result. 2.82842712475 is not the exact square root of 8 (indeed, the actual square root of 8 cannot be represented by a finite decimal, since it is an irrational number). If all we cared about was the decimal form of the square root of 8, we would be done.

But suppose we want to go further. Recall that *8=4⋅2=22*. We would have a hard time deducing this from the above result. This is where symbolic computation comes in. With a symbolic computation system like SymPy, square roots of numbers that are not perfect squares are left unevaluated by default

```
>>>import sympy
>>>sympy.sqrt(3)
sqrt(3)
```

Furthermore—and this is where we start to see the real power of symbolic computation—symbolic results can be symbolically simplified.

```
>>>sympy.sqrt(8)
2*sqrt(2)
```

Another Example

The above example starts to show how we can manipulate irrational numbers exactly using SymPy. But it is much more powerful than that. Symbolic computation systems (which by the way, are also often called computer algebra systems, or just CASs) such as SymPy are capable of computing symbolic expressions with variables.

As we will see later, in SymPy, variables are defined using symbols. Unlike many symbolic manipulation systems, variables in SymPy must be defined before they are used

Let us define a symbolic expression, representing the mathematical expression *x+2y*.

```
>>>from sympy import symbols
>>> x, y =symbols('x y')
>>> expr = x +2*y
>>> expr
x + 2*y
```

Note that we wrote x + 2*y just as we would if x and y were ordinary Python variables. But in this case, instead of evaluating to something, the expression remains as just x + 2*y. Now let us play around with it:

```
>>>expr +1
x + 2*y + 1
>>>expr - x
2*y
```

Notice something in the above example. When we typed expr - x, we did not get x + 2*y - x, but rather just 2*y. The x and the -x automatically canceled one another. This is similar to how sqrt(8) automatically turned into 2*sqrt(2) above. This isn't always the case in SymPy, however:

```
>>>x*expr
x*(x + 2*y)
```

Here, we might have expected *x(x+2y)* to transform into *x2+2xy*, but instead we see that the expression was left alone. This is a common theme in SymPy. Aside from obvious simplifications like *x−x=0* and *8=22*, most simplifications are not performed automatically. This is because we might prefer the factored form *x(x+2y)*, or we might prefer the expanded form *x2+2xy*. Both forms are useful in different circumstances. In SymPy, there are functions to go from one form to the other

```
>>>from sympy import expand, factor
>>>expanded_expr= expand(x*expr)
```

```
>>>expanded_expr
x**2 + 2*x*y
>>> factor(expanded_expr)
x*(x + 2*y)
```

The Power of Symbolic Computation

The real power of a symbolic computation system such as SymPy is the ability to do all sorts of computations symbolically. SymPy can simplify expressions, compute derivatives, integrals, and limits, solve equations, work with matrices, and much, much more, and do it all symbolically. It includes modules for plotting, printing (like 2D pretty printed output of math formulas, or *LATEX*), code generation, physics, statistics, combinatorics, number theory, geometry, logic, and more. Here is a small sampling of the sort of symbolic power SymPy is capable of, to whet your appetite.

```
>>>from sympy import*
>>>x, t, z, nu =symbols('x t z nu')
```

This will make all further examples pretty print with unicode characters.

```
>>>init_printing(use_unicode=True)
```

Take the derivative of $\sin(x)e^{x}$.

```
>>>diff(sin(x)*exp(x), x)
 x        x
e ·sin(x) + e ·cos(x)
```

Compute $\int(e^{x}\sin(x)+e^{x}\cos(x))dx$.

```
>>>integrate(exp(x)*sin(x) + exp(x)*cos(x), x)

eˣ ·sin(x)
```

Compute $\int_{-\infty}^{\infty} \sin(x^2)\ dx$

```
>>>integrate(sin(x**2), (x, -oo, oo))
√2√π
────
```

```
  2
```

Find $lim_{x\to 0} sin(x)/x$.

```
>>>limit(sin(x)/x, x, 0)
1
```

Solve $x^2-2=0$.

```
>>>solve(x**2-2, x)
[-√2, √2]
```

Find the eigenvalues of $[1\ 2$

$2\ 2]$.

```
>>>Matrix([[1, 2], [2, 2]]).eigenvals()
⎧3   √17        3   √17   ⎫
⎨─ - ───: 1, ─ + ───: 1⎬
⎩2    2         2    2    ⎭
```

EX NO. 1

Introduction to Anaconda &Jupyter Notebook setup and evaluating elementary functions.

Anaconda is an open-source package manager for Python and R. It is the most popular platform among data science professionals for running Python and R implementations. There are over 300 libraries in data science, so having a robust distribution system for them is a must for any professional in this field.

Anaconda simplifies package deployment and management. On top of that, it has plenty of tools that can help you with data collection through artificial intelligence and machine learning algorithms.

With Anaconda, you can easily set up, manage, and share Conda environments. Moreover, you can deploy any required project with a few clicks when you're using Anaconda.

Uses of Anaconda?

There are many advantages to using Anaconda and following are the most prominent ones among them:

- Anaconda is free and open-source. This means you can use it without spending any money.
- In the data science sector, Anaconda is an industry staple. It is open-source too, which has made it widely popular. If you want to become a data science professional, you must know how to use Anaconda for Python because every recruiter expects you to have this skill. It is a must-have for data science.
- It has more than 1500 Python and R data science packages, so you don't face any compatibility issues while collaborating with others. For example, suppose your colleague sends you a project which requires packages called A and B but you only have package A. Without having the package B, you wouldn't be able to run the project. Anaconda mitigates the chances of such errors. You can easily collaborate on projects without worrying about any compatibility issues.
- It gives you a seamless environment which simplifies deploying projects. You can deploy any project with just a few clicks and commands while managing the rest.
- Anaconda has a thriving community of data scientists and machine learning professionals who use it regularly. If you encounter an issue, chances are, the community has already answered the same. On the other hand, you can also ask people in the community about the issues you face there, it's a very helpful community ready to help new learners.
- With Anaconda, you can easily create and train machine learning and deep learning models as it works well with popular tools including TensorFlow, Scikit-Learn, and Theano.
- You can create visualizations by using Bokeh, Holoviews, Matplotlib, and Datashader while using Anaconda.
- Anaconda works with all major Python libraries including Dask, Pandas, NumPy and Numba that allow you to analyze data quickly and scalably.

Installing Anaconda?

You can download Anaconda for your system from the official website of Anaconda.

You should download the version that matches your device's compatibility as Anaconda is available for both 64-bit and 32-bit machines.

After the download is complete, open the download.exe setup and click the 'Next' button. The installer would ask you to read the agreement and you will have to click on 'I Agree' to proceed.

In the next window, the installer asks you if you want to download the software for all users or just yourself. Note that if you want to install Anaconda for all users, you'll need administrator privileges, which can make things complicated.

In the following section, the installer asks you for the destination of the software. Here, you can choose the place where you want Anaconda to be installed.

Now, the installer allows you to add Anaconda to your machine's PATH environment variable and register as the primary system Python 3.8. By adding it to PATH, you ensure that it gets found before another installer. Now, you can click the 'Install' button and start the installation process.

After the installer has completed the extraction of Anaconda and its related files, you will have to click the Next button after which the installer informs you about PyCharm.

After that window, your installation process has completed. You can click on the Finish button to end the task or learn more about Anaconda Cloud through the installer's final window.

Steps After the Installation

After you have completed the installation, you can search your system for Anaconda, which would show you the following files:

- The Anaconda Prompt
- The Jupyter Notebook
- Anaconda Powershell Prompt
- Spyder IDE
- Anaconda Navigator

Now, go to the command prompt and type 'Jupyter notebook' so it would open the Jupyter dashboard.

At the top right of the menu, you’ll find the option to create a new notebook. In your new notebook, you can execute one or multiple statements at once and start working.

Using Anaconda for Python

Let’s discuss some fundamental commands you can use to start using this package manager.

Listing All Environments

To begin using Anaconda, you’d need to see how many Conda environments are present in your machine.

```
conda env list
```

It will list all the available Conda environments in your machine.

Creating a New Environment

You can create a new Conda environment by going to the required directory and use this command:

```
conda create -n <your_environment_name>
```

You can replace <your_environment_name> with the name of your environment. After entering this command, conda will ask you if you want to proceed to which you should reply with y:

```
proceed ([y])/n)?
```

On the other hand, if you want to create an environment with a particular version of Python, you should use the following command:

```
conda create -n <your_environment_name> python=3.6
```

Similarly, if you want to create an environment with a particular package, you can use the following command:

```
conda create -n <your_environment_name>pack_name
```

Here, you can replace pack_name with the name of the package you want to use.

If you have a .yml file, you can use the following command to create a new Conda environment based on that file:

```
conda env create -n <your_environment_name> -f <file_name>.yml
```

We have also discussed how you can export an existing Conda environment to a .yml file later in this article.

Activating an Environment

You can activate a Conda environment by using the following command:

```
conda activate <environment_name>
```

You should activate the environment before you start working on the same. Also, replace the term <environment_name> with the environment name you want to activate. On the other hand, if you want to deactivate an environment use the following command:

```
conda deactivate
```

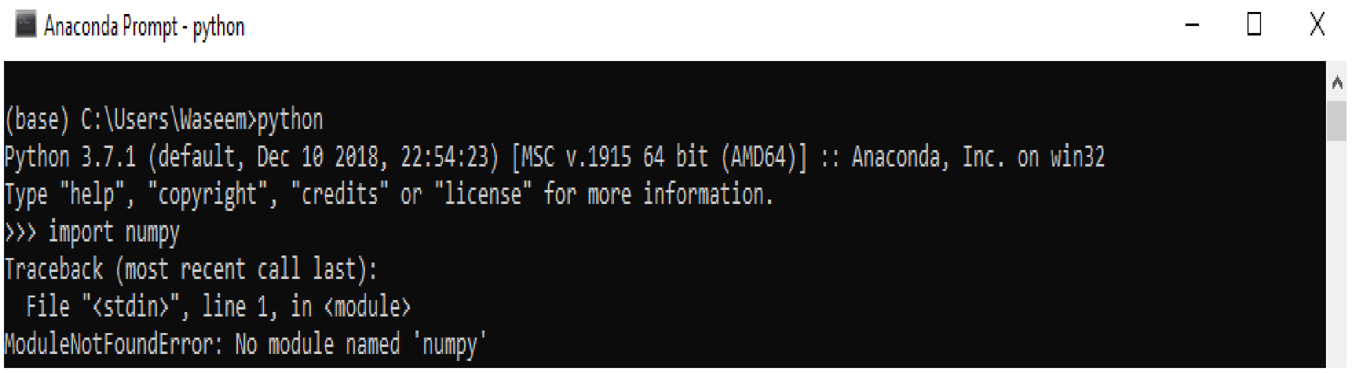
Installing Packages in an Environment

Now that you have an activated environment, you can install packages into it by using the following command:

```
conda install <pack_name>
```

Replace the term <pack_name> with the name of the package you want to install in your Conda environment while using this command.

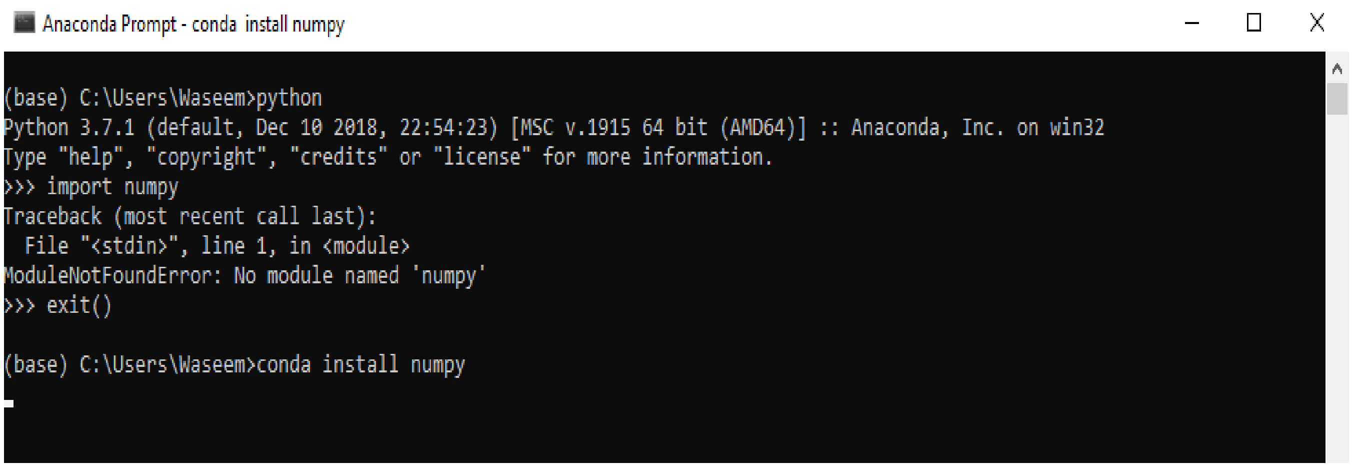
Open anaconda prompt and check if the library is already installed or not.



Anaconda Prompt - python

```
(base) C:\Users\Waseem>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'numpy'
```

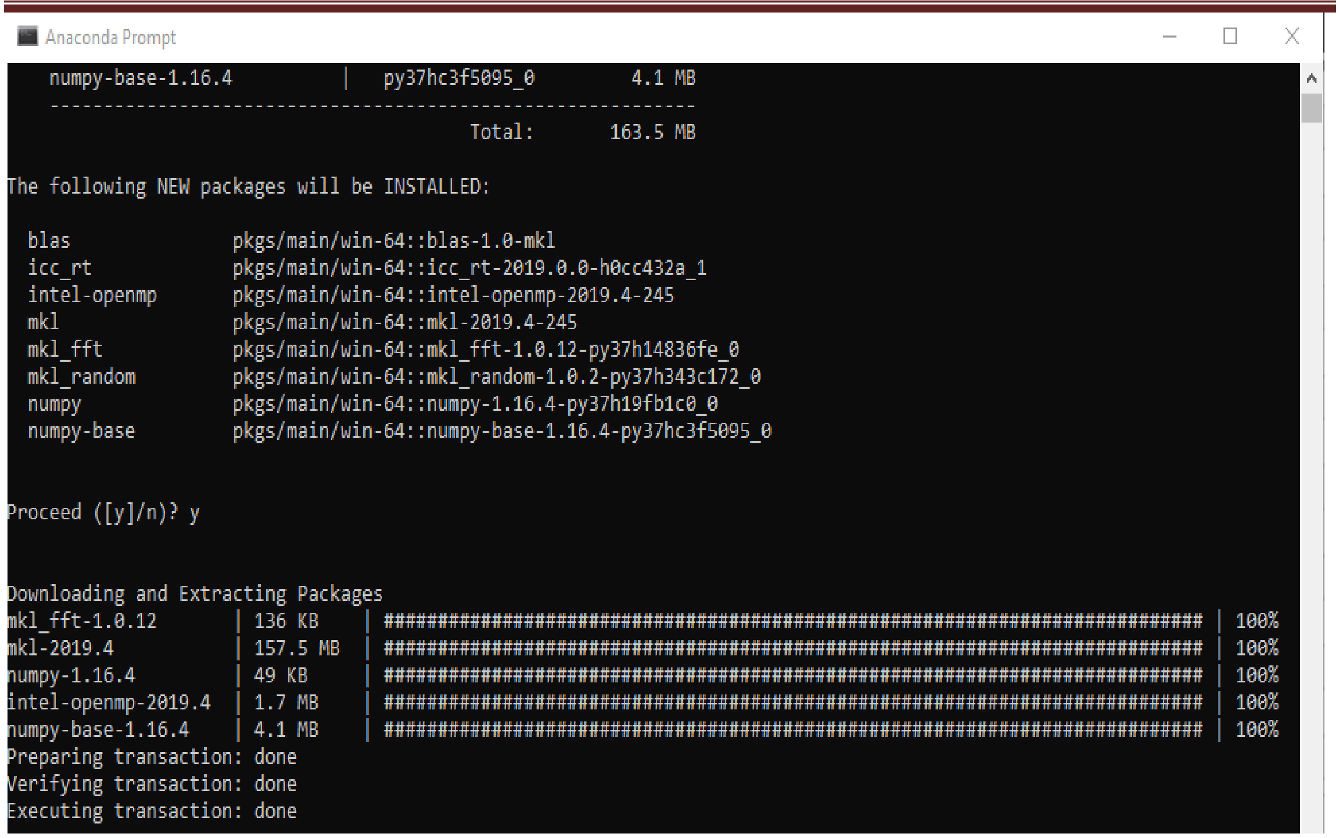
Since there is no module named numpy present, we will run the following command to install numpy.



Anaconda Prompt - conda install numpy

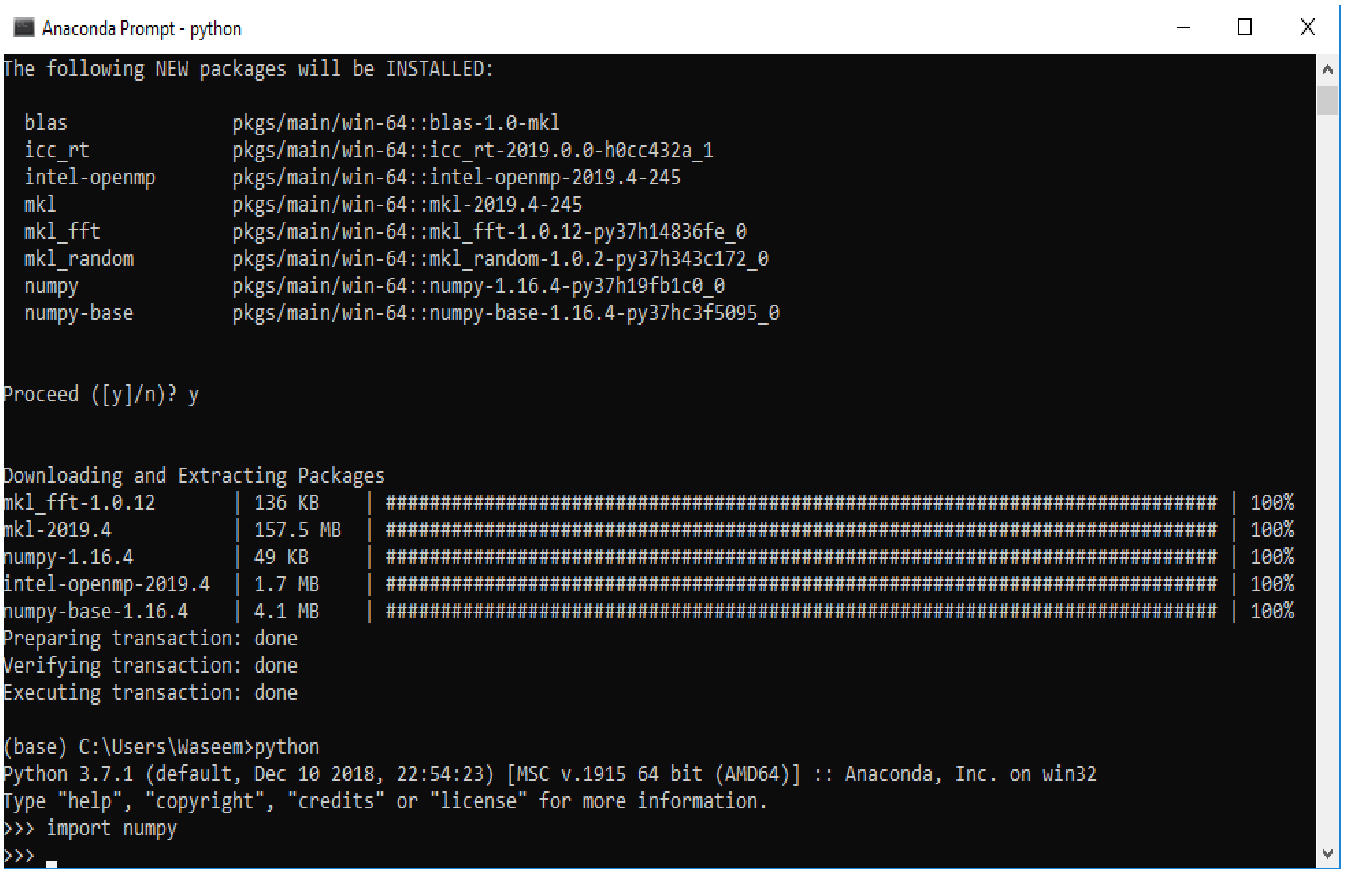
```
(base) C:\Users\Waseem>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'numpy'
>>> exit()

(base) C:\Users\Waseem>conda install numpy
```

You will get the window shown in the image once you complete the installation.


Anaconda Prompt

```
   numpy-base-1.16.4          |   py37hc3f5095_0         4.1 MB
   ------------------------------------------------------------
                                          Total:       163.5 MB

The following NEW packages will be INSTALLED:

  blas               pkgs/main/win-64::blas-1.0-mkl
  icc_rt             pkgs/main/win-64::icc_rt-2019.0.0-h0cc432a_1
  intel-openmp       pkgs/main/win-64::intel-openmp-2019.4-245
  mkl                pkgs/main/win-64::mkl-2019.4-245
  mkl_fft            pkgs/main/win-64::mkl_fft-1.0.12-py37h14836fe_0
  mkl_random         pkgs/main/win-64::mkl_random-1.0.2-py37h343c172_0
  numpy              pkgs/main/win-64::numpy-1.16.4-py37h19fb1c0_0
  numpy-base         pkgs/main/win-64::numpy-base-1.16.4-py37hc3f5095_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
mkl_fft-1.0.12       | 136 KB    | ############################################################################ | 100%
mkl-2019.4           | 157.5 MB  | ############################################################################ | 100%
numpy-1.16.4         | 49 KB     | ############################################################################ | 100%
intel-openmp-2019.4  | 1.7 MB    | ############################################################################ | 100%
numpy-base-1.16.4    | 4.1 MB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Once you have installed a library, just try to import the module again for assurance.


Anaconda Prompt - python

```
The following NEW packages will be INSTALLED:

  blas               pkgs/main/win-64::blas-1.0-mkl
  icc_rt             pkgs/main/win-64::icc_rt-2019.0.0-h0cc432a_1
  intel-openmp       pkgs/main/win-64::intel-openmp-2019.4-245
  mkl                pkgs/main/win-64::mkl-2019.4-245
  mkl_fft            pkgs/main/win-64::mkl_fft-1.0.12-py37h14836fe_0
  mkl_random         pkgs/main/win-64::mkl_random-1.0.2-py37h343c172_0
  numpy              pkgs/main/win-64::numpy-1.16.4-py37h19fb1c0_0
  numpy-base         pkgs/main/win-64::numpy-base-1.16.4-py37hc3f5095_0


Proceed ([y]/n)? y


Downloading and Extracting Packages
mkl_fft-1.0.12       | 136 KB    | ############################################################################ | 100%
mkl-2019.4           | 157.5 MB  | ############################################################################ | 100%
numpy-1.16.4         | 49 KB     | ############################################################################ | 100%
intel-openmp-2019.4  | 1.7 MB    | ############################################################################ | 100%
numpy-base-1.16.4    | 4.1 MB    | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(base) C:\Users\Waseem>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> _
```

As you can see, there is no error that we got in the beginning, so this is how we can install various libraries in anaconda.

Updating Packages in an Environment

If you want to update the packages present in a particular Conda environment, you should use the following command:

```
conda update
```

The above command will update all the packages present in the environment. However, if you want to update a package to a certain version, you will need to use the following command:

```
conda install <package_name>=<version>
```

Exporting an Environment Configuration

Suppose you want to share your project with someone else (colleague, friend, etc.). While you can share the directory on Github, it would have many Python packages, making the transfer process very challenging. Instead of that, you can create an environment configuration .yml file and share it with that person. Now, they can create an environment like your one by using the .yml file.

For exporting the environment to the .yml file, you'll first have to activate the same and run the following command:

```
conda env export ><file_name>.yml
```

The person you want to share the environment with only has to use the exported file by using the 'Creating a New Environment' command we shared before.

Removing a Package from an Environment

If you want to uninstall a package from a specific Conda environment, use the following command:

```
conda remove -n <env_name><package_name>
```

On the other hand, if you want to uninstall a package from an activated environment, you'd have to use the following command:

```
conda remove <package_name>
```

Deleting an Environment

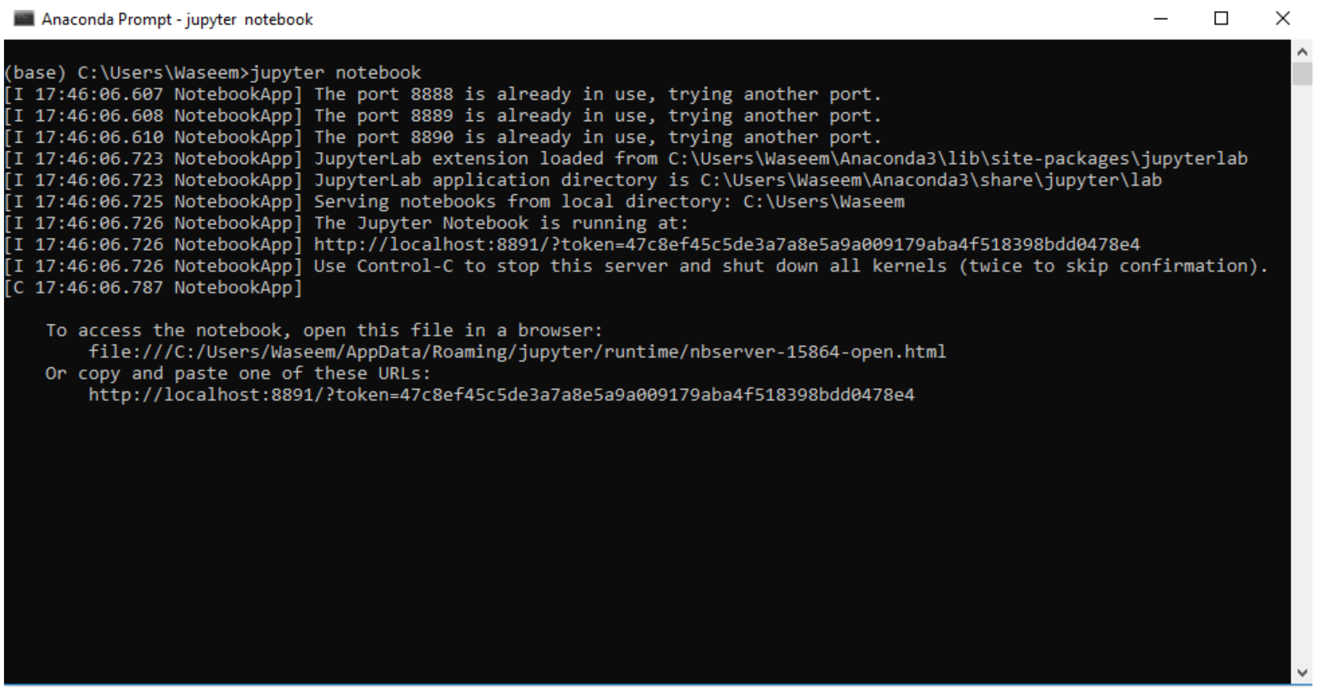
Sometimes, you don't need to add a new environment but remove one. In such cases, you must know how to delete a Conda environment, which you can do so by using the following command:

conda env remove –name <env_name>

The above command would delete the Conda environment right away.

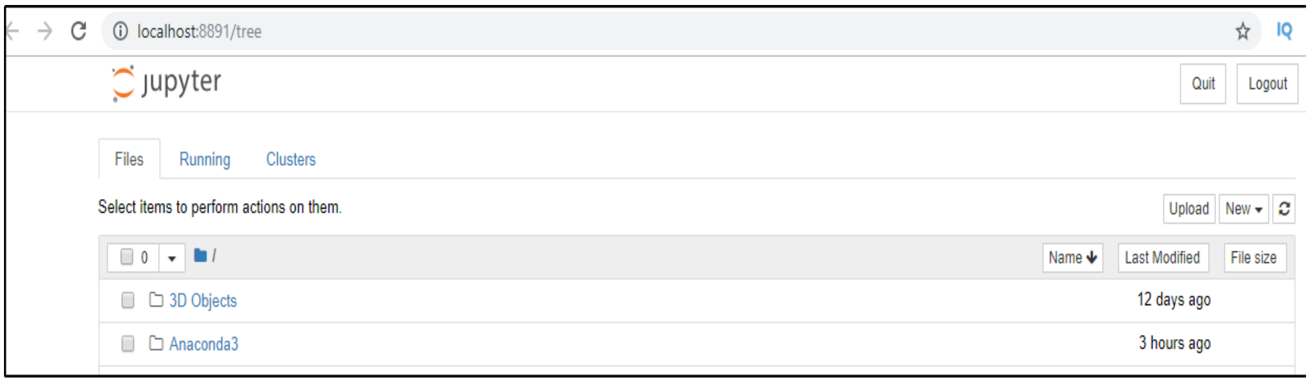
JUPYTER NOTEBOOK

open anaconda prompt and type jupyter notebook.



```
(base) C:\Users\Waseem>jupyter notebook
[I 17:46:06.607 NotebookApp] The port 8888 is already in use, trying another port.
[I 17:46:06.608 NotebookApp] The port 8889 is already in use, trying another port.
[I 17:46:06.610 NotebookApp] The port 8890 is already in use, trying another port.
[I 17:46:06.723 NotebookApp] JupyterLab extension loaded from C:\Users\Waseem\Anaconda3\lib\site-packages\jupyterlab
[I 17:46:06.723 NotebookApp] JupyterLab application directory is C:\Users\Waseem\Anaconda3\share\jupyter\lab
[I 17:46:06.725 NotebookApp] Serving notebooks from local directory: C:\Users\Waseem
[I 17:46:06.726 NotebookApp] The Jupyter Notebook is running at:
[I 17:46:06.726 NotebookApp] http://localhost:8891/?token=47c8ef45c5de3a7a8e5a9a009179aba4f518398bdd0478e4
[I 17:46:06.726 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 17:46:06.787 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/Waseem/AppData/Roaming/jupyter/runtime/nbserver-15864-open.html
    Or copy and paste one of these URLs:
        http://localhost:8891/?token=47c8ef45c5de3a7a8e5a9a009179aba4f518398bdd0478e4
```

You will see a window like shown in the image below.

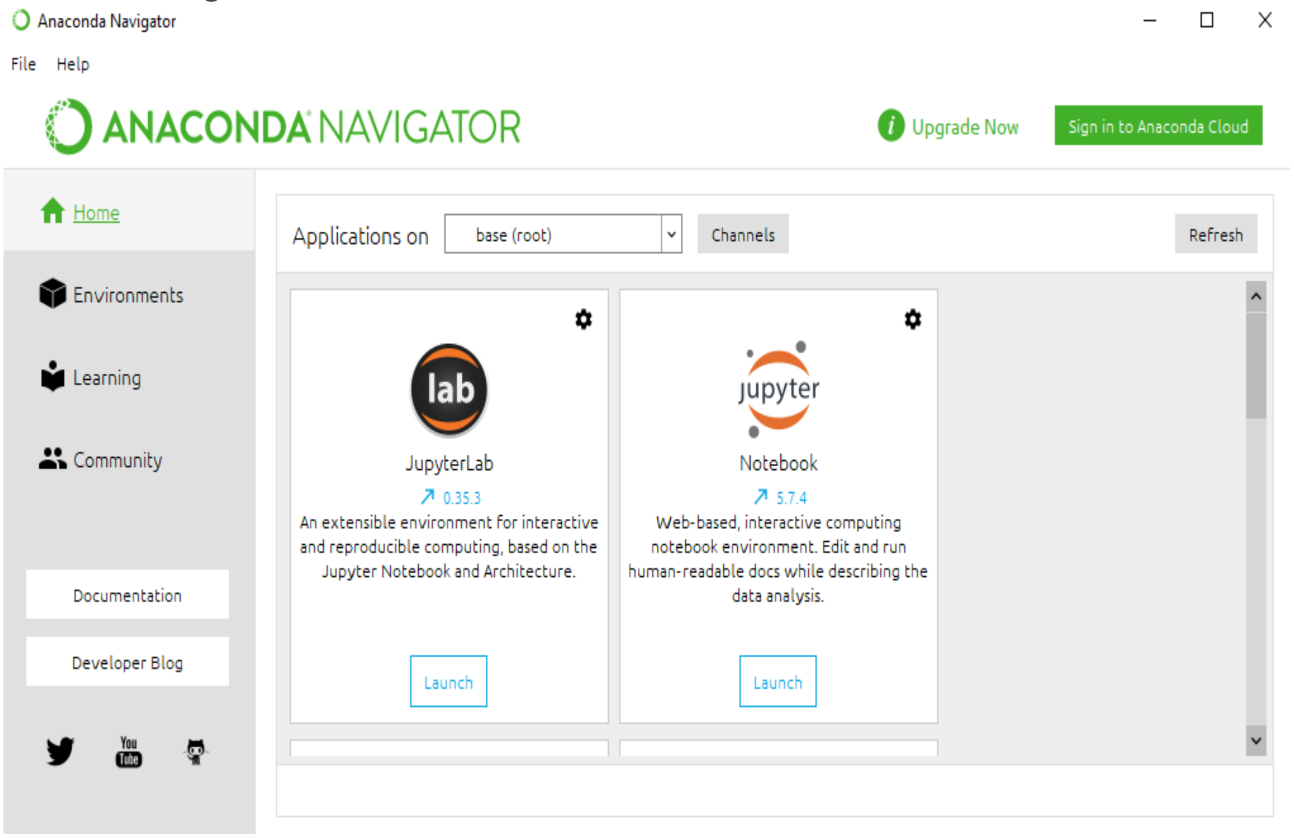


Select items to perform actions on them.

3D Objects	12 days ago	
Anaconda3	3 hours ago	

Now that we know how to use anaconda for python letslook at how we can install various libraries in anaconda for any project.

Anaconda Navigator



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Anaconda Navigator is a desktop GUI that comes with anaconda distribution. It allows us to launch applications and manage conda packages, environment and without using command-line commands.

Python: The Basics

```
print(5*3)
l = 5; b = 6
print("length={}, width={}, Area={}".format(l, b, l*b) )
```

```
15
length=5, width=6, Area=30
```

```
import math
radius = 5.0
Area = math.pi*radius ** 2
print("Area of a circle:{}".format(Area) )
```

```
Perimeter = 2 * math.pi *radius
print("Perimeter of a circle:{}".format(Perimeter) )
```

```
Area of a circle:78.53981633974483
Perimeter of a circle:31.41592653589793
```

```
print(math.sin(math.pi / 2))
print(math.log(1024, 2))
```

```
1.0
10.0
```

```
help(math)
```

```
Help on built-in module math:

NAME
   math

DESCRIPTION
   This module provides access to the mathematical functions
   defined by the C standard.

FUNCTIONS
acos(x, /)
       Return the arc cosine (measured in radians) of x.

acosh(x, /)
       Return the inverse hyperbolic cosine of x.

asin(x, /)
       Return the arc sine (measured in radians) of x.

asinh(x, /)
       Return the inverse hyperbolic sine of x.

atan(x, /)
```

```
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
  Number of ways to choose k items from n items without repetition and without order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.

        Also called the binomial coefficient because it is equivalent
        to the coefficient of k-th term in polynomial expansion of the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign of y.

        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.
```

```
dist(p, q, /)
    Return the Euclidean distance between two points p and q.

    The points should be specified as sequences (or iterables) of
    coordinates.  Both inputs must have the same dimension.

    Roughly equivalent to:
      sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

erf(x, /)
    Error function at x.

erfc(x, /)
    Complementary error function at x.

exp(x, /)
    Return e raised to the power of x.

expm1(x, /)
    Return exp(x)-1.

    This function avoids the loss of precision involved in the direct evaluation of exp(x)-1 for small x.

fabs(x, /)
    Return the absolute value of the float x.

factorial(x, /)
    Find x!.

    Raise a ValueError if x is negative or non-integral.

floor(x, /)
    Return the floor of x as an Integral.

    This is the largest integer <= x.

fmod(x, y, /)
    Return fmod(x, y), according to platform C.

    x % y may differ.
```

```
frexp(x, /)
 Return the mantissa and exponent of x, as pair (m, e).

        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

fsum(seq, /)
        Return an accurate floating point sum of values in the iterable seq.

        Assumes IEEE-754 floating point arithmetic.

    gamma(x, /)
        Gamma function at x.

gcd(x, y, /)
        greatest common divisor of x and y

hypot(...)
hypot(*coordinates) -> value

        Multidimensional Euclidean distance from the origin to a point.

        Roughly equivalent to:
            sqrt(sum(x**2 for x in coordinates))

        For a two dimensional point (x, y), gives the hypotenuse
        using the Pythagorean theorem:  sqrt(x*x + y*y).

        For example, the hypotenuse of a 3/4/5 right triangle is:

>>>hypot(3.0, 4.0)
            5.0

isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
        Determine whether two floating point numbers are close in value.

rel_tol
            maximum difference for being considered "close", relative to the
            magnitude of the input values
abs_tol
     maximum difference for being considered "close", regardless of the
            magnitude of the input values
        Return True if a is close in value to b, and False otherwise.

        For the values to be considered close, the difference between them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
        is, NaN is not close to anything, even itself.  inf and -inf are
        only close to themselves.

isfinite(x, /)
        Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(x, /)
        Return True if x is a positive or negative infinity, and False otherwise.

isnan(x, /)
        Return True if x is a NaN (not a number), and False otherwise.

isqrt(n, /)
        Return the integer part of the square root of the input.

ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x, [base=math.e])
       Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).
```

```
        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

modf(x, /)
        Return the fractional and integer parts of x.

    Both results carry the sign of x and are floats.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repetition and with order.

        Evaluates to n! / (n - k)! when k <= n and evaluates
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    pow(x, y, /)
        Return x**y (x to the power of y).

    prod(iterable, /, *, start=1)
        Calculate the product of all the elements in the input iterable.

        The default start value for the product is 1.

        When the iterable is empty, return the start value.  This function is
        intended specifically for use with numeric values and may reject
        non-numeric types.

    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

```
a = [1, 0, 0]
print(a)
print(type(a))
```

```
[1, 0, 0]
<class 'list'>


help(random.sample)


---------------------------------------------------------------------------
NameError  Traceback (most recent call last)
<ipython-input-7-81a606966755> in <module>
----> 1 help(random.sample)

NameError: name 'random' is not defined

import random
marks = random.sample(range(100), 10)
print(marks)

import statistics
print("Mean:{}".format(statistics.mean(marks)) )
print("Mode:{}".format(statistics.mode(marks)) )
print("Deviation:{}".format(statistics.stdev(marks)) )


dir(statistics)


def factorial(n):
   if(n==0): return 1

   if(n==1): return 1

   res = 1
   while(n>=1):
  res = res * n
n = n-1
   return res

def recursive_factorial(n):
   if(n==0): return 1

   if(n==1): return 1
```

```
    return n*recursive_factorial(n-1)

print(factorial(4))
print(recursive_factorial(4))
```

Numpy: The Basics

```
a = [ [1, 0, 0], [1,0,0] ]
print(a)
print(type(a))


a = [1, 0, 'Testing']
print(a)
print(type(a))


import numpy as np
a = np.array( [1,0,0] )
print(a)
print(type(a), a.dtype)


a = np.array( [ [1,0,0], [0,0,1] ] )
print(a)


a = np.array( [ (1,0,0), (0,0,1) ] )
print(a)


print(dir(np))
#help(np)
#help(np.unravel_index)


#help(np.zeros)
a = np.zeros( (3,3) )
print(a)


print(a.ndim)
print(a.shape)
print(a.size)
print(a.dtype)
print(a[2][2])


a = np.arange( 10, 30, 5 )
print(a)
a = np.linspace( 0, 2, 9 )
print(a)
```

Indexing, Slicing and Iterating

```
a = np.arange(10)**3
a


print(a[2:5])


print(a[5:])
print(a[:5])


print(a[5:-2])
print(a[-5:])


print(a[2:10:2])
a[2:10:2] = 0
print(a)


ar = np.random.normal(0, 1, size=(3,3))
print(ar)
print(ar[:1,])


print(ar[:1,1:3])


print(ar[...])
```

```
for i in ar:
    print(i)


for i in ar.flat:
    print(i)


aravel = ar.ravel()
print(aravel)
print(ar.flatten())
```

Shape Manipulation

```
print(ar.shape)
print(aravel.shape)


are= aravel.reshape(3,3)
print(are.shape)
print(are)


print(aravel)
aravel.resize(3,3)
print(aravel)
```

Stacking

```
a = np.floor(10*np.random.random((2,2)))
b = np.floor(10*np.random.random((2,2)))
print(a)
print(b)


print(np.vstack((a,b)))
print(np.hstack((a,b)))
```

SCipy

```
from scipy import special
from scipy import constants
```

```
print(special.entr(0.5) )
print(-0.5*math.log(0.5) )

print(constants.speed_of_light)
print(constants.g)
```

Viva Questions:

1. What are the applications of Anaconda Distribution?

The following are the applications that are provided by Anaconda Distribution:

1. **Jupyter Notebook** Jupyter Notebook is a web-based interactive environment that works as an IDE for many programming languages including Python and is the best platform for data science beginners.
2. **JupyterLab**Jupyter Lab is another development environment that is based on the Jupyter Notebook architecture.
3. **Visual Studio Code** Visual Studio Code or simply VS Code is a code editor by Microsoft that supports almost every programming language and provides various extensions to support them.
4. **Spyder** Spyder is a Python IDE that comes with advanced features such as interactive testing, debugging, task running, advanced editing, and introspective features.

2. Give us some of the salient features of Python Anaconda?

Anaconda is a package manager for Python and R and is one of the most popular platforms for data science aspirants. The following are some of the reasons that get Anaconda way ahead of its competitors.

1. Its robust distribution system helps in managing languages like Python which has over 300 libraries.
2. It is a free and open-source platform. Its open-source community has many eligible developers that keep helping the newbies constantly.
3. It has some AI and ML-based tools that can extract the data from different sources easily.
4. Anaconda has over 1500 Python and R data science packages and is considered the industry standard for testing and training models.

3. How good is Python for data analysis?

The following reasons make Python an essential language that every Data Scientist should know:

1. Python is the most suitable language for all fields of Data Science. Data analysis becomes efficient when Python is used in combination with R.
2. It has a rich library of data-oriented packages. You can visualize your data using different plots and charts.

3. Complex data sets can be handled using the powerful tools provided along with data frames.
4. The power-packed packages of Python such as Numpy, Pandas, Sci-kit provide features to produce accurate results.
5. It provides scalable and flexible solutions for the applications.
6. Matplotlib provides various plotting tools for accurate graphics and visualizations of your data.

4. What is Jupiter notebook?
5. Give examples of interpreted languages.

EX NO 2

Basic operations on Matrix & Vectors

Open in Colab

Python: The Basics

https://www.w3schools.com/python/

In [ ]:

```
print(5*3)
l = 5; b = 6
print("length={},width={},Area={}".format(l,b,l*b) )
```

```
15
length=5, width=6, Area=30
```

In [ ]:

```
import math
radius = 5.0

Area = math.pi *radius ** 2

print("Area of a circle:{}".format(Area) )

Perimeter = 2 * math.pi *radius
```

```
Area of a circle:78.53981633974483
Perimeter of a circle:31.41592653589793
```

In [ ]:

```
print(math.sin(math.pi / 2))
print(math.log(1024, 2))

print(0.3 == 3 * 0.1)
print(0.3 == 0.3)
```

```
1.0
10.0
False
True
```

In [ ]:

```
dir(math)
```

Out[ ]:

```
['_doc_',
'copysign','cos',
'cosh',    'degrees',
'dist',
'e',
'erf',
'erfc',
'exp',
'expm1',
'fabs',    'factorial',
'floor'
'tau',
'trunc',
'ulp']
```

In [ ]:

```
help(math)
```

```
Help on built-in module math:

NAME
    math


FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.The

        result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.The

        result is between -pi/2 and pi/2.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.The

        result is between -pi/2 and pi/2.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x. Unlike

        atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.This

        is the smallest integer >= x.

        intended specifically for use with numeric values and may reject
        non-numeric types.

    radians(x, /)
```

```
        Convert angle x from degrees to radians.
    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the__trunc_  magic method.

    ulp(x, /)
            Return the value of the least significant bit of the float x.

DATA
    e = 2.718281828459045
    inf  =  inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

In [ ]:

```python
import random
help(random.sample)
```

```
Help on method sample in module random:

sample(population, k, *, counts=None) method of random.Random instance
    Chooses k unique random elements from a population sequence or set.

    Returns a new list containing elements from the population while
    leaving the original population unchanged. The resulting list is in
    selection order so that all sub-slices will also be valid random samples.
    This allows raffle winners (the sample) to be partitioned into grand prize
    and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
```

selection in the sample.

Repeated elements can be specified one at a time or with the optional counts parameter. For example:

```
sample(['red', 'blue'], counts=[4, 2], k=5) is
```

equivalent to:

```
sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)
```

To choose a sample from a range of integers, use range() for the population argument. This is especially fast and space efficient for sampling from a large population:

```
sample(range(10000000), 60)
```

In [ ]:

```
import random

marks = random.sample(range(100), 10)
print(marks)
print(type(marks))

import statistics
print("Mean:{}".format(statistics.mean(marks)))
print("Mode:{}".format(statistics.mode(marks)))
```

```
[67, 2, 1, 47, 68, 42, 87, 72, 79, 6]
<class 'list'>
Mean:47.1
Mode:67
Deviation:33.228000240760
```

8In [ ]:

```
dir(statistics)
```

```
Out[ ]:   ['Counter',
            'Decimal',
            'Fraction',
            'NormalDist',
            'StatisticsError',     '
            all_',
            '__builtins__',  '
            cached_',
            '_doc_',
            '_file_',
```

```
def factorial(n):

    if(n==0):return 1


    if(n==1):return 1


    res = 1

    while(n>=1):

        res = res * nn =
        n-1

    return res

def  recursive_factorial(n):
    if(n==0):return 1
```

```
24
24
```

Numpy: The Basics

https://numpy.org/ https://www.w3schools.com/python/numpy/default.asp

In [ ]:

```
a = [ [1, 0, 0], [1,0,0] ]
print(a) print(type(a))
```

```
[[1, 0, 0], [1, 0, 0]]
<class 'list'> In
```

[ ]:

```
a  =  [1,  0,  'Testing']
print(a) print(type(a))
```

```
[1, 0, 'Testing']
<class 'list'> In
```

[ ]:

```
import numpy as np

a  =  np.array(  [1,0,0]  )
print(a)
print(type(a) , a.dtype)
```

```
[1 0 0]
<class 'numpy.ndarray'>  int32 In
```

[ ]:

```
a = np.array( [ [1,0,0], [0,0,1] ] )
print(a)
```

```
[[1 0 0]
 [0 0 1]]
```

In [ ]:

```
a = np.array( [ (1,0,0), (0,0,1) ] )
print(a)
```

```
[[1 0 0]
 [0 0 1]]
```

In [ ]:

```
print(dir(np)) #help(np)
#help(np.unravel_index)
```

```
['take_along_axis', 'tan', 'tanh', 'tensordot', 'test', 'testing', 'tile', 'timed elta64', 'trace',
'tracemalloc_domain', 'transpose', 'trapz', 'tri', 'tril', 'tril_ indices', 'tril_indices_from',
```

'trim_zeros', 'triu', 'triu_indices', 'triu_indices_ from', 'true_divide', 'trunc', 'typeDict', 'typecodes', 'typename', 'ubyte', 'ufunc ', 'uint', 'uint0', 'uint16', 'uint32', 'uint64', 'uint8', 'uintc', 'uintp', 'ulong long', 'unicode_', 'union1d', 'unique', 'unpackbits', 'unravel_index', 'unsignedint eger', 'unwrap', 'use_hugepage', 'ushort', 'vander', 'var', 'vdot', 'vectorize', 'v ersion', 'void', 'void0', 'vsplit', 'vstack', 'warnings', 'where', 'who', 'zeros','zeros_like']

In [ ]:

```
#help(np.zeros)

a = np.zeros( (3,3) )
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [ ]:

```
print(a.ndim)
print(a.shape)
print(a.size)
print(a.dtype)
print(a[2][2])
```

```
2
(3, 3)
9
float640.0
```

In [ ]:

```
a = np.arange( 10, 30, 5 )
print(a)
a = np.linspace( 0, 2, 9 )
print(a)
b= np.repeat(10, 5)
print(b)

d = np.linspace( 0, np.pi, 5)
print(d)
print(np.sin(d))
```

```
[10 15 20 25]
[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
[10 10 10 10 10]
[0.         0.78539816 1.57079633 2.35619449 3.14159265]
```

```
[0.00000000e+00           7.07106781e-01           1.00000000e+00
 7.07106781e-011.22464680e-16]
[ 1.  2.19328005  4.81047738 10.55072407 23.14069263]
```

In [ ]:

```
a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])

print(np.intersect1d(a,b))
print(np.setdiff1d(a,b))

print(a % 2 == 0)


print(np.where(a % 2 == 0))
```

```
[2 4]
[1 3 5 6]
[False  True  False  True  False  True  False  True  False  True]
(array([1, 3, 5, 7, 9], dtype=int64),)
[2 2 4 4 6]
```

Indexing, Slicing and Iterating

In [ ]:

```
a = np.arange(10)**3
print(a)
```

```
[  0    1    8   27   64 125 216 343 512 729]
```

In [ ]:

```
print(a[2:5])
```

```
[ 8 27 64]
```

In [ ]:

```
print(a[5:])
print(a[:5])
```

```
[125 216 343 512 729]
[  0   1   8  27  64]
```

In [ ]:

```
print(a[5:-2])
print(a[-5:])
```

```
[125 216 343]
```

```
[125 216 343 512 729]
```

In [ ]:

```
print(a[2:10:2])
a[2:10:2] = 0
print(a)
print(a[::-1])
```

```
[0 0 0 0]
[  0   1   0  27   0 125   0 343   0 729]
[729   0 343   0 125   0  27   0   1   0]
```

In [ ]:

```
ar  =  np.random.normal(0,  1,  size=(3,3))
print(ar)
print(ar[:1,])
```

```
[[ 2.12409173 -1.10461695  0.01692881]
 [ 0.25208916 -1.06828206 -0.06389821]
 [-1.41394657 -2.0987585   1.26396372]]
[[ 2.12409173 -1.10461695  0.01692881]]
```

In [ ]:

```
print(ar[:1,1:3])
```

```
[[-1.10461695  0.01692881]]
```

In [ ]:

```
print(ar[...])
print(ar[::-1])
print(ar[:,  ::-1])
```

```
[[ 2.12409173 -1.10461695  0.01692881]
 [ 0.25208916 -1.06828206 -0.06389821]
 [-1.41394657 -2.0987585   1.26396372]]
[[-1.41394657 -2.0987585   1.26396372]
 [ 0.25208916 -1.06828206 -0.06389821]
 [ 2.12409173 -1.10461695  0.01692881]]
[[ 0.01692881 -1.10461695  2.12409173]
 [-0.06389821 -1.06828206  0.25208916]
 [ 1.26396372 -2.0987585  -1.41394657]]
```

In [ ]:

```
for i in ar.flat:print(i)
```

```
2.1240917269169577
-1.1046169520593274
0.016928811303672946
0.2520891580311893
```

```
-1.068282058287259
-0.06389820817308421
-1.4139465719217985
-2.098758502243779
1.263963719832538
```

In [ ]:

```
aravel  =  ar.ravel()
print(aravel)
print(ar.flatten())
```

```
[ 2.12409173 -1.10461695  0.01692881  0.25208916 -1.06828206 -0.06389821


 -1.41394657 -2.0987585    1.26396372]
[ 2.12409173 -1.10461695  0.01692881  0.25208916 -1.06828206 -0.06389821
```

Shape Manipulation

In [ ]:

```
print(ar.shape)
print(aravel.shape)
```

```
(3, 3)
(9,)
```

In [ ]:

```
are=        aravel.reshape(3,3)
print(are.shape) print(are)
```

```
(3, 3)
[[ 2.12409173 -1.10461695  0.01692881]
 [ 0.25208916 -1.06828206 -0.06389821]
 [-1.41394657 -2.0987585   1.26396372]]
```

In [ ]:

```
print(aravel) aravel.resize(3,3)
print(aravel)
```

```
[ 2.12409173 -1.10461695  0.01692881  0.25208916 -1.06828206 -0.06389821
 -1.41394657 -2.0987585   1.26396372]
[[ 2.12409173 -1.10461695  0.01692881]
 [ 0.25208916 -1.06828206 -0.06389821]
 [-1.41394657 -2.0987585   1.26396372]]
```

Stacking

In [ ]:

```
a  =  np.floor(10*np.random.random((2,2)))
b  =  np.floor(10*np.random.random((2,2)))
print(a)
print(b)
```

```
[[1. 5.]
 [6. 3.]]
[[3. 3.]
 [5. 2.]]
```

In [ ]:

```
print(np.vstack((a,b)))
print(np.hstack((a,b)))
```

```
[[1. 5.]
 [6. 3.]
 [3. 3.]
 [5. 2.]]
[[1. 5. 3. 3.]
 [6. 3. 5. 2.]]
```

In [ ]:

```
#Exercises

#1.   Create a 2d array of (20, 20) shape with 1 on the border and 0 inside

#2. Create a checker or chess board as 8x8 matrix using the np.tile function(0- w#3.   Normalize a 5x5
random matrix. Z = (X - mu )/ Sigma

#4. How to get all the dates corresponding to the month of May 2022 [use np.arang#5. How to compute
```

In [ ]:

```
maze=np.ones((20,20))

maze[1:-1,1:-1]=0

print(maze)



chessboard  =  np.tile(  np.array([[0,1],[1,0]]),  (4,4))print(chessboard)



X  =  np.random.random((5,5))print(X)

Z = (X - np.mean (X)) / (np.std (X))print(Z)



days  =  np.arange('2022-05',  '2022-06',  dtype='datetime64[D]')print(days)



A=np.ones(3)*1
print(A)

B=np.ones(3)*2


vals=np.arange(10000)

np.random.shuffle(vals)n = 5

# Slow

print  (vals[np.argsort(vals)[-n:]])
```

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]

 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
[[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
[[0.78987272 0.6445424  0.79217012 0.13278395 0.58234288]
 [0.27929288 0.40593347 0.47542461 0.68341698 0.73909277]
 [0.23501403 0.73246197 0.49794173 0.44884328 0.94709192]
 [0.90104697 0.85533997 0.7137858  0.8692463  0.82307228]
 [0.89939226 0.98160857 0.15146111 0.83172449 0.25083619]]
[[ 0.63269861  0.06970251  0.64159852 -1.91280213 -0.17125265]
 [-1.34524016 -0.85464628 -0.58544409  0.22029899  0.4359818 ]
 [-1.5167723   0.41029471 -0.49821487 -0.6884177   1.24175115]
 [ 1.06337732  0.88631267  0.33794497  0.94018449  0.76131063]
 [ 1.05696712  1.37546545 -1.84044853  0.79482849 -1.45547872]]
['2022-05-01' '2022-05-02' '2022-05-03' '2022-05-04' '2022-05-05'
 '2022-05-06' '2022-05-07' '2022-05-08' '2022-05-09' '2022-05-10'
 '2022-05-11' '2022-05-12' '2022-05-13' '2022-05-14' '2022-05-15'
 '2022-05-16' '2022-05-17' '2022-05-18' '2022-05-19' '2022-05-20'
 '2022-05-21' '2022-05-22' '2022-05-23' '2022-05-24' '2022-05-25'
 '2022-05-26' '2022-05-27' '2022-05-28' '2022-05-29' '2022-05-30'
 '2022-05-31']
[1. 1. 1.]
[2. 2. 2.]
[3. 3. 3.]
[9995 9996 9997 9998 9999]
```

SCipy

https://scipy.org/
https://www.w3schools.com/python/scipy/index.php

In [ ]:

```
from scipy import special

from scipy import constants


print(special.entr(0.5)          )
print(-0.5*math.log(0.5)  )

print(constants.speed_of_light)
```

```
0.34657359027997264
0.34657359027997264
299792458.0
9.80665
```

In [ ]:

```
from scipy import constants
dir(constants)
```

Out[ ]:

```
['Avogadro',
 'Boltzmann',
 'Btu',
 'Btu_IT',
 'Btu_th',
 'ConstantWarning',
 'G',
 'Julian_year',
 'N_A',
 'Planck',
 'R',
 'Rydberg',
 'Stefan_Boltzmann','Wien',
 '_all_',
 '__builtins__',  '
 cached_',
 '_doc_',
 'atomic_mass',
 'atto',
 'au',
 'bar'
 'ton_TNT',   'torr',
'troy_ounce',
'troy_pound','u',
 'unit',
 'value',
 'week',
 'yard',
 'year',
 'yobi',
 'yotta',
 'zebi',
 'zepto',
 'zero_Celsius',
```

In [ ]:

```
from scipy.optimize import root

from math import cos


  def LinearEqn(x):

     return x + 27


def  QuadraticEqn(x):
    return x**2 -49


def  NonLinearEqn(x):
    return x + cos(x)


def  LinearEqnIn2Vars(x):
       f = [x[0] + x[1] - 9, x[0] - x[1] - 3]

       return   f


myroot = root(LinearEqn, 0)r
= myroot.x
print(r)
print(LinearEqn(r))

myroot = root(QuadraticEqn, 0)r
= myroot.x
print(r)
print(QuadraticEqn(r))

myroot   =   root(NonLinearEqn,
[-27.]
[0.]
[7.]
[0.]

[-0.73908513]

[0.]

<function LinearEqnIn2Vars at 0x000002117B480A60>[6. 3.]
```

In [ ]:

```
import scipy
help(root)
```

```
Help on function root in module scipy.optimize._root:

root(fun, x0, args=(), method='hybr', jac=None, tol=None, callback=None, options=None)
    Find a root of a vector function.

    Parameters
    ----------
    fun : callable
        A vector function to find a root of.x0 :
    ndarray
        Initial guess. args :
    tuple, optional
        Extra arguments passed to the objective function and its Jacobian.
    method : str, optional
        Type of solver. Should be one of

            - 'hybr'             :ref:`(see here) <optimize.root-hybr>`
            - 'lm'               :ref:`(see here) <optimize.root-lm>`
            - 'broyden1'         :ref:`(see here) <optimize.root-broyden1>`
            - 'broyden2'         :ref:`(see here) <optimize.root-broyden2>`
            - 'anderson'         :ref:`(see here) <optimize.root-anderson>`
            - 'linearmixing'     :ref:`(see here) <optimize.root-linearmixing>`
            - 'diagbroyden'      :ref:`(see here) <optimize.root-diagbroyden>`
            - 'excitingmixing'   :ref:`(see here) <optimize.root-excitingmixing>`
            - 'krylov'           :ref:`(see here) <optimize.root-krylov>`
            - 'df-sane'          :ref:`(see here) <optimize.root-dfsane>`

    'method' parameter. The default method is *hybr*.

    Method *hybr* uses a modification of the Powell hybrid method as
    implemented in MINPACK [1]_.
```

Method *lm* solves the system of nonlinear equations in a least squares sense using a modification of the Levenberg-Marquardt algorithm as implemented in MINPACK [1]_.

Method *df-sane* is a derivative-free spectral method. [3]_

Methods *broyden1*, *broyden2*, *anderson*, *linearmixing*, *diagbroyden*, *excitingmixing*, *krylov* are inexact Newton methods, with backtracking or full line searches [2]_. Each method corresponds to a particular Jacobian approximations. See `nonlin` for details.

- Method *broyden1* uses Broyden's first Jacobian approximation, it is known as Broyden's good method.
- Method *anderson* uses (extended) Anderson mixing.
- Method *Krylov* uses Krylov approximation for inverse Jacobian. It is suitable for large-scale problem.
- Method *diagbroyden* uses diagonal Broyden Jacobian approximation.
- Method *linearmixing* uses a scalar Jacobian approximation.
- Method *excitingmixing* uses a tuned diagonal Jacobian approximation.

.. warning::

    The algorithms implemented for methods *diagbroyden*, *linearmixing* and *excitingmixing* may be useful for specific problems, but whether they will work may depend strongly on the problem.

.. versionadded:: 0.11.0

References

.. [1] More, Jorge J., Burton S. Garbow, and Kenneth E. Hillstrom. 1980. User Guide for MINPACK-1.
.. [2] C. T. Kelley. 1995. Iterative Methods for Linear and Nonlinear Equations. Society for Industrial and Applied Mathematics. <https://archive.siam.org/books/kelley/fr16/>
.. [3] W. La Cruz, J.M. Martinez, M. Raydan. Math. Comp. 75, 1429 (2006).

```
>>> def fun(x):
...         return [x[0]  + 0.5 * (x[0] - x[1])**3 - 1.0,
...                     0.5 * (x[1] - x[0])**3 + x[1]]

>>> def jac(x):
...         return np.array([[1 + 1.5 * (x[0] - x[1])**2,
...                                  -1.5 * (x[0] - x[1])**2],
...                                 [-1.5 * (x[1] - x[0])**2,
...                                  1 + 1.5 * (x[1] - x[0])**2]])
```

A solution can be obtained as follows.

```
>>> from scipy import optimize
>>> sol = optimize.root(fun, [0, 0], jac=jac, method='hybr')
>>> sol.x
array([ 0.8411639,   0.1588361])
```

In [ ]:

```
#exercises - Find the roots for the following equations#univariate with
degree 2 & 3

#1.   5x**2 + 3*x - 9 =0

#2.   3x**3 + 2*x**2 + 6x - 11 =0#Non-
linear univariate
```

In [ ]:

```
defQuad(x):

    return 5*x**2 + 3*x - 9


myroot = root(Quad, 0)
r   = myroot.x
print(r) print(Quad(r))
```

```
[1.07477271]
[-1.19015908e-13]
```

In [ ]:

```
defCubic(x):

    return 3*x**3 + 2*x**2 + 6*x - 11


myroot = root(Cubic, 0)r
= myroot.x
print(r) print(Cubic(r))
```

```
[1.]
[0.]
```

In [ ]:

```
defexpEqn(x):

   returnmath.exp(-x)-math.sin(x)


myroot = root(expEqn, 0)
r   = myroot.x
```

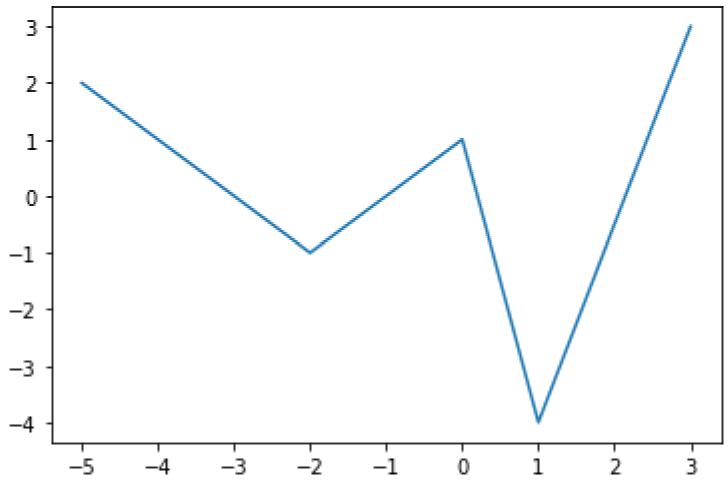
```
[0.58853274]
1.2212453270876722e-
15
```

Matplotlib

https://matplotlib.org/ https://www.w3schools.com/python/matplotlib_intro.asp

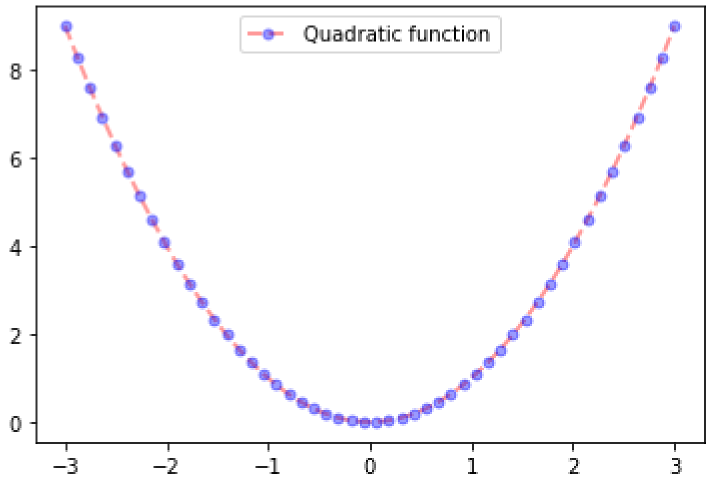
In [1]:

```
import numpy as np

import matplotlib.pyplot as plt

%matplotlib  inline


x=[-5,-2,0,1,3]
```



In [2]:

```
x=np.linspace(-3,3,50)y = x**2

plt.plot(x,y,alpha=0.4,label='Quadratic                    function',
            color='red',linestyle='dashed',linewidth=2,
            marker='o',markersize=5,markerfacecolor='blue', markeredgecolor='blue')
plt.legend()
plt.show()
```



In [3]:

```
help(plt.plot)
```

```
Help on function plot in module matplotlib.pyplot:

plot(*args, scalex=True, scaley=True, data=None, **kwargs) Plot
    y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], *, data=None, **kwargs) plot([x], y,
        [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string notation
    described in the *Notes* section below.

    >>> plot(x, y)          # plot x and y using default line style and color
    >>> 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
```

You can use `.Line2D` properties as keyword arguments for more control on the appearance. Line properties and *fmt* can be mixed. The following two calls yield identical results:

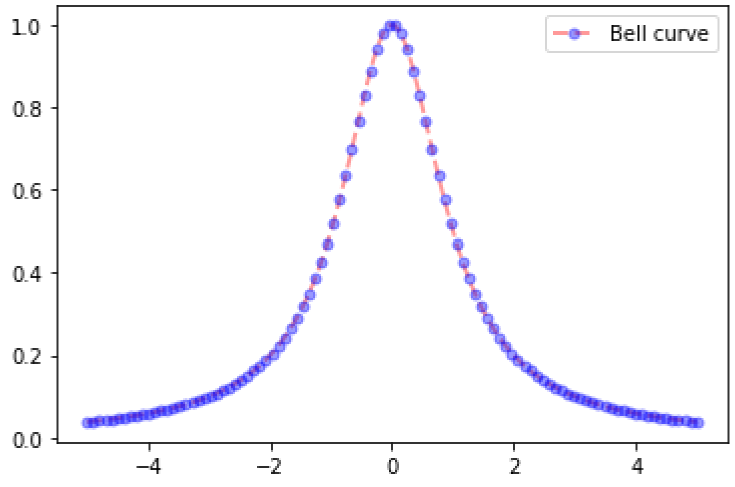
```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)
>>> plot(x, y, color='green', marker='o', linestyle='dashed',
...          linewidth=2, markersize=12)
```

In [4]:

```
#EXercise

#1. Draw plot for 1/(1+x**2) for the input range (-5,5)


#2. Plot the following butterfly curve

# x = sin(t) * ( e**(cos(t)- 2*cos(4*t) - (sin(t/12))**5 )
```

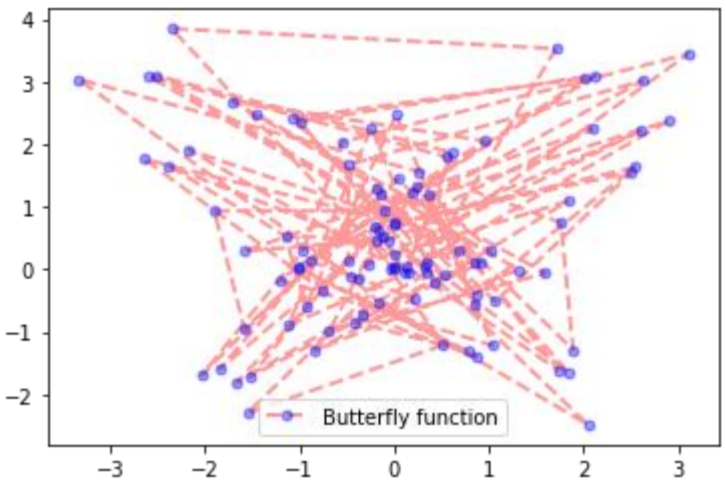
In [5]:

```
x = np.linspace(-5,5,100)y = 1/(1
+ x**2)

plt.plot(x,y,alpha=0.4,label='Bell                              curve',
         color='red',linestyle='dashed',linewidth=2,
         marker='o',markersize=5,markerfacecolor='blue', markeredgecolor='blue')
plt.legend()
plt.show()
```



In [6]:

```
t = np.arange(0, 101, 1)
x=np.sin(t)*(np.exp(np.cos(t))-2*np.cos(4*t)-(np.sin(t/12))**5)

y=np.cos(t)*(np.exp(np.cos(t))-2*np.cos(4*t)-(np.sin(t/12))**5)


plt.plot(x,y,alpha=0.4,label='Butterfly                                    function',
             color='red',linestyle='dashed',linewidth=2,
             marker='o',markersize=5,markerfacecolor='blue', markeredgecolor='blue')
plt.legend()
```



Exercises

1. Verify commutative property of random a b vectors in 3D w.r.t cross product
2. Veirfy|(a cross b)| **2 = |a|**2 * |b|**2 - (a.dot(b))**2 for any random a,b vectors
3. Compute 2I + 3A - AB for any random A, B matrices, I is identity(np.eye)
4. prove AB not equal to BA
5. Verify (A + B) ** 2 = A**2 + B**2 + 2A*B or not? A,B random sampled data from normaldistribution with mu = 0, sigma=1
6. prove k(A+B) = kA + kB and (k1 + k2)A = k1*A + k2*A for any radom A, B matrices, k, k1, k2 is any scalar
7. (A.T).T = A; (AB).T = A.T @ B.T; (A + B).T = A.T + B.T
8. Find matrix A such that (A.T)A = A(A.T) = I

Viva questions:

1. What are the applications of matrix?
2. Find possible order of matrix with n elements.
3. Define Row matrix, column matrix, square matrix, Rectangular matrix, Zero matrix, Diamond matrix, Scalar matrix, Unit matrix, Nilpotent matrix, Idempotent matrix, Orthogonal matrix, Upper triangular matrix, lower triangular matrix.
4. Explain the conditions required for Addition, Subtraction and multiplication of matrices.
5. Define null vector, Collinear vectors, coinitial vectors.

EX NO.3

Matrix analysis: Rank, Determinant, Trace, Orthogonal basis & Inverse of matrices

In [ ]:

```
import numpy as np

import scipy.linalg as spylinalg


a = np.random.normal(5, 1, size=(4,4))
#a=np.array([1,2])
```

```
[[ 1.41837222  -1.87463182  0.22843014   -0.18059195]
 [-0.22685521   2.20944718  -3.06101299   0.96310858]
 [ 0.55141178  -0.69427558  0.54570157   -0.48318058]
 [-2.09623058   0.45890915  2.94536955   -0.31466202]]
```

In [ ]:

```
print(a.dot(ainv))
```

```
[[  1.00000000e+00  -9.58118345e-16  -1.40644813e-16     3.08037797e-
 16]     [      4.40563844e-16           1.00000000e+00      -5.22381225e-16
 1.38154090e-16] [  6.00829958e-16  -1.28137707e-16    1.00000000e+00
 -2.96493798e-16] [  5.44382921e-16  -3.77294946e-16  -1.47143817e-16
 1.00000000e+00]]
```

In [ ]:

```
i4=np.eye(4)
i4
```

```
Out[ ]: array([[1.,  0., 0., 0.],
           [0.,      1., 0., 0.],
           [0.,      0., 1., 0.],
           [0.,      0., 0., 1.]])
```

In [ ]:

```
c=a.dot(ainv)
```

In [ ]:

```
np.array_equal(c,i4)
```

```
Out[  ]:  False
```

In [ ]:

```
(c>=i4).all()
```

Out[ ]: False

In [ ]:

```
(c>=i4).any()
```

Out[ ]: True In

[ ]:

```
np.isclose(c,i4)
```

```
Out[ ]: array([[ True,   True,   True,   True],
            [      True,   True,   True,   True],
            [      True,   True,   True,   True],
            [      True,   True,   True,   True]])
```

In [ ]:

```
np.allclose(c,i4)
```

Out[ ]: True In

[ ]:

```
print(spylinalg.det(a))
```

```
46.877814293583974
```

In [ ]:

```
print(a,spylinalg.norm(a))
help(spylinalg.norm)
```

In [ ]:

```
import math
a = np.arange(9) - 4.0
print(a,spylinalg.norm(a))
 math.sqrt(2*(4*4+3*3+2*2+1*1))
```

```
[-4. -3. -2. -1.  0.  1.  2.  3.  4.] 7.745966692414834
```

Out[ ]: 7.745966692414834

In [ ]:

```
print(spylinalg.norm(np.eye(4)))
```

```
2.0
```

In [ ]:

```
arect   =   np.random.normal(5,   1,   size=(4,3))
print(spylinalg.inv(arect))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-f96c37b73d0c> in <cell line: 2>()

      1 arect = np.random.normal(5, 1, size=(4,3))

----> 2 print(spylinalg.inv(arect))



/usr/local/lib/python3.10/dist-packages/scipy/linalg/_basic.py in inv(a, overwri te_a, check_finite)

    946     a1 = _asarray_validated(a, check_finite=check_finite)

    947     if len(a1.shape) != 2 or a1.shape[0] != a1.shape[1]:
```

In [ ]:

```
print("order=",arect.shape)    arectinv
= spylinalg.pinv(arect)
print("order    of    inverse=",arectinv.shape)
print(arectinv)    print(arect.dot(arectinv))
```

```
order= (4, 3)
order of inverse= (3, 4)
[[-0.39799764  0.30444227  -0.26011595   0.36714797]
 [ 0.25280472  -0.20359456  0.02565415   0.019315   ]
 [ 0.14802093  -0.01802459  0.24717887  -0.293071   ]]
[[ 0.77519776 -0.22434781  0.31187804   0.16329832]
 [-0.22434781   0.7761057   0.31124759   0.16296822]
[ 0.31187804  0.31124759   0.56731787 -0.22655095]
[ 0.16329832  0.16296822 -0.22655095   0.88137866]]
```

In [ ]:

```
a = np.random.normal(5, 1, size=(4,4)) print(a)
print(np.trace(a))
```

```
[[5.93865977  5.51837072  4.62052817  5.94203064]
 [5.8385074   3.89774421  4.83745607  6.19888733]
 [3.69924739  4.41159784  4.71505997  3.4419113  ]
 [5.16310566  5.23583371  6.62973053  4.75143893]]
19.302902877718495
```

In [ ]:

```
b=np.array([[1,3,4],[3,2,1],[2,3,4]])
print(np.trace(b))
```

```
7
```

In [ ]:

```
np.random.randint(1,10)
```

Out[ ]: 9

In [ ]:

```
[np.random.randint(-10,10) for i in range(20)]
```

Out[ ]: [-2, -8, 5, 0, 8, -4, -6, -8, 4, -5, 4, 7, -5, 6, -8, -6, 3, 8, 1, -10] In [ ]:

```
x=np.array([np.random.randint(-10,10) for i in range(20)]) print(x)
print(np.mean(x))
print("exp",np.exp(x))
print("max",np.max(x))
print("argmax", np.argmax(x))
```

```
[
2.85
exp   [4.03428793e+02   2.00855369e+01   8.10308393e+03   4.97870684e-02
 8.10308393e+03 8.10308393e+03 1.23409804e-04 8.10308393e+03
 2.00855369e+01  1.09663316e+03  3.35462628e-04  1.35335283e-01
 2.47875218e-03  4.03428793e+02  2.71828183e+00  2.98095799e+03
 2.98095799e+03  2.71828183e+00  2.98095799e+03  1.35335283e-01]
max 9
argmax 2
```

The difference between them is that the reshape() does not changes the original array but only returns the changed array, whereas the resize() method returns nothing and directly changes the original array.

In [ ]:

```
x.reshape(5,4)
```

```
Out[ ]: array([[  6,   3,  9,   -3],
    [       9,  9, -9,  9],
    [       3,  7, -8, -2],
 [-6,          6,  1,   8],
 [ 8,          1,  8,  -2]])
```

In [ ]:

```
x.resize(5,4)
```

In [ ]:

```
x
```

```
Out[ ]: array([[  6,   3,  9,  -3],
       [     9,   9, -9,   9],
       [     3,   7, -8,  -2],
            [-6,  6,  1,  8],
            [ 8,  1,  8, -2]])
```

In [ ]:

```
import random

import numpy as np

marks  =  random.sample(range(100),  9)
a  =  random.sample(range(100),  9)
b  =  random.sample(range(100),  9)
print(marks,a)   print(type(marks))
A=np.array(a)
A.resize(3,3)  print(A)
B=np.array(b)
B.resize(3,3)  print(B)
print(A.dot(B))
AB=A.dot(B)
print("AB.T",AB.T)
BT=B.T
AT=A.T
```

```
 [58, 90, 94, 32, 28, 18, 78, 47, 48] [36, 32, 67, 14, 2, 5, 83, 29, 73]
<class 'list'>

 [[36 32   67]

 [14 2     5]

 [83 29    73]]

 [[ 3 11    37]

 [29 97    33]

 [34 81    87]]

 [[ 3314   8927   8217]

 [ 270     753     1019]

 [ 3572    9639   10379]]

 AB.T [[    3314   270 3572]
```

```
[   8927 753   9639]

[   8217  1019  10379]]

[[  3314  270   3572]

[   8927  753   9639]
```

Exercises

1. A Aiv = Ainv A =1
2. (A.inv).inv = A
3. (AB).inv = B.invA.inv
4. (A.T).inv = (A.inv).T
5. (k * A.inv) = 1/k * A.inv ; k – scalar
6. pick some properties related to determinant, Trace & Rank and do programs

Viva questions:

1. Explain Closure Property, Commutative Associative Property, Commutative Property, Distributive Property with respect to matrix addition, subtraction and multiplication.
2. What is the determinant of the product of two matrices in terms of their individual determinants?
3. How the determinant is varries with the operations, such as scalar multiplication, row operations, and swapping rows?

EX NO.4

Eigen values and Eigenvectors of Matrix.

In [ ]:

```
import numpy as np

from scipy import linalg


m = 4

n = 4


A     =     np.random.standard_exponential(size=(m,n))

A=np.array([[5,1],[2,3]])
print(A)
lam, V = linalg.eig(A)
```

```
[[5 1]
 [2 3]]
[5.73205081+0.j  2.26794919+0.j][[
0.80689822 -0.34372377]
 [ 0.59069049   0.9390708 ]]
```

In [ ]:

```
ainv=linalg.inv(A)
lam1,V1=linalg.eig(ainv)
print(lam1)
print(V1)
if          (lam==lam1).all():
    print(" equal")
else:
```

```
[-4.23606798+0.j  0.23606798+0.j]
[[-0.85065081 -0.52573111]
 [ 0.52573111 -0.85065081]]
not Equal
```

EigenVector Decomposition

A (nonzero) vector v of dimension N is an eigenvector of a square N × N matrix A if it satisfies a linear equation of the form

Av = Lambda*v

for some scalar λ. Then λ is called the eigenvalue corresponding to v. Geometrically speaking, the eigenvectors of A are the vectors that A merely elongates or shrinks, and the amount that they elongate/shrink by is the eigenvalue. The above equation is called the eigenvalue equation or the eigenvalue problem.

In [ ]:

```
print(A.dot(V))
print(lam * V )
```

```
[[ 0.20081142 -2.22703273]
 [-0.12410828  -3.60341465]]
[[  0.20081142-0.j  -2.22703273+0.j]  [-
 0.12410828+0.j  -3.60341465+0.j]]
```

In [ ]:

```
print(linalg.norm(A.dot(V)  -  lam*V))
```

```
7.260351408704585e-16
```

In [ ]:

```
# A @ V = Lambda * V = V * Lambda
# A = V*Lambda @ V-inv


m=n=2

lambdaMatrix  =  np.zeros((m,  n)  ,  dtype=complex)
lambdaInvMatrix  =  np.zeros((m,  n),  dtype=complex)
for i in range(min(m, n)):
      lambdaMatrix[i,     i]     =     lam[i]
      lambdaInvMatrix[i, i] = 1/lam[i]

Vinv = linalg.inv(V)
print(  (V.dot(lambdaMatrix)).dot(Vinv)  )
```

```
[[1.+0.j 2.+0.j]
 [2.+0.j   3.+0.j]]  [[1
2]
 [2 3]]
```

In [ ]:

```
```

In [ ]:

```
# A = V*Lambda @ V-inv

# Ainv = V * (Lambdainv) @ (V-inv)


Vinv = linalg.inv(V)
#print(Vinv)

Ainv          =          V.dot(lambdaInvMatrix).dot(Vinv)
print(Ainv)
Ainvd  =  linalg.inv(A)
```

```
[[-3.+0.j   2.+0.j]
 [ 2.+0.j -1.+0.j]]
[[-3.   2.]
 [            2.            -1.]]
1.9641850382783467e-
15
```

In [ ]:

```
# det(A) = multipy lambda values


print("Det:\n{}".format(linalg.det(A))) print(lam)
print(lam[0]*lam[1])
print("lambda  Cumulative  product:\n{}".format(lam.cumprod())  )
```

```
Det:
-1.0
```

```
[-0.23606798+0.j  4.23606798+0.j]
(-1.0000000000000004+0j)
lambda Cumulative product:
[-0.23606798+0.j -1.          +0.j]
```

In [ ]:

```
# Trace(A) = sum lambda values

print("Trace:\n{}".format(np.trace(A)))
print("lambda Cumulative sum:\n{}".format(lam.cumsum()) )
```

```
Trace:
4
lambda Cumulative sum:
[-0.23606798+0.j  4.          +0.j]
```

In [ ]:

```
D = np.array( [(1,1), (0,1)] )
print(D)
lamn,  Vn  =  linalg.eig(D)
print(lamn)
print(Vn)
print(linalg.norm(D.dot(Vn)  -  lamn*Vn))
```

```
[[1 1]
 [0 1]]
[1.+0.j 1.+0.j]
[[  1.00000000e+00  -1.00000000e+00]
[  0.00000000e+00  2.22044605e-16]]
2.220446049250313e-16
```

In [ ]:

```
arr=np.array([[1,1],[2,2]])
print(linalg.norm(arr))  import
math print(math.sqrt(10))
```

```
3.1622776601683795
3.1622776601683795
```

EigenVector Decomposition

A (nonzero) vector v of dimension N is an eigenvector of a square N × N matrix A if it satisfies a linear equation of the form

Av = Lambda*v

for some scalar λ. Then λ is called the eigenvalue corresponding to v. Geometrically speaking, the eigenvectors of A are the vectors that A merely elongates or shrinks, and the amount that they elongate/shrink by is the eigenvalue. The above equation is called the eigenvalue equation or the eigenvalue problem.

Exercises

1. Compute eigenvectors and eigen values for any random (3,3) matrix

2. Compute inverse for any random matrix using Eigen factorization and verify with inv() &pinv() methods.
3. Compute and compare eigenvectors and eigen values for any random (3,3) matrix and its inverse matrix - verify the results
4. Compute and compare eignevectors and eigne values for any real symmetric matrx and verify the results - eigen values are real
5. Compute eignevectors and eigne values for any real symmetric matrx - check eigen vectors are orthogonal
6. Verify matrix is not invertible if any one of its eigen value is zero.\

#Positive Definite Matrix - 1) symmetric, 2) all eigenvalues are positive, 3) all the subdeterminants are also positive
#Read - https://towardsdatascience.com/decomposing-eigendecomposition-f68f48470830

Viva questions:

1. How do you find the eigenvalues of a square matrix?
2. What is the relationship between the determinant and the eigenvalues of a matrix?
3. Explain the geometric interpretation of eigenvectors and eigenvalues in terms of stretching and compression.
4. How do eigenvalues and eigenvectors play a role in solving linear transformations in computer graphics?

EX.NO: 5

Matrix decompositions: SVD, QR, LU, Pseudo Inverse

solve system of linear equations:

x+2y = 5 3x+4y = 7 Ax = bx

= Ainv.dot(b)

SVD: Singular value Decomposition(or factorization) of a Matrix

https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/

In [ ]:

```
import numpy as np
from scipy import linalg

m = 4
n = 4

A = np.random.randint(100, size=(m,n))
print(A)
```

```
[[57 22 45 11]
 [26 84 47  0]
 [27 44 25 32]
 [10 51 63  4]]
[140.96680271+0.j          -33.76180003+0.j
  31.39749866+6.12798716j                 31.39749866-
  6.12798716j]
[[ 0.46444385+0.j              0.17492047+0.j             -0.77072148+0.j
  -0.77072148-0.j              ]
 [ 0.59335982+0.j              0.21447308+0.j
                               0.15219943+0.13410252j                0.15219943-
   0.13410252j]
 [ 0.4622611 +0.j             -0.63414188+0.j              0.23852999-
   0.13024366j0.23852999+0.13024366j]
 [ 0.46747268+0.j              0.72198906+0.j              0.513651                 -
   0.16475112j0.513651   +0.16475112j]]
```

In [ ]:

```
#help(linalg.svd)

#genralization of EigenDecomposition - works for any matrix of m*n shape ;   A

A=np.array([[2,2],[1,1]])
U, s, V = linalg.svd(A)
```

```
[[-0.89442719 -0.4472136 ]
 [-0.4472136   0.89442719]]
[3.16227766 0.         ]
[[-0.70710678  -0.70710678]
 [-0.70710678  0.70710678]]
multi U (2, 2) [[ 1.00000000e+00 -2.43158597e-17]
```

```
 [-2.43158597e-17
1.00000000e+00]]          s=          (2,)
[3.16227766 0.                          ]
multi V (2, 2) [[ 1.00000000e+00 -2.23711432e-17]
 [-2.23711432e-17   1.00000000e+00]]
A= [3. 1.]
```

In [ ]:

```
sigma   =   np.zeros((m,   n))
sigmainv  =  np.zeros((m,  n))
for i in range(min(m, n)):
      sigma[i,     i]     =     s[i]
      sigmainv[i,i] = 1/s[i]

A1 = np.dot(U, np.dot(sigma, V))
print(A)
print(A1)
print(sigma)
```

```
[[28 82 71    93]
 [80 39   4    92]
 [71 59 75    38]
 [86 55 98    86]]
[[28.  82.  71. 93.]
 [80.  39.   4. 92.]
 [71.  59.  75. 38.]
 [86.  55.  98. 86.]]
[[268.68442519       0.           0.        0.        ]
     [   0.      70.8340234        0.        0.        ]
     [   0.          0.      54.09113766    0.        ]
 [   0.            0.           0.          21.82589327]]
[[0.00372184     0.         0.       0.           ]
     [0.     0.01411751     0.       0.           ]
     [0.         0.     0.01848732   0.           ]
     [0.     0.         0.             0.04581714]]
```

In [ ]:

```
# Verify U,V matrices are orthogonal

#U(U.transpose) = I; V(V.transpose) = I - means U, V orthogonal represents rot

print("U(U.transpose)=\n{}".format(U.dot(U.transpose())))
```

```
U(U.transpose)=
[[1.00000000e+00 4.46556167e-16 5.86748493e-16 1.97698197e-16]
 [4.46556167e-16   1.00000000e+00   4.33526340e-16   1.47787471e-
 16]        [5.86748493e-16        4.33526340e-16        1.00000000e+00
 1.07838093e-16]  [1.97698197e-16   1.47787471e-16   1.07838093e-
 16  1.00000000e+00]]
V(V.transpose)=
[[  1.00000000e+00  -4.40693757e-17  -1.24488813e-16  -1.40144299e-16]
 [-4.40693757e-17 1.00000000e+00 4.96232664e-16 -4.32704398e-18] [-
 1.24488813e-16  4.96232664e-16  1.00000000e+00  8.55723150e-17]
 [-1.40144299e-16  -4.32704398e-18  8.55723150e-17  1.00000000e+00]]
```

In [ ]:

```
#inv is easy for orthogonals U-inv = U.transpose

Uinv_ortho  =  U.transpose()
Uinv      =     linalg.pinv(U)
print(Uinv_ortho)
```

```
[[-0.5128713 -0.41361687 -0.44275104  -0.60815757]
 [ 0.19981509 -0.89614636  0.34962047   0.18644347]
 [ 0.82733753 -0.06015567 -0.45242466  -0.32742301]
 [-0.11202487 -0.14907744 -0.69068732   0.69869691]]
```

```
[[-0.5128713 -0.41361687 -0.44275104 -0.60815757]
 [ 0.19981509 -0.89614636  0.34962047  0.18644347]
 [ 0.82733753 -0.06015567 -0.45242466 -0.32742301]
```

In [ ]:

```
#A = UsV; A.inv = (V.T)(s.inv)(U.T)

Ainv_svd  =   ((V.transpose()).dot(sigmainv)).dot(U.transpose())

Ainv  =  linalg.pinv(A)
print(Ainv)
print(Ainv_svd)

print(A.dot(Ainv)) print(A.dot(Ainv_svd))
```

```
[[-0.01098503   0.0073777   0.01134864  -0.00102777]
 [ 0.01128064  0.00346817  0.02348059  -0.02628411]
 [-0.00091193 -0.01147722 -0.00596975   0.01590191]
 [ 0.00480984  0.00348299 -0.01956256  0.01134451]]
[[-0.01098503   0.0073777   0.01134864  -0.00102777]
 [ 0.01128064  0.00346817  0.02348059  -0.02628411]
 [-0.00091193 -0.01147722 -0.00596975   0.01590191]
 [ 0.00480984  0.00348299 -0.01956256  0.01134451]]
[[  1.00000000e+00  7.35522754e-16  5.68989300e-16  -8.70831185e-
  16]    [     2.08166817e-16     1.00000000e+00     1.05471187e-15     -
  4.57966998e-16]         [          3.26128013e-16           6.38378239e-16
  1.00000000e+00  1.31838984e-16] [  7.42461648e-16  1.94289029e-
  16  1.94289029e-16  1.00000000e+00]]
[[ 1.00000000e+00   3.87277016e-16   1.11369247e-15 -6.48786580e-16]
 [-7.97972799e-17  1.00000000e+00  1.15185639e-15 -4.57966998e-16]
 [-5.20417043e-18  1.83013327e-16  1.00000000e+00  1.31838984e-16]
 [ 2.02962647e-16 -1.56992475e-16  4.85722573e-17  1.00000000e+00]]
```

In [ ]:

```
#x+2y = 5 3x+4y = 7 Ax = b#x =
Ainv.dot(b)

A= np.array([[1,2], [3,4]])
b= np.array([5,7])
x=     linalg.inv(A).dot(b)
print(x) print(A.dot(x))
```

```
[-3.   4.]
[5. 7.]
```

In [ ]:

```
x=          linalg.pinv(A).dot(b)
print(x)
print(A.dot(x))
```

```
[-3.   4.]
[5. 7.]
```

In [ ]:

```
#x+2y = 5 3x+4y = 7   5x+11y=21   Ax = b

A= np.array([[1,2], [3,4], [5,11]])
b= np.array([5,7,21])
x=          linalg.pinv(A).dot(b)
print(x)
```

```
[-0.51724138  2.16091954]
[ 3.8045977    7.09195402  21.18390805]
```

QR Decompostion

In [ ]:

```
#help(linalg.qr)

#QR Decomposition - works for any matrix of m*n shape A = QR

#   https://atozmath.com/example/MatrixEv.aspx?he=e&q=qrdecompgs

A  =  np.random.randint(100,  size=(4,3)) Q,
R = linalg.qr(A)
```

```
[[-0.68549465  0.31183315   0.30691354  0.58194609]
 [-0.69588093 -0.02347815  -0.26733946 -0.6661292 ]
 [-0.16618052 -0.57167744  -0.65977848  0.45854259]
 [-0.13502167 -0.75854714   0.63168509 -0.0857282 ]]
[[-96.28083922  -78.94613364      -75.69522721]
     [   0.      -45.27149194      -65.65178988]
     [   0.            0.           -66.52123768]
     [   0.            0.             0.        ]]
```

LU Decompostion

In [ ]:

```
#help(linalg.lu)

#QR Decomposition - works for any matrix of m*n shape A = LU

P,  L,  U  =  linalg.lu(A)
print(L)
print(U)print(P)
```

```
[[      1.           0.           0.         0.        ]
 [  0.45614035      1.           0.         0.        ]
 [   0.1754386   0.63733397      1.         0.        ]
 [  0.47368421  0.45398482 -0.21799254    1.        ]]
 [[57.         22.         45.         11.        ]
 [ 0.         73.96491228 26.47368421  -5.01754386]
 [ 0.               0.     38.23268501   5.26802657]
 [ 0.               0.          0.     30.21575294]]
[[1. 0. 0.       0.]
 [0. 1. 0.       0.]
 [0. 0. 0.       1.]
 [0. 0. 1.      0.]]
```

In [ ]:

```
A=np.array([[3,2,4],[2,0,2],  [4,2,3]])
p,l,u=linalg.lu(A) print(p,"\n",l,"\n",u)
```

```
[[0. 0.      1.]
 [0. 1.      0.]
 [1. 0.     0.]]
  [[ 1.    0.    0.   ]
  [ 0.5    1.    0.   ]
 [ 0.75 -0.5    1.  ]]
  [[ 4.    2.   3. ]
   [ 0.   -1.   0.5]
   [ 0.    0.   2. ]]
```

In [ ]:

```
import numpy as np

from scipy import linalg


m = 4

n = 4


A  =  np.random.standard_exponential(size=(m,n))
A=np.array([[5,-10,-5],[2,14,2],[-4,-8,6]])

print(A)
lam,  V  =  linalg.eig(A)
print("lam=",lam)
```

```
[[  5 -10   -5]
 [  2  14    2]
 [ -4  -8    6]]
lam= [ 5.+0.j 10.+0.j 10.+0.j]
V=  [[  7.45355992e-01  7.44760246e-16  -5.22799043e-
 01] [-2.98142397e-01  -4.47213595e-01   5.75730417e-
 01]
 [  5.96284794e-01  8.94427191e-01  -6.28661791e-01]]
(3, 3)
AxV=      [[       3.72677996e+00        5.66213743e-15       -
 5.22799043e+00]   [-1.49071198e+00    -4.47213595e+00
 5.75730417e+00]
 [  2.98142397e+00    8.94427191e+00  -6.28661791e+00]]
lam.V=          [[          3.72677996e+00+0.j          7.44760246e-15+0.j          -
 5.22799043e+00+0.j]      [-1.49071198e+00+0.j       -4.47213595e+00+0.j
 5.75730417e+00+0.j]
 [  2.98142397e+00+0.j     8.94427191e+00+0.j  -6.28661791e+00+0.j]]
```

In [ ]:

```
print(A.dot(V))
print(lam * V )
```

```
[[ 0.20081142 -2.22703273]
 [-0.12410828  -3.60341465]]
[[  0.20081142-0.j  -2.22703273+0.j] [-
 0.12410828+0.j  -3.60341465+0.j]]
```

In [ ]:

```
import numpy as np
from scipy import linalg
A=np.array([[3,1,1],[-1,3,  1]])
U,  s,  V  =  linalg.svd(A)
print(U)
print(s)print(V)
print("multi        U",U.shape,        U.dot(U))
print("s=",s.shape,s)
```

```
[[-0.70710678 -0.70710678]

[-0.70710678 0.70710678]]

[3.46410162 3.16227766]

[[-4.08248290e-01 -8.16496581e-01 -4.08248290e-01]

[-8.94427191e-01 4.47213595e-01 5.26260748e-16] [-1.82574186e-01 -3.65148372e-01
9.12870929e-01]] multi U (2, 2) [[ 1.00000000e+00 -1.33393446e-16]

[-1.56386917e-16 1.00000000e+00]]

s= (2,) [3.46410162 3.16227766]

multi V (3, 3) [[ 0.97149901   0.11725616 -0.20601133]
[-0.03485163   0.93029674   0.36514837]
[ 0.23446756 -0.34756145   0.90786893]]
```

Exercises

1. Compute singular values for any random (3,3) matrix
2. Compute SVD factorization for any random (4,3) matrix
3. compute inverse for any random matrix using SVD factorization and verify with pinv() methods.
4. Compute and compare singular values for any real symmetric matrx and verify the results –
5. For any random matrix A, Compute eigen vectors for A @ A.T, A.T @ A and compare with U, V in SVD.
6. verify matrix is not invertible if any one of its singular value is zero.
7. Compute SVD for any Positive Definite Matrix - 1) symmetric, 2) all eigenvalues are positive, 3) all the subdeterminants are also positive

#Read - https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d#:~:text=In%20linear%20algebra%2C%20the%20Singular%20Value%20Decomposition%20%28SVD%29,also%20has%20some%20important%20applications%20in%20data%20science.

Viva Questions:

1. Explain the mathematical representation of a matrix A using SVD.
2. Explain linear transformations and nonlinear transformations.
3. How is the pseudoinverse calculated using the SVD?
4. Explain the differences between pseudo inverse and ordinary matrix inverse. Discuss the advantages and disadvantages of both.

EX NO: 6

Solve system of linear equations

solve system of linear equations:

Root finding refers to the general problem of searching for a solution of an equation

$F(x) = 0$ for some function $F(x)$. This is a very general problem and it comes up a lotin mathematics! For example, if we want to optimize a function $f(x)$ then we need to find critical points and therefore solve the equation $f'(x) = 0$

solve system of linear equations:

x+2y = 53x+4y = 7

Ax = b

x = Ainv.dot(b)

In [ ]:

```
#x+2y = 5 3x+4y = 7 Ax = b#x =
Ainv.dot(b)
import numpy as np
from scipy import linalg

A= np.array([[1,2], [3,4]])
b= np.array([5,7])
x=      linalg.inv(A).dot(b)
print(x)
print(A.dot(x))
```

```
[-3.   4.]
[5. 7.]
```

In [ ]:

```
linalg.solve(A, b)
```

Out[ ]: array([-3., 4.])

In [ ]:

```
m = 4; n=2

A  =  np.random.randint(100,  size=(m,n))
b  =  np.random.randint(100,  size=(m,1))
print("A=\n{}".format(A))
print("b=\n{}".format(b))
```

```
 A=
[[ 8  42]
 [41  45]
 [87  45]
 [93   4]]
```

```
b= [[34]
    [23]
    [37]
```

In [ ]:

```
#linalg.solve(A, b) - does not work

Ainv  =  linalg.pinv(A)
x     =     Ainv.dot(b)
print("x=",x)
```

```
x= [[0.51192628]
 [0.18402415]]
[[-22.17557527]
 [  6.27006443]
 [ 15.81867332]
 [-15.65475934]]
```

linalg.lstsq(a, b)

Return the least-squares solution to a linear matrix equation.

Computes the vector x that approximately solves the equation a @ x = b. The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rowsof a can be less than, equal to, or greater than its number of linearly independent columns). If a is square and of full rank, then x (but for round-off error) is the “exact” solution of the equation.

In [ ]:

```
#Minimizing  |Ax-b|
x,  res,  rnk,  s  =  linalg.lstsq(A,  b)
print(x)
print(A.dot(x) - b)
```

```
[[0.51192628]
 [0.18402415]]
[[-22.17557527]
 [  6.27006443]
 [ 15.81867332]
 [-15.65475934]]
```

In [ ]:

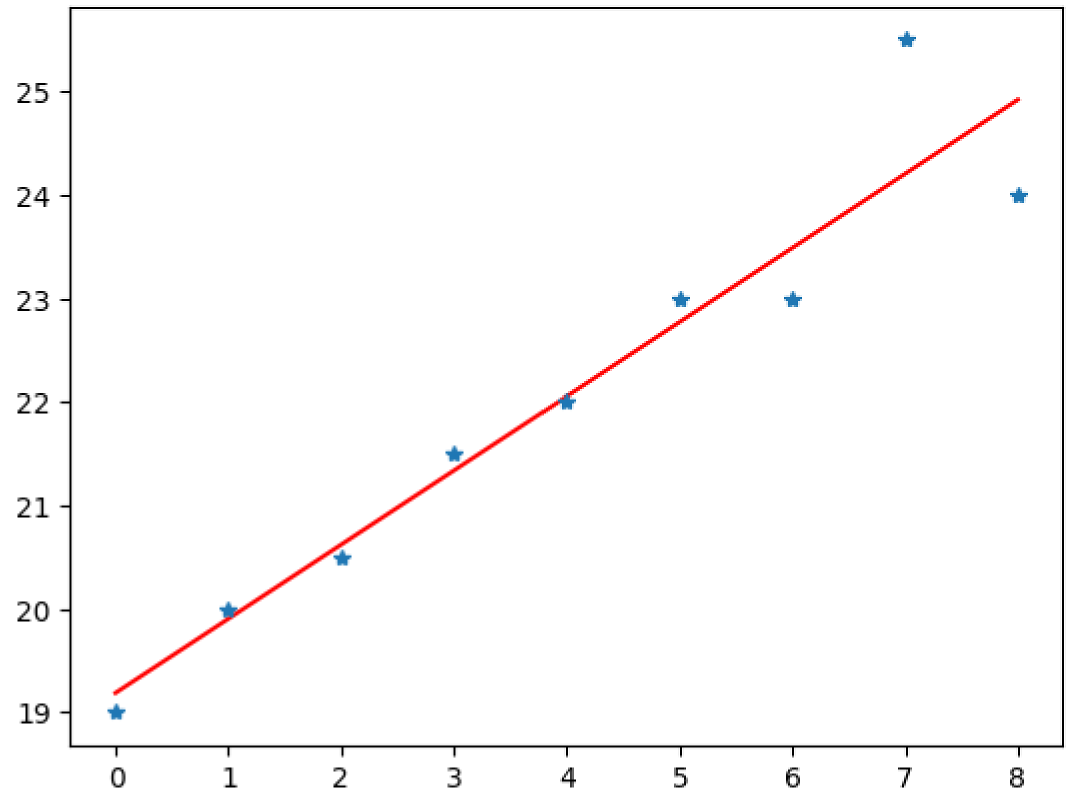
```
import numpy as np
import matplotlib.pyplot as plt

# x co-ordinates
x = np.arange(0, 9)

A  =  np.array([x,  np.ones(9)])
print("x=",x)
print("A=",A)
# linearly generated sequence
y = [19, 20, 20.5, 21.5, 22, 23, 23, 25.5, 24]

# obtaining the parameters of regression line
print("A transpose is", A.T)
w  =  np.linalg.lstsq(A.T,  y,  rcond=None)[0]
print(w)
```

```
x= [0 1 2 3 4 5 6 7 8]
A= [[0. 1. 2. 3. 4. 5. 6. 7. 8.]
 [1. 1. 1. 1. 1. 1.  1. 1. 1.]]
A transpose is [[0.     1.]
 [1.           1.]
 [2.           1.]
 [3.           1.]
 [4.           1.]
 [5.           1.]
 [6.           1.]
 [7.           1.]
 [8.          1.]]
```

In [ ]:

```
print(A.T@w)
import matplotlib.pyplot as plt
plt.plot(x,A.T@w,'r-')
plt.plot(x,y,'*')
plt.show()
```

```
[19.18888889 19.90555556 20.62222222  21.33888889 22.05555556 22.77222222
 23.48888889 24.20555556      24.92222222]
```

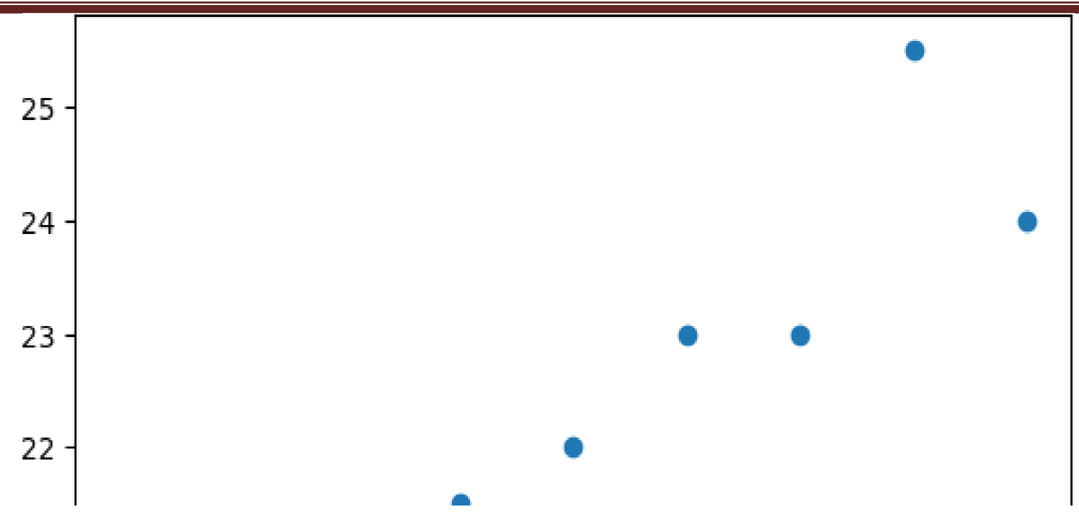


In [ ]:

```
plt.scatter(x,y)
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x7f09bee5cfd0>
```



Fit a line, y = mx + c, through some noisy data-points:

The simplest root finding algorithm is the bisection method. The algorithm applies to any continuous function $f(x)$ on an interval $[a, b]$ where the value of the function changes sign from a to b. The idea is simple: divide the interval in two, a solution mustexist within one subinterval, select the subinterval where the sign of $f(x)$ changes andrepeat.

We can rewrite the line equation as y = Ap, where A = [[x 1]] and p = [[m], [c]]. Now uselstsq to solve for p:

In [ ]:

```
x = np.array([0, 1, 2, 3])
y = np.array([-1, 0.2, 0.9, 2.1])
```

In [ ]:

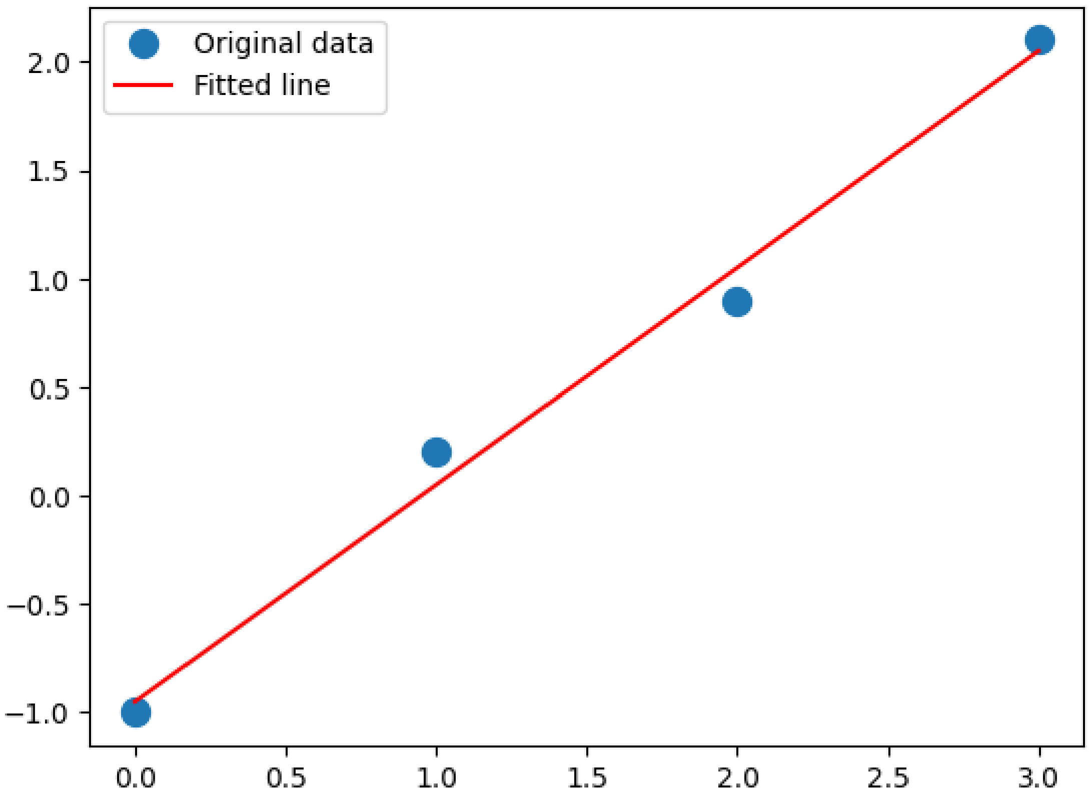
```
on=np.ones(4)print(x,on)
A  =  np.vstack([x,  np.ones(4)]).T
display(A)
m,c=np.linalg.lstsq(A,y)[0]
print("m=",m)
print("c=",c)
```

```
[0 1 2 3] [1. 1. 1. 1.]
array([[0.,  1.],
       [1.,  1.],
       [2.,  1.],
       [3.,  1.]])
m= 0.9999999999999999
```

```
<ipython-input-14-5cfe87b58da3>:5: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None
`, to keep using the old, explicitly pass `rcond=-1`.
c= -0.9499999999999997
```

In [ ]:

```
import matplotlib.pyplot as plt

plt.plot(x, y, 'o', label='Original data', markersize=10)
plt.plot(x, m*x + c, 'r', label='Fitted line') plt.legend()
plt.show()
```



In [ ]:

```
def f(x):
    return (x**2 - 1)
from scipy import optimize
root1 = optimize.bisect(f, 0, 2)
root2 = optimize.bisect(f, -2, 0)
print(root1,root2)
```

```
1.0 -1.0
```

```
In [ ]:
def f(x):
    return (x**3 - 1)   # only one real root at x = 1
root  =  optimize.newton(f,  1.5)
root
```

```
Out[]: .0000000000000016
```

```
In [ ]:
root=optimize.newton(f,1.5,fprime2=lambda x: 6 * x)root
```

```
Out[]: .0000000000000016
```

```
In [ ]:
root=optimize.newton(f,1.5,fprime=lambda x: 3 * x**2)root
```

```
Out[ ]: 1.0
```

```
In [ ]:
root=optimize.newton(f,1.5,fprime=lambda x: 3 * x**2,

                            fprime2=lambda x: 6 * x)
```

```
Out[ ]: 1.0
```

```
In [ ]:
def fun(x):

    return [x[0]   + 0.5 * (x[0] - x[1])**3 - 1.0,

            0.5 * (x[1] - x[0])**3 + x[1]]
```

```
Out[ ]: array([0.8411639, 0.1588361])
```

```
In [ ]:
solution=optimize.root(f,1.5)
solution.x
```

```
Out[ ]: array([1.])
```

Exercises

1. https://www.atozmath.com/example/CONM/LeastSquare.aspx?he=h (first three problems)

2. Find the roots of the equation using newton, bisect (with and without using formula)

   a. f(x) = 2x3 – 2x – 5

   b. f(x) = x3 – x – 1

   c. f(x) = x3 + 2x2 + x – 1

   d. f(x) = x3 – 2x – 5

   e. f(x) = x3 – x + 1

3. Solve the following equation cos(x)=x**3 by using builtin function and without using function

4. Solve x**2=2 by using builtin function and without using function

5. Find the square and cube root using builtin function and without using function

6. Find the root of x2 + y = 5 and y2 + x = 10

Viva questions:

1. What are the different methods for solving systems of linear equations?
2. Under what conditions does a given system of equation have a unique solution, infinite solutions, or no solution?
3. Explain the geometric intrepration of solving the system of linear equations.

EX NO. 7:

Data plotting (2D,3D) of various mathematical functions

Aim: To demonstrate various plots in python using pyplot, mplot3d.

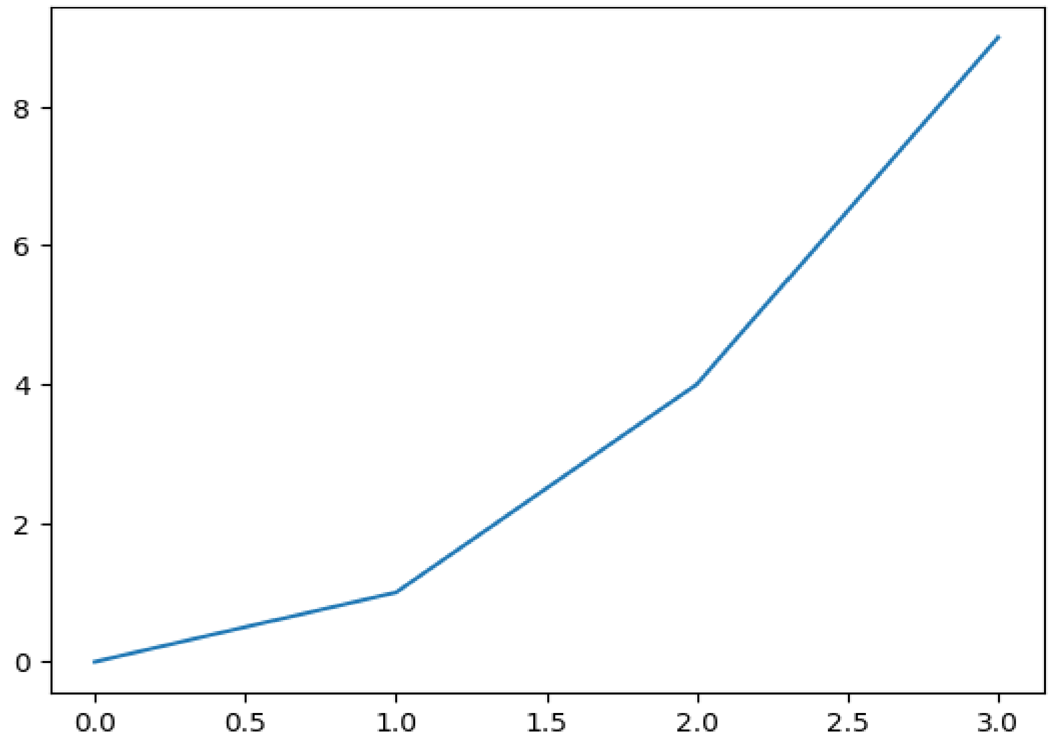
https://piktochart.com/blog/types-of-graphs/

https://www.w3schools.com/python/matplotlib_intro.asp

question;

In [ ]:

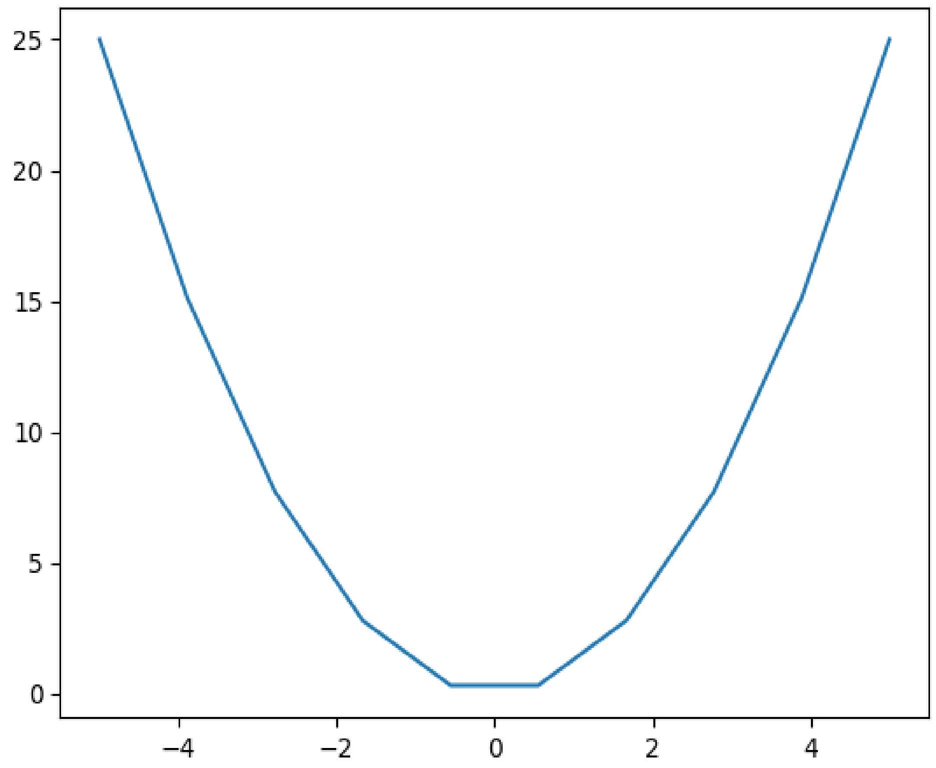
```
import numpy as np
import  matplotlib.pyplot  as  plt
x=[0,1,2,3]
y=[0,1,4,9]
plt.plot(x,y)

plt.show()
```



In [ ]:

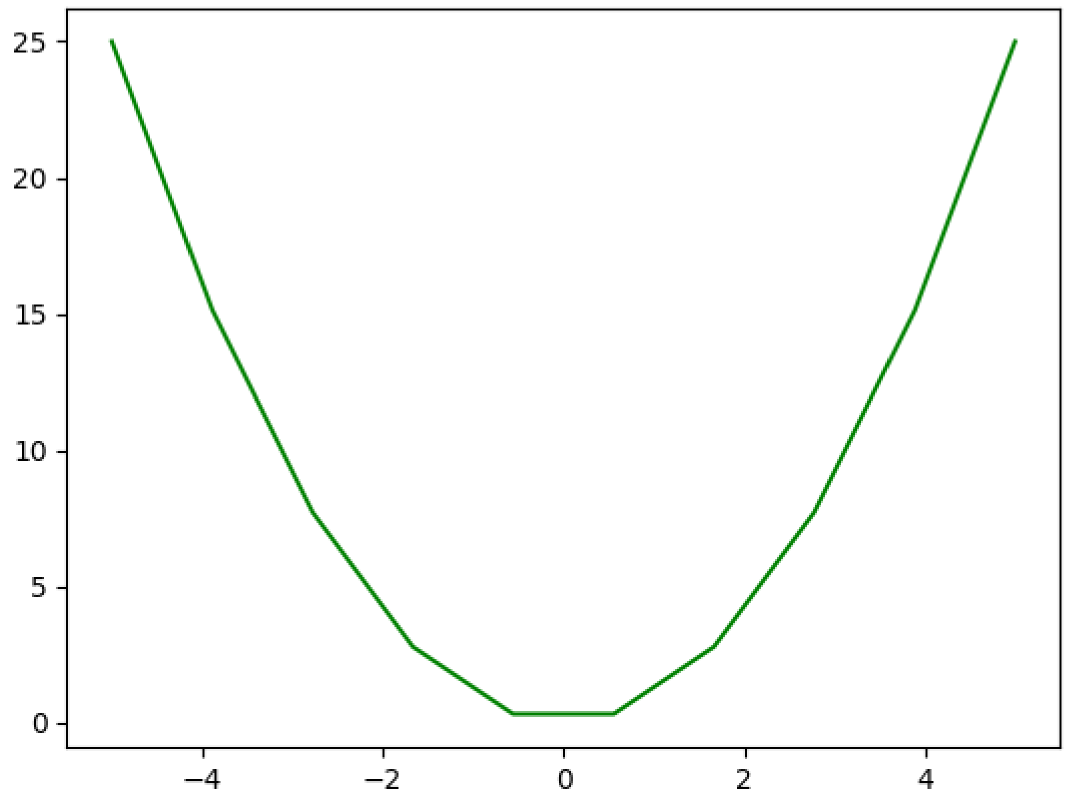
```
print(x)

x=np.linspace(-5,5,10)
plt.plot(x,x**2)
plt.show()
```

```
[0, 1, 2, 3]
```



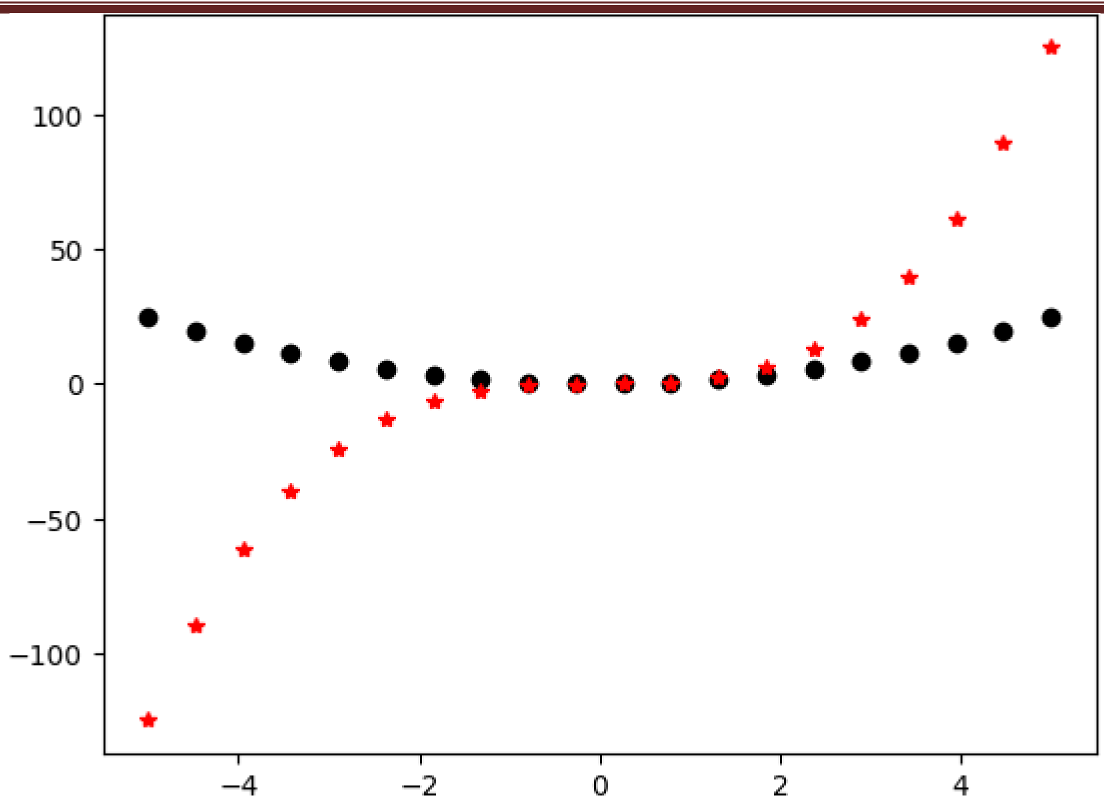
In [ ]:

```
plt.plot(x,x**2, "g-") plt.show()
```



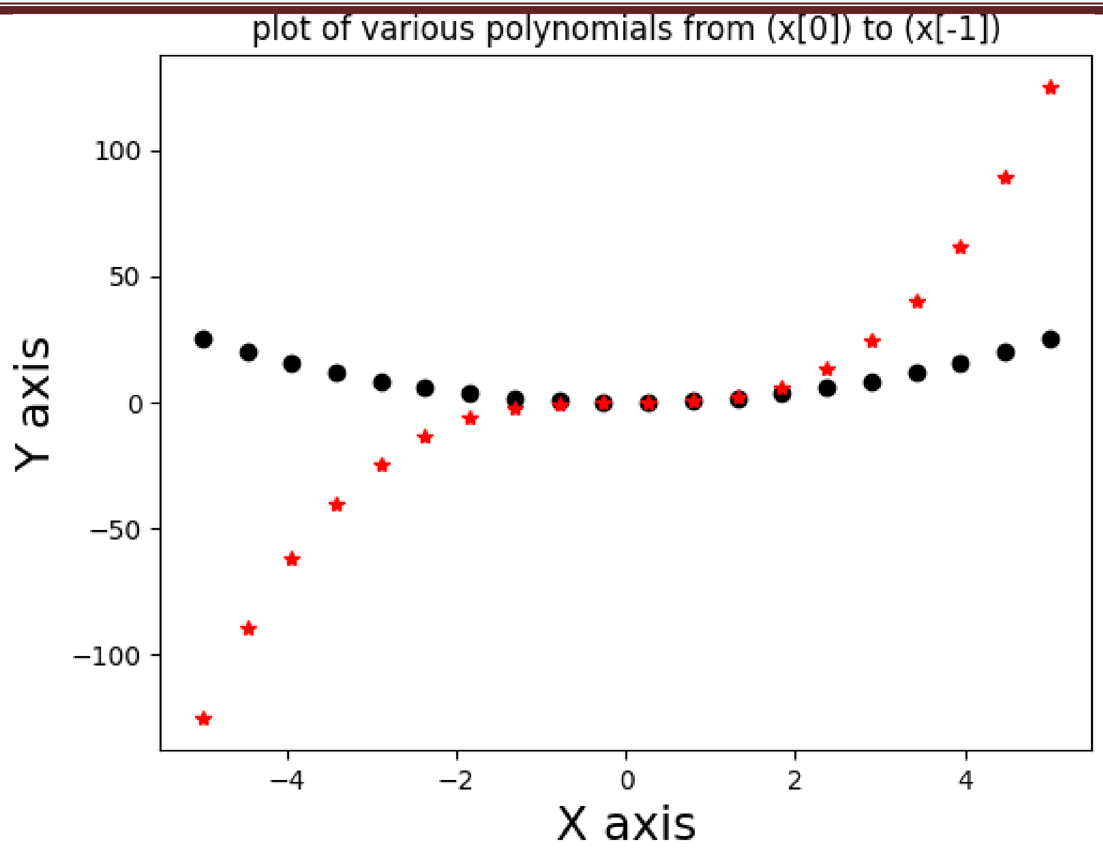
In [ ]:

```
x=np.linspace(-5,5,20)
plt.plot(x,x**2,"ko")
plt.plot(x,x**3,"r*")
plt.show()
```



In [ ]:

```
#plt.figure(figsize= (10,6))
x=np.linspace(-5,5,20)
plt.plot(x,x**2,"ko")

plt.plot(x,x**3,"r*")
plt.title("plot of various polynomials from (x[0]) to (x[-1])") plt.xlabel("X axis",
fontsize=18)
plt.ylabel("Y axis", fontsize=18) plt.show()
```



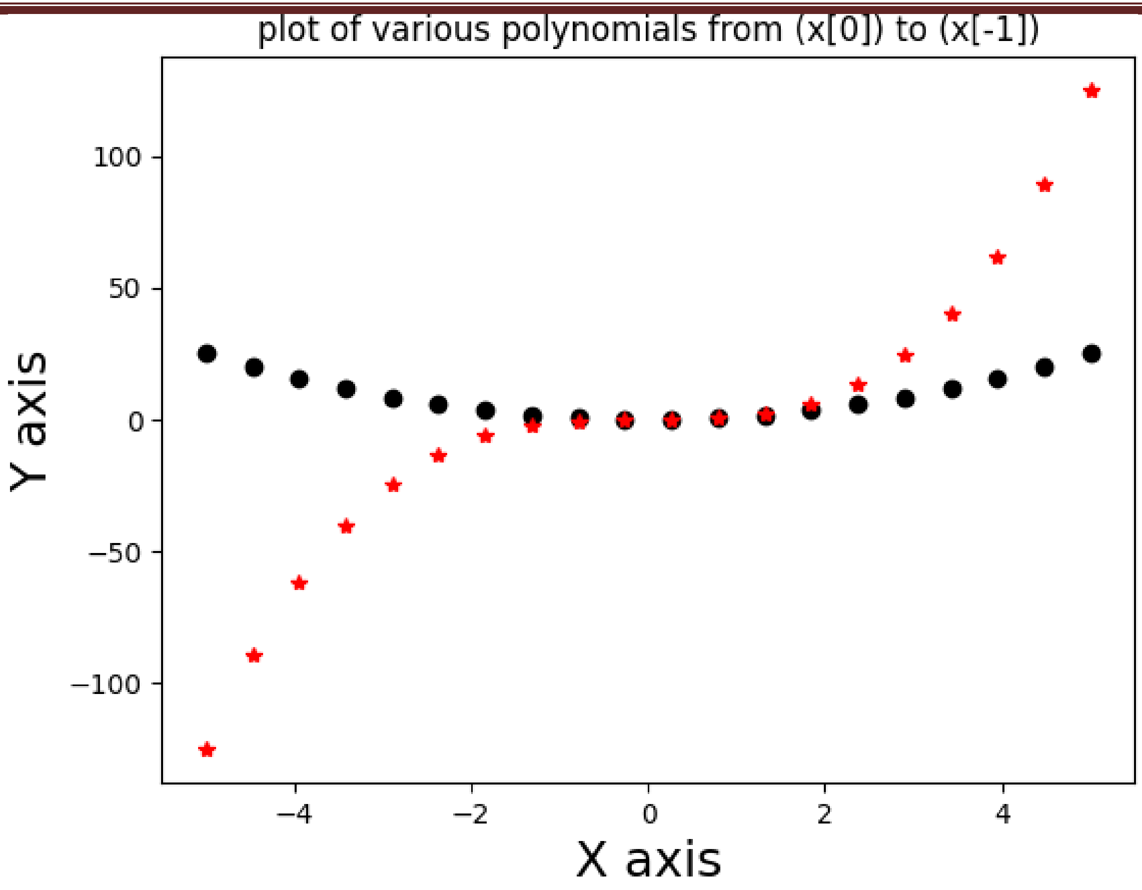
In [ ]:

```
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid ', 'bmh',
'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'gr ayscale', 'seaborn-v0_8',
'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'sea born-v0_8-dark', 'seaborn-v0_8-
dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn- v0_8-deep', 'seaborn-v0_8-muted',
'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel', 'seaborn-v0_8-
poster', 'seaborn-v0_8-talk', 'seaborn-v0_8
-ticks', 'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

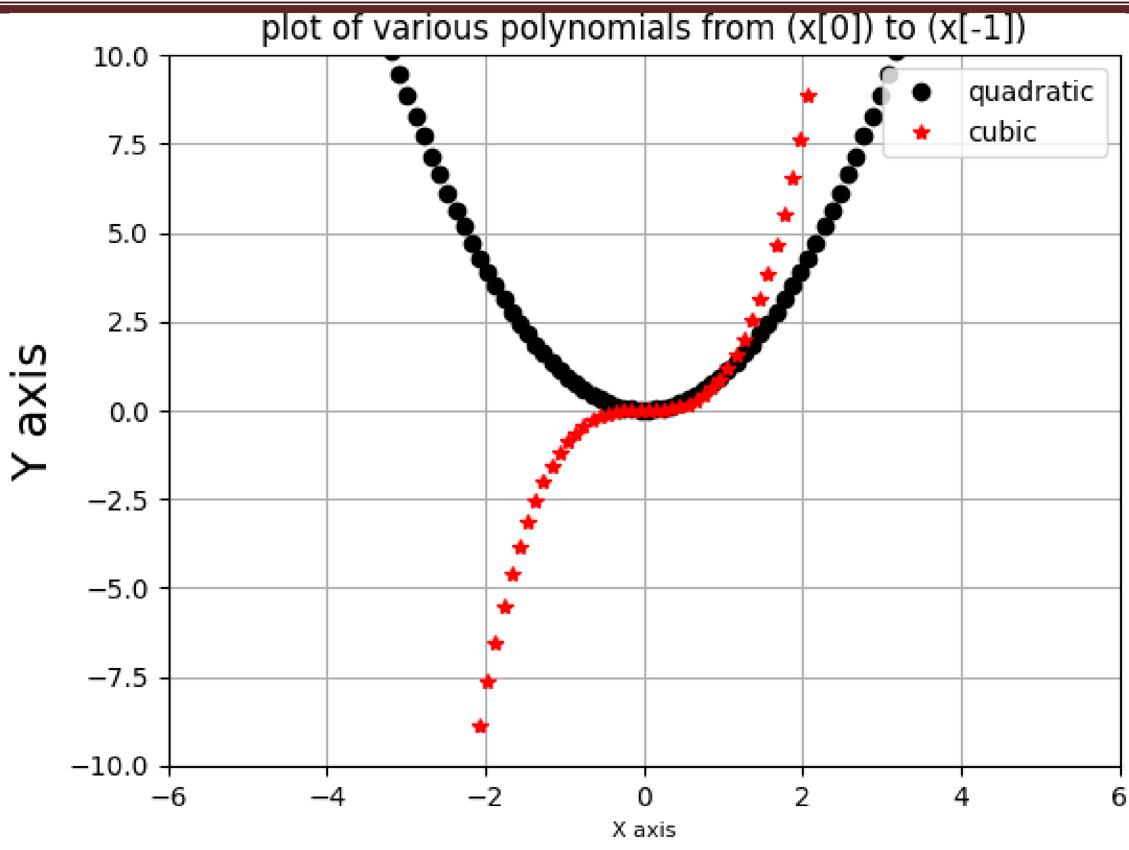
In [ ]:

```
#plt.style.use('seaborn-v0_8-notebook')


#plt.figure(figsize=(10,6))
x=np.linspace(-5,5,20)
plt.plot(x,x**2,"ko")

plt.plot(x,x**3,"r*")
plt.title("plot of various polynomials from (x[0]) to (x[-1])") plt.xlabel("X axis",
fontsize=18)
plt.ylabel("Y axis", fontsize=18) plt.show()
```



In [ ]:

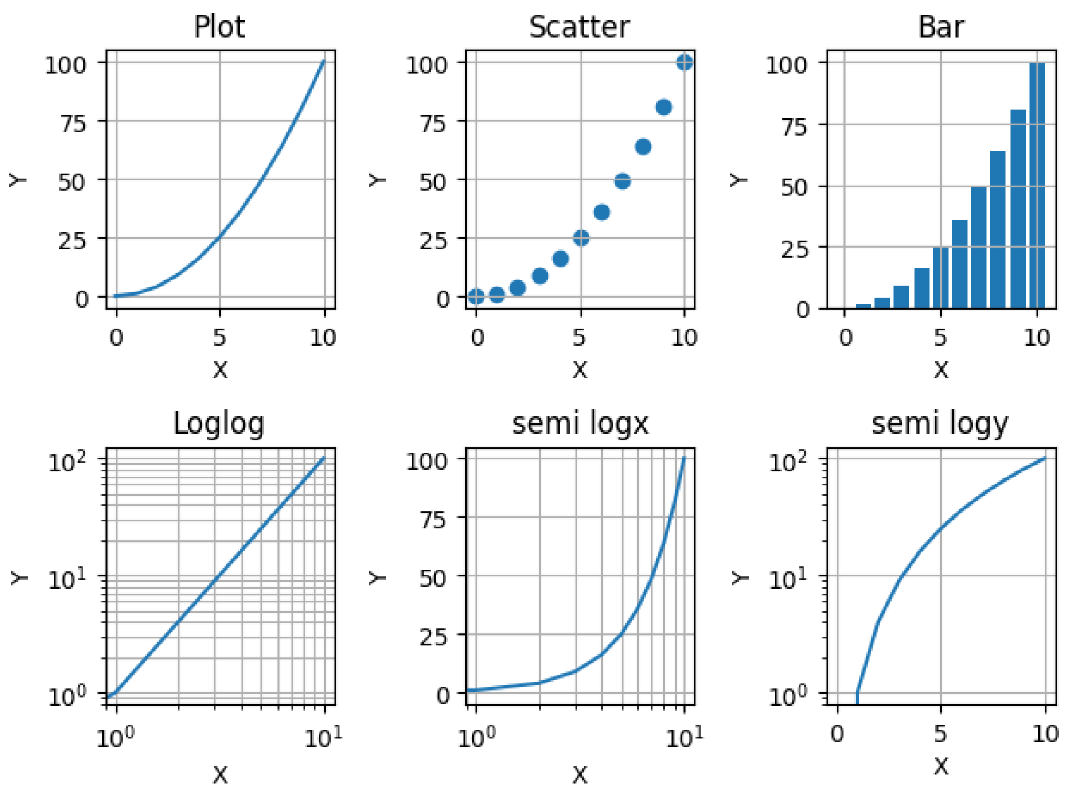
```
#plt.figure(figsize=(10,6)) x=np.linspace(-5,5,100)
plt.plot(x,x**2,"ko", label="quadratic")
plt.plot(x,x**3,"r*", label="cubic")

plt.title("plot of various polynomials from (x[0]) to (x[-1])") plt.xlabel("X axis", fontsize=8)
plt.ylabel("Y        axis",        fontsize=18)
plt.legend(loc=1)
plt.xlim(-6,6)
plt.ylim(-10,10)
plt.grid() plt.show()
```


plot of various polynomials from (x[0]) to (x[-1])
Y axis
X axis
quadratic
cubic
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−6
−4
−2
0
2
4
6

In [ ]:

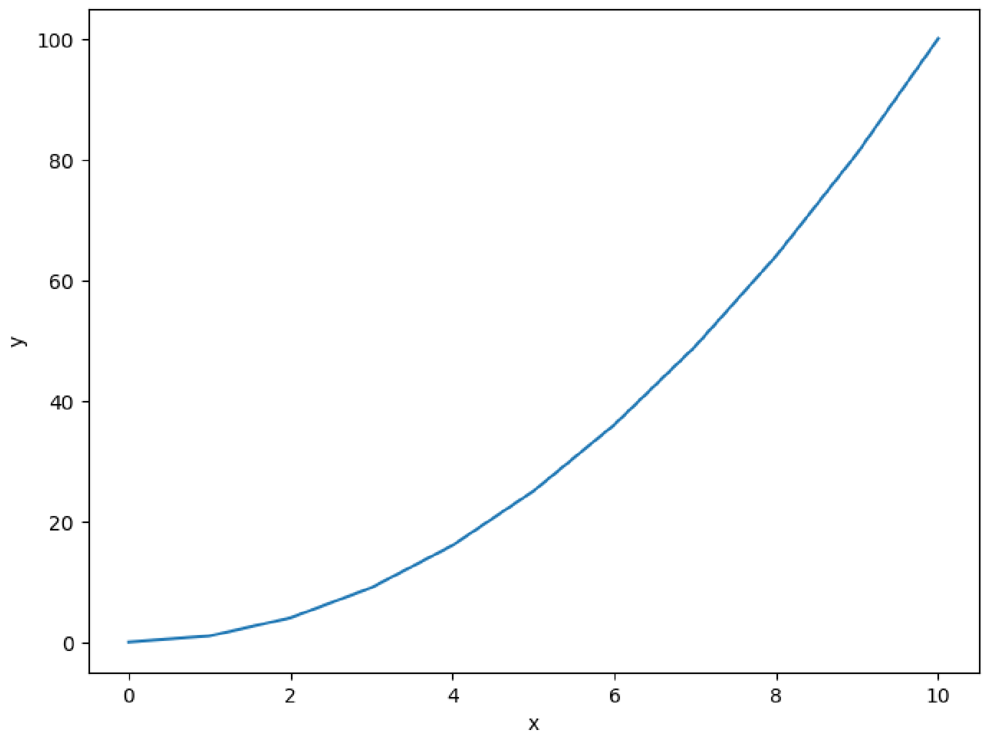
```
x=np.arange(11)
print(x)
y=x**2 #plt.figure(figsize=(14,8))
plt.subplot(2,3,1) plt.plot(x,y)
plt.title("Plot") plt.xlabel("X")

plt.ylabel("Y") plt.grid()
###########################

plt.subplot(2,3,2)     plt.scatter(x,y)
plt.title("Scatter") plt.xlabel("X")
plt.ylabel("Y") plt.grid()
############################33

plt.subplot(2,3,3)
plt.bar(x,y)
plt.title("Bar")
plt.xlabel("X")
plt.ylabel("Y") plt.grid()
##########################3

plt.subplot(2,3,4)     plt.loglog(x,y)
plt.title("Loglog") plt.xlabel("X")
plt.ylabel("Y") plt.grid(which="both")
##########################

plt.subplot(2,3,5)
plt.semilogx(x,y)     plt.title("semi
logx") plt.xlabel("X")
plt.ylabel("Y") plt.grid(which="both")
#############################
##3

plt.subplot(2,3,6)
plt.semilogy(x,y)     plt.title("semi
logy") plt.xlabel("X")
plt.ylabel("Y") plt.grid()

##################################33

plt.tight_layout()
plt.show()
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
```



In [ ]:

```
plt.figure(figsize=(8,6))
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("y")
plt.savefig("newplot.pdf")
```

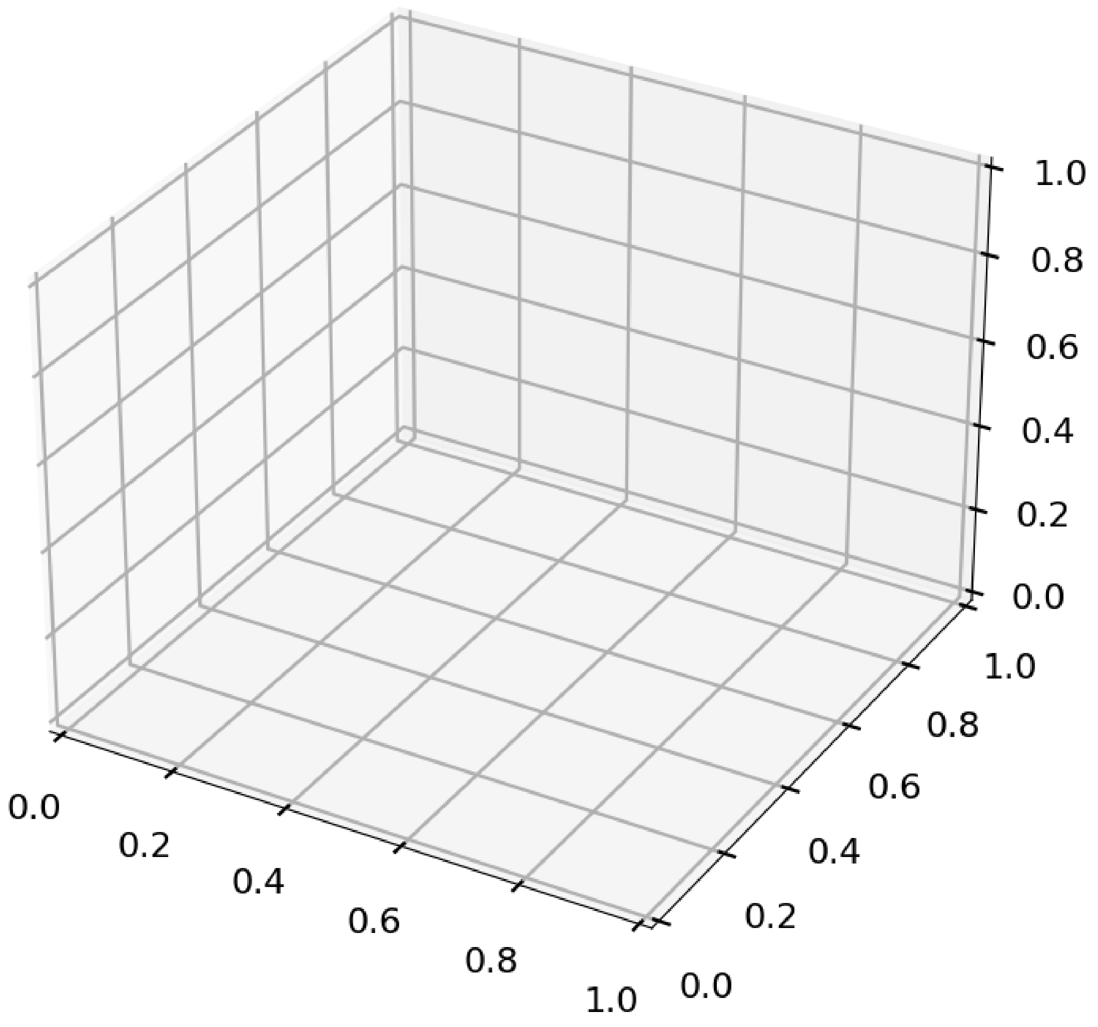


100
80
60
40
20
0
y
0
2
4
6
8
10
x

In [ ]:

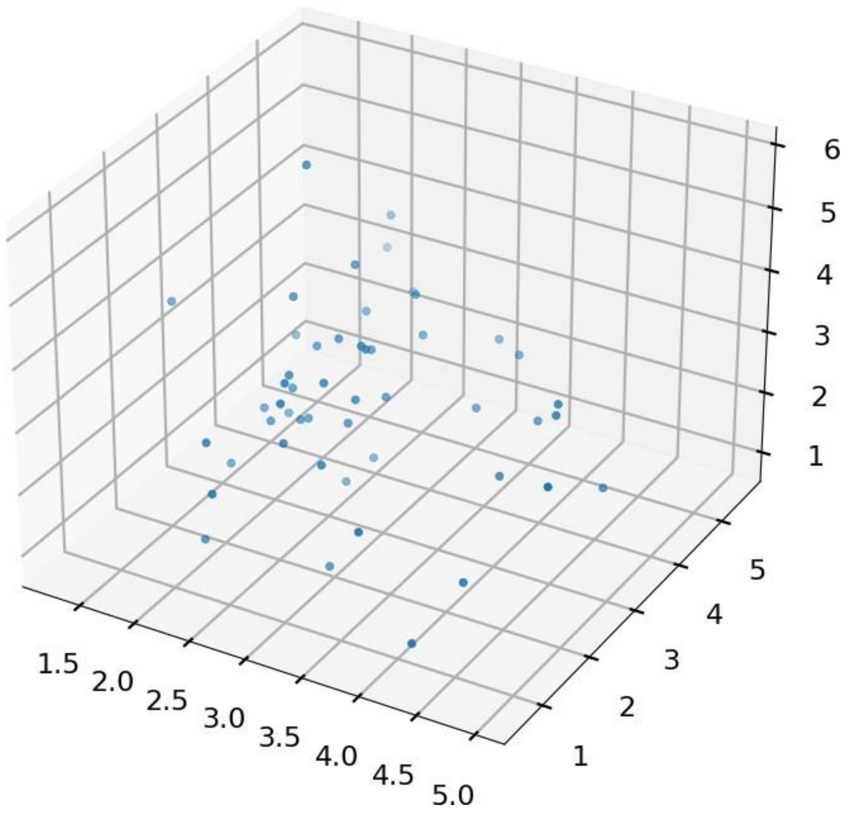
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
%matplotlib inline plt.style.use("seaborn-
poster") #fig=plt.figure(figsize=(10,10))
ax=plt.axes(projection="3d") plt.show()
```

```
<ipython-input-11-b63968fd7152>:6: MatplotlibDeprecationWarning: The seaborn sty
les shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the
styles shipped by seaborn. However, they will remain available as 'seabor n-v0_8-
<style>'. Alternatively, directly use the seaborn API instead.
```



In [ ]:

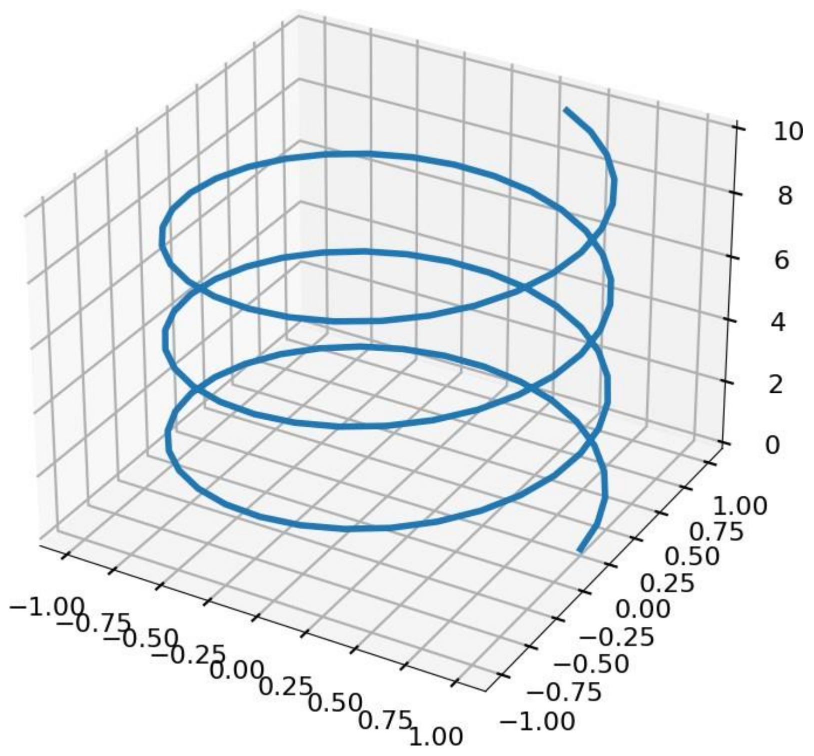
```
np.random.seed(31)

mu = 3
n=50

x = np.random.normal(mu, 1, size=n)
y = np.random.normal(mu, 1, size=n)
z = np.random.normal(mu, 1, size=n)

ax = plt.axes(projection='3d')
#plt.plot(x,y,z) ax.scatter3D(x, y, z);
```



In [ ]:

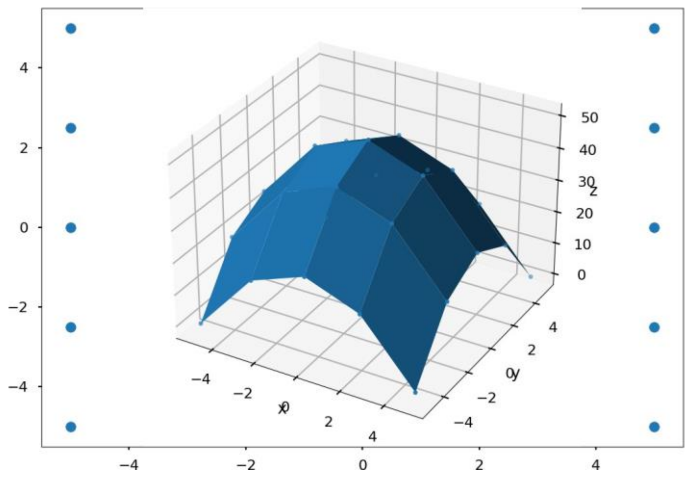
```
omega = 2

z_line   =  np.linspace(0,   10,   100)
x_line    =     np.cos(omega*z_line)
y_line = np.sin(omega*z_line)

ax              =              plt.axes(projection='3d')
ax.plot3D(x_line, y_line, z_line, lw=4)
```

```
Out[ ]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x7cfa0125e860>]
```


10
8
6
4
2
0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

In [

```
# 3 D surface plots.

# Example surface z=50-(x**2+y**2)

def function_z(x, y):
      return 50 - (x**2 + y**2)

N = 5

x_values  =  np.linspace(-5,  5,  N)
y_values = np.linspace(-5, 5, N)


X,  Y  =  np.meshgrid(x_values,  y_values)

X
X.shape
plt.scatter(X,Y)

Z = function_z(X,  Y)

ax     =     plt.axes(projection='3d')
ax.set_xlabel('x')  ax.set_ylabel('y')
ax.set_zlabel('z');

#ax.plot_surface(X, Y, Z);
ax.plot_surface(X,Y,Z)
ax.scatter(X,Y,Z)   print(X)

print(Y)

print(Z)
#ax.plot(x,z)

#plt.plot(x,y,z)
```

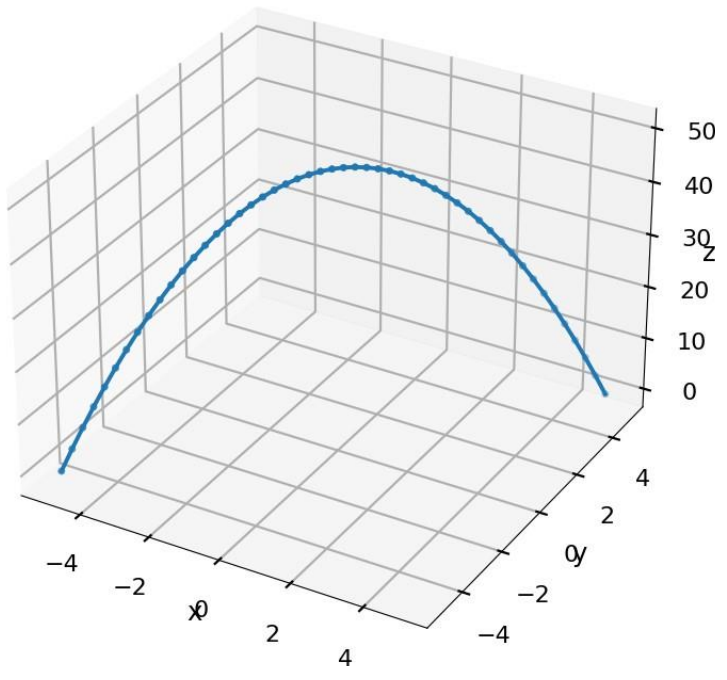


```
[[-5.  -5.   -5.  -5.  -5.    ]
 [-2.5 -2.5  -2.5 -2.5     -2.5]
 [ 0.   0.    0.   0.      0. ]
 [ 2.5  2.5   2.5  2.5      2.5]
 [  5.    5.   5.   5.    5.    ]]
[[  0.  18.75 25.    18.75   0.   ]
 [18.75  37.5   43.75  37.5   18.75]
 [25.   43.75   50.   43.75  25.   ]
 [18.75  37.5   43.75  37.5   18.75]
 [ 0.   18.75   25.  18.75   0.   ]]
```

In [ ]:

```
# 3 D surface plots.

# Example surface z=50-(x**2+y**2)

def function_z(x, y):
      return 50 - (x**2 + y**2)

N = 50
x_values  =  np.linspace(-5,  5,  N)
y_values = np.linspace(-5, 5, N)

z_values=function_z(x_values,y_values)
ax = plt.axes(projection='3d')
#ax.plot_surface(x_values,y_values,z_values);
#ax.plot_trisurf(x_values,y_values,z_values) ax.set_xlabel('x')

ax.set_ylabel('y') ax.set_zlabel('z');

ax.plot3D(x_values,y_values,z_values) ax.scatter3D(x_values,y_values,z_values)
```

Out[ ]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7cfa01fc6bc0>



In [ ]:

```
# Sample code for generation of first example
import numpy as np
# from matplotlib import pyplot as plt # pyplot
imported for plotting graphs

x = np.linspace(-4, 4, 9)

# numpy.linspace creates an array of # 9
linearly placed elements between # -4 and 4,
both inclusive
y = np.linspace(-5, 5, 11) print(x)
print(y)
# The meshgrid function returns # two 2-
dimensional arrays
x_1, y_1 = np.meshgrid(x, y)

print("x_1      =      ")
print(x_1)
print("y_1=”)
print(y_1)
```

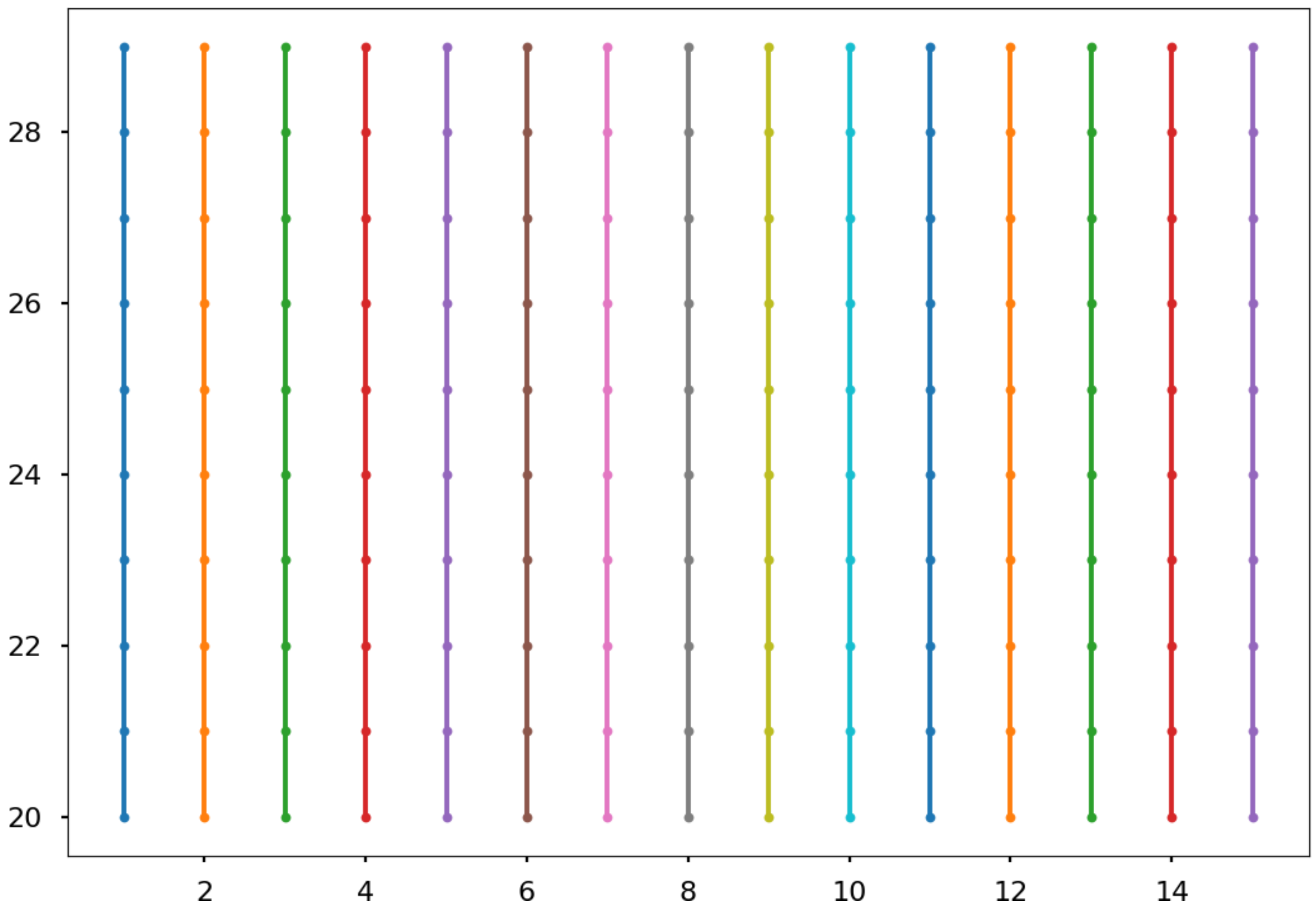
```
[-4. -3. -2. -1.  0.  1.  2.  3.  4.]
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
 x_1 =
 [[-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]
  [-4. -3. -2. -1.  0.  1.  2.  3.   4.]]
y_1 =
 [[-5. -5. -5. -5. -5. -5. -5. -5.    -5.]
  [-4. -4. -4. -4. -4. -4. -4. -4.    -4.]
  [-3. -3. -3. -3. -3. -3. -3. -3.    -3.]
  [-2. -2. -2. -2. -2. -2. -2. -2.    -2.]
  [-1. -1. -1. -1. -1. -1. -1. -1.    -1.]
  [ 0.  0.  0.  0.  0.  0.  0.  0.     0.]
  [ 1.  1.  1.  1.  1.  1.  1.  1.     1.]
  [ 2.  2.  2.  2.  2.  2.  2.  2.     2.]
  [ 3.  3.  3.  3.  3.  3.  3.  3.     3.]
  [ 4.  4.  4.  4.  4.  4.  4.  4.     4.]
  [ 5.  5.  5.  5.  5.  5.  5.     5.  5.]]
```

In [ ]:

```
[  1.   2.   3.   4.   5.   6.   7.   8.  9.  10.  11.  12.  13.  14.  15.] [20.  21.  22.  23.  2
```

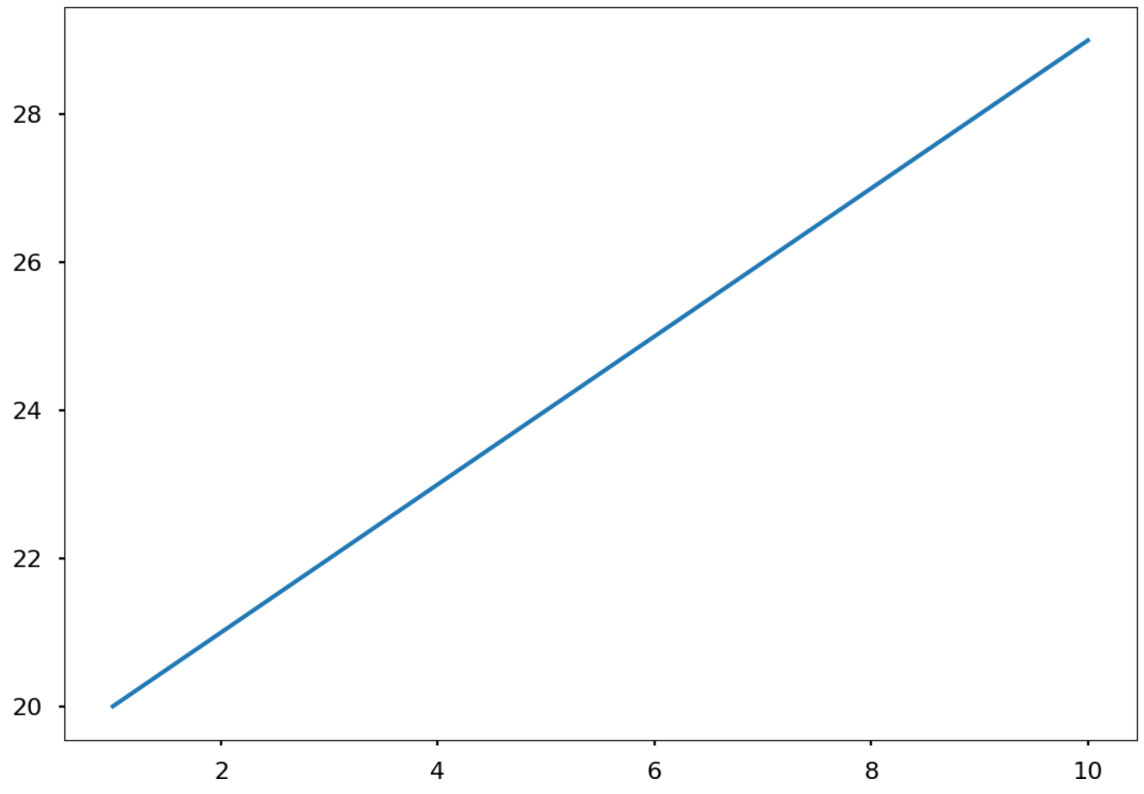
```
X   =   np.linspace(1,15,15)

Y  =  np.linspace(20,29,10)

xx, yy = np.meshgrid(X,Y)

fig = plt.figure()

ax = fig.add_subplot()
#fig.add_subplot()


#ax.plot(xx, yy, ls="None", marker=".")

ax.plot(xx,yy,  marker=".")

print(X,Y)
print(xx,yy)
ax.plot()
plt.show()
x10=X[ :10]
print(x10)
```

```
[21.   21.  21.  21.  21.  21.  21.  21.  21.  21.  21.  21.  21.  21.  21.]
[22.   22.  22.  22.  22.  22.  22.  22.  22.  22.  22.  22.  22.  22.  22.]
[23.   23.  23.  23.  23.  23.  23.  23.  23.  23.  23.  23.  23.  23.  23.]
[24.   24.  24.  24.  24.  24.  24.  24.  24.  24.  24.  24.  24.  24.  24.]
[25.   25.  25.  25.  25.  25.  25.  25.  25.  25.  25.  25.  25.  25.  25.]
[26.   26.  26.  26.  26.  26.  26.  26.  26.  26.  26.  26.  26.  26.  26.]
[27.   27.  27.  27.  27.  27.  27.  27.  27.  27.  27.  27.  27.  27.  27.]
[28.   28.  28.  28.  28.  28.  28.  28.  28.  28.  28.  28.  28.  28.  28.]
[29.  29.  29.  29.  29.  29.  29.  29.  29.  29.  29.  29.  29.  29.  29.]]
```


28
26
24
22
20
2
4
6
8
10
12
14

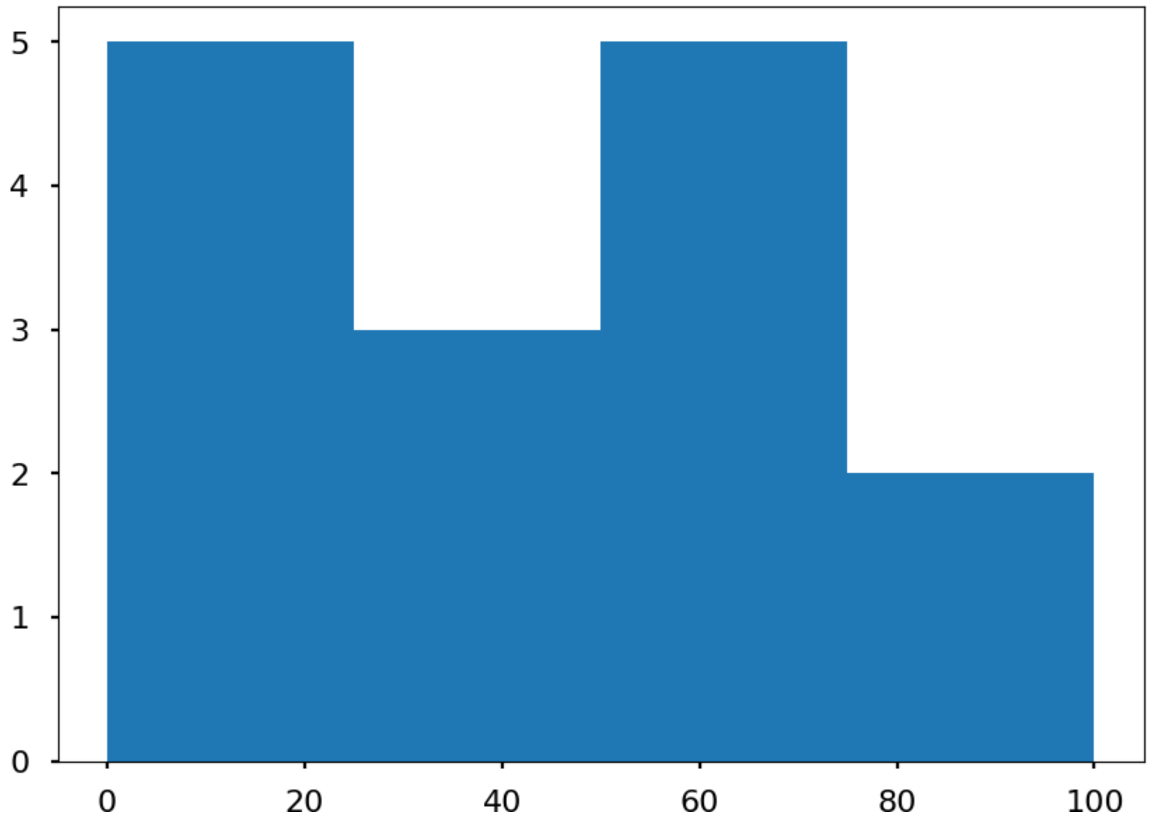
In [ ]:

```
plt.plot(X[:10],Y)
plt.show()
```



In [ ]:

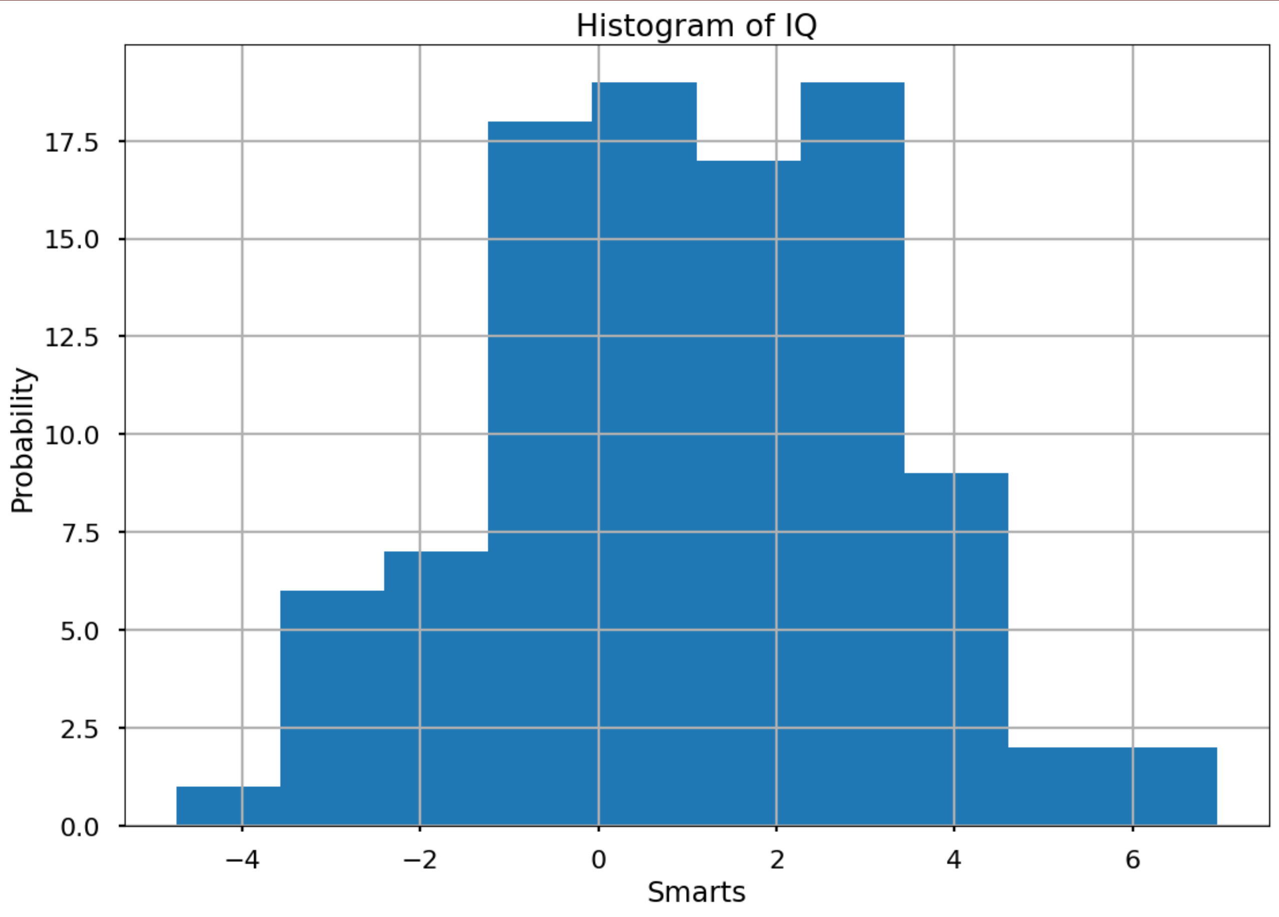
```
#https://jakevdp.github.io/PythonDataScienceHandbook/04.05-histograms-and-bin

from matplotlib import pyplot as plt import numpy as
#  Creating  dataset
a  = np.array([22, 87, 5, 43, 56,
               73, 55,   54, 11,
               20, 51, 5, 79, 31,
               27])


# Creating histogram

fig, ax = plt.subplots(figsize =(10, 7))



# Show plot
```



In [ ]:

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
print(np.random.randn(10))
x=np.random.normal(1,2,100)
print(x)
# the histogram of the data  #n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)

counts,   bins   =   np.histogram(x)
print(counts,bins)
plt.hist(x)     plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram     of     IQ')
plt.grid(True)
plt.show()
```

```
[-1.39460318  0.46318788 -0.29772803  1.47129311 -1.86923793  -0.76730646
 -1.53419169 -0.74964839  0.24670888        2.08641303]
[-0.73402284 -0.06940621 -2.09109744 -0.23280979  2.04193275  -1.18461842
 -0.51515467  2.61528409 -2.79877906  2.702573    3.28239125  -0.49649636
  1.21824963 -1.03942523  3.68832993 -1.91063543  0.37937346  -1.03011764
  3.1533374   2.73037934  2.3195281  -0.04475738  2.14882731   1.65520687
  0.10262818  1.85973807  3.58617925 -0.40901499 -1.41823497   0.48658311
  0.95116077  0.69945522 -2.34689165 -1.60469666  2.78669135   3.72799388
 -0.02096635 -0.35173997  0.1739947   1.84821033  2.66917384   3.66537861
 -0.27452196 -1.66815113  2.19721523  3.02828851  3.30477238   0.29101004
 -2.47849112  0.05056384 -3.23650403 -2.98360121 -3.3368014   -4.74023654
  3.51723698  0.59886917 -2.31232515  1.18550523  2.50962533   2.47612097
  1.02234363  1.1541722   0.93806938  1.12777143  2.45455192  -0.75159793
  2.95486581 -1.08638471  0.41985237 -0.34938161  0.72890284   0.53673426
  5.15093062  4.31592566  1.79051216  2.41908667  0.23486113   0.68910317
  3.49675903 -0.50351085  1.59254059  2.72512504  1.31455084   4.97393245
  3.92008861  1.79289603 -0.23552585 -1.02237197 -0.47279324   3.12981817
  2.51489218  0.00990866  6.94268228  1.27199861 -2.47572629   6.35872567
  2.48769726  1.29548356  4.2820773  2.26417524]
[ 1  6  7 18 19 17 19  9  2  2] [-4.74023654 -3.57194466 -2.40365278 -1.23536089
 -0.06706901  1.10122287
  2.26951475  3.43780664  4.60609852  5.7743904          6.94268228]
```



Exercises

1. plot tanx, sin inverse
2. plot e^x x:[0,10]
3. Plot x**2 * y**2 * z**2 =1
4. plot sinx * cosy
5. plot x^2 * y^3 ; x:[-1,1]; y:[0,3]
6. plot Arcsin(xy)
7. stock market, cricket scores corona virus graphs ,etc

Viva questions:

1. When to use bar chart, scatter plot, line graph, histogram?
2. Explain the role of meshgrid (numpy library) in 3 D plotting.
3. Explain various pyplot functions.
4. Explain title, xlable, ylable, etc used in plotting.

EXP NO. 8

Test the convergence of infinite series

A Taylor series is a series expansion of a function about a point. A one-dimensional Taylor series is an expansion of a real function f(x) about a point x=a is given by f(x)=f(a)+f^'(a)(x-a)+(f^('')(a))/(2!)(x-a)^2+(f^((3))(a))/(3!)(x-a)^3+...+(f^((n))(a))/(n!)(x-a)^n+.... (1) https://mathworld.wolfram.com/TaylorSeries.html

If a=0, the expansion is known as a Maclaurin series. https://www.mathsisfun.com/algebra/taylor-series.html

```
from sympy.abc import n, x
from sympy import expand, factor, series
fx= x**2
s1 = series(fx, x, 0, 10)
print(s1)
s1 = series(fx, x, 2, 10)
print(s1)
```

```
x**2
4*x + (x - 2)**2 - 4
```

```
from sympy import *
f = E**x
s = series(f, x, 0, 10)
print(s)
```

```
ne = 1 + x + x**2/2 + x**3/6 + x**4/24 + x**5/120 + x**6/720+ x**7/5040 + x**8/40320 +
x**9/362880
print(ne.subs(x, 2).evalf())
```

```
1 + x + x**2/2 + x**3/6 + x**4/24 + x**5/120 + x**6/720 + x**7/5040 + x**8/40320 + x**9/362880
+ O(x**10)
7.38871252204586
```

```
limit(x**n/factorial(n),n,oo)
l = Limit(s, x, 4)
print(l)
print(l.doit().evalf())
print(f.subs(x, 4).evalf())
```

```
Limit(1 + x + x**2/2 + x**3/6 + x**4/24 + x**5/120 + x**6/720 + x**7/5040 + x**8/40320 +
x**9/362880 + O(x**10), x, 4)
54.1541446208113
54.5981500331442
```

```
fx = 1/(1-x)
s1 = series(fx, x, 0, 10)
print(s1)
nplus1thterm = x**(n+1)
nthterm = x**(n)
ratiotest = nplus1thterm/nthterm
print(ratiotest)
l = Limit(ratiotest, n, oo)
print(l)
print(l.doit().evalf())
print("convergence test using nthratios:|{}| < 1".format(ratiotest.simplify()))
```

```
1 + x + x**2 + x**3 + x**4 + x**5 + x**6 + x**7 + x**8 + x**9 + O(x**10)
x**(-n)*x**(n + 1)
Limit(x**(-n)*x**(n + 1), n, oo, dir='-')
```

x

convergence test using nthratios:|x| < 1

https://tutorial.math.lamar.edu/problems/calcii/convergenceofseries.aspx

https://www.dummies.com/article/academics-the-arts/math/calculus/determining-whether-a-taylor-series-is-convergent-or-divergent-178073/

Viva questions:

1. How can you use a Taylor series to approximate value of e ?
2. What is the difference between a sequence and a series?
3. Explain the terms "convergent series" and "divergent series."
4. What is the significance of testing the convergence of a series?

EX NO. 9

Intro to calculus and examine minima, maxima and saddle points of a given function.

In [ ]:

```
fromsympy.abcimportn,x
#from sympy.abc.
#/content/sample_data
```

In [ ]:

```
import math
print(math.sqrt(25),  math.sqrt(7))
```

```
5.0  2.6457513110645907
```

In [ ]:

```
import                    sympy
print(sympy.sqrt(7))
print(sympy.sqrt(50))
```

```
sqrt(7) 5*sqrt(2)
```

In [ ]:

```
from   sympy   import    *
x=Symbol('x')
expr=integrate(x**2,x)
print(expr)
expr2=integrate(x**x,x)
print(expr2) print(sqrt(7))
```

```
x**3/3
Integral(x**x,       x)
sqrt(7)
```

In [ ]:

```
print(Rational(3/4))
Rational(0.2)
```

```
3/4
```

Out[ ]: $\frac{3602879701896397}{18014398509481984}$

In [ ]:
```
3602879701896397*5
```

Out[ ]:
```
18014398509481985
```

In [ ]:
```
Rational(0.2).limit_denominator(100)
```

Out[ ]: $\frac{1}{5}$

In [ ]:
```
sympify("10/5+4/3")
```

Out[ ]: $\frac{10}{3}$

In [ ]:
```
from sympy.abc import *
#sympy.abc.

expr=a/b
expr.evalf(5,subs={a:100,b:3})
```

Out[ ]:
```
33.333
```

In [ ]:
```
help(expr.evalf)
```

```
Help on method evalf in module sympy.core.evalf:

evalf(n=15, subs=None, maxn=100, chop=False, strict=False, quad=None, verbose=False)
method of sympy.core.mul.Mul instance
    Evaluate the given formula to an accuracy of *n* digits.

    Parameters
    ==========

    subs : dict, optional
        Substitute numerical values for symbols, e.g.
        ``subs={x:3, y:1+pi}``. The substitutions must be given as a
        dictionary.

    maxn : int, optional
        Allow a maximum temporary working precision of maxn digits.

    chop : bool or number, optional
        Specifies how to replace tiny real or imaginary parts in
        subresults by exact zeros.

        When ``True`` the chop value defaults to standard precision.

        Otherwise the chop value is used to determine the
        magnitude of "small" for purposes of chopping.

        >>> from sympy import N
        >>> x = 1e-4
        >>>   N(x,   chop=True)
        0.000100000000000000
        >>>   N(x,   chop=1e-5)
        0.000100000000000000
        >>> N(x, chop=1e-
        4)0

    strict : bool, optional
        Raise ``PrecisionExhausted`` if any subresult fails to
        evaluate to full accuracy, given the available maxprec.

    quad : str, optional
        Choose algorithm for numerical quadrature. By default,
        tanh-sinh quadrature is used. For oscillatory integrals on
        an infinite interval, try ``quad='osc'``.
    verbose : bool, optional Print
        debug information.

    Notes
    =====

    When Floats are naively substituted into an expression, precision
    errors may adversely affect the result. For example,adding 1e16 (a
    Float) to 1 will truncate to 1e16; if 1e16 is then subtracted, the
    result will be 0.
    That is exactly what happens in the following:

    >>> from sympy.abc import x, y, z
    >>> values = {x: 1e16, y: 1, z: 1e16}
    >>> (x + y - z).subs(values)0

    Using the subs argument for evalf is the accurate way to
    evaluate such an expression:

    >>> (x + y - z).evalf(subs=values)
    1.00000000000000
```

In [ ]:

```
a,b=symbols('a            b')
expand((a+b)**2)
```

Out[ ]: $a^2 + 2ab + b^2$

In [ ]:
```
x,y,z=symbols('x          y          z')
expr=(x**2*z+ 4*x*y*z + *y**2*z)
factor(expr)
```

Out[ ]: $z(x + 2y)^2$

In [ ]:
```
a,b=symbols('a             b')
print(expand((a+b)**2))
print(factor(x**2+2*x+1))
```

```
a**2   +   2*a*b   +
b**2(x + 1)**2
```

In [ ]:
```
expr=(x**2*z     +     4*x*y*z     +
4*y**2*z)factor_list(expr)
```

Out[ ]: (1, [(z, 1), (x + 2*y, 2)])In [ ]:

```
expr1=x**2+2*x+1
expr2=x+1
cancel(expr1/expr2)
```

Out[ ]: $x + 1$

In [ ]:
```
simplify(expr1/expr2)
```

Out[ ]: $x + 1$

In [ ]:
```
from sympy import trigsimp, sin, cos
from sympy.abc import x, y
expr = 2*sin(x)**2 + 2*cos(x)**2
trigsimp(expr)
```

Out[ ]: 2

In [ ]:
```
expr=factorial(x)/factorial(x    -    3)
print(expr)
combsimp(expr)
```

```
factorial(x)/factorial(x - 3)
```

Out[ ]: $x\,(x-2)\,(x-1)$

In [ ]:

```
from sympy import diff, sin, exp
from   sympy.abc   import   x,y
expr=x*sin(x*x)+1
print(expr)diff(expr)
```

```
x*sin(x**2) + 1
```

Out[ ]: $2x^2\cos\left(x^2\right)+\sin\left(x^2\right)$

In [ ]:

```
diff(x**4,x,3)
```

Out[ ]: $24x$

In [ ]:

```
expr=x**2   +   x   +   1
integrate(expr, x)
```

Out[ ]: $\frac{x^3}{3}+\frac{x^2}{2}+x$

In [ ]:

```
expr=exp(-x**2)
integrate(expr,(x,0,oo) )
```

Out[ ]: $\frac{\sqrt{\pi}}{2}$

In [ ]:

```
expr=exp(-x**2        -        y**2)
integrate(expr,(x,0,oo),(y,0,oo))
```

Out[ ]: $\frac{\pi}{4}$

In [ ]:

```
from sympy import fourier_transform, exp
from sympy.abc import x, k
expr=exp(-x**2)
fourier_transform(expr, x, k)
```

Out[ ]: $\sqrt{\pi} e^{-\pi^2 k^2}$

In [ ]:

```
from sympy.integrals import laplace_transform from
sympy.abc import t, s, a

laplace_transform(t**a, t, s)
```

Out[ ]: (gamma(a + 1)/(s*s**a), 0, re(a) > -1)

In [ ]:

```
#prove that (1-sinx)/(sinx cotx)=cosx/(1+sinx)
exp1=1-sin(x)
exp2=sin(x)*cot(x)
exp3=trigsimp((exp1)/exp2)
print(exp3)
print(trigsimp(cos(x)/(1+sin(x))))
trigsimp(exp3*(1+sin(x)))
```

```
-tan(x) + 1/cos(x)
cos(x)/(sin(x) + 1)
```

Out[ ]: $\cos(x)$

In [ ]:

```
#prove sec2(x)/cot(x)-tan3(x)=tan(x)
exp1=trigsimp(sec(x)**2/cot(x)-tan(x)**3)
trigsimp(exp1/tan(x))
```

Out[ ]: 1

In [ ]:

```
trigsimp(sin(x-90))
```

Out[ ]: $\sin(x - 90)$

In [ ]:
```
expand_trig(sin(pi/2-x))
```

Out[ ]: $\sin\left(\frac{\pi}{2}\right)\cos(x) - \sin(x)\cos\left(\frac{\pi}{2}\right)$

In [ ]:
```
from sympy import
*x,y=symbols('x y')
Eq(x,y)
```

Out[ ]: $x = y$

In [ ]:
```
solveset(Eq(x**2-9,0),  x)
```

Out[ ]: {−3, 3}

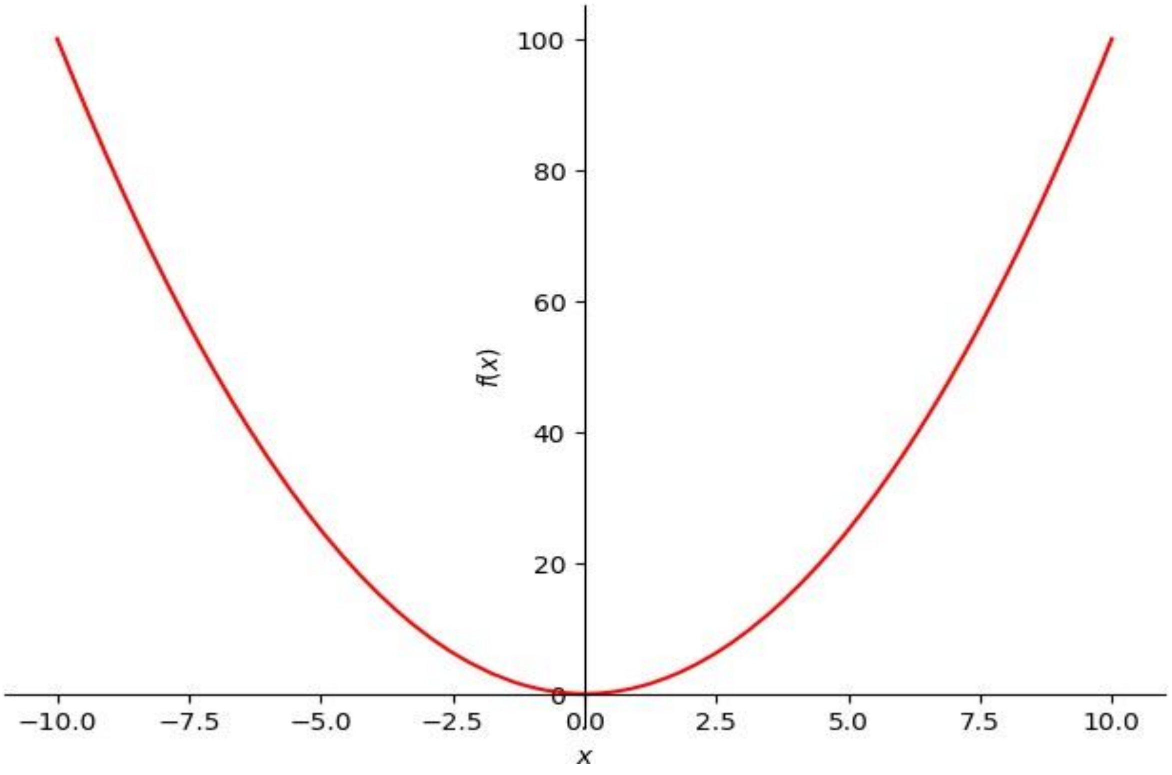
In [ ]:
```
x,y=symbols('x  y')
linsolve([Eq(x-y,4),Eq( x + y ,1) ], (x, y))
```

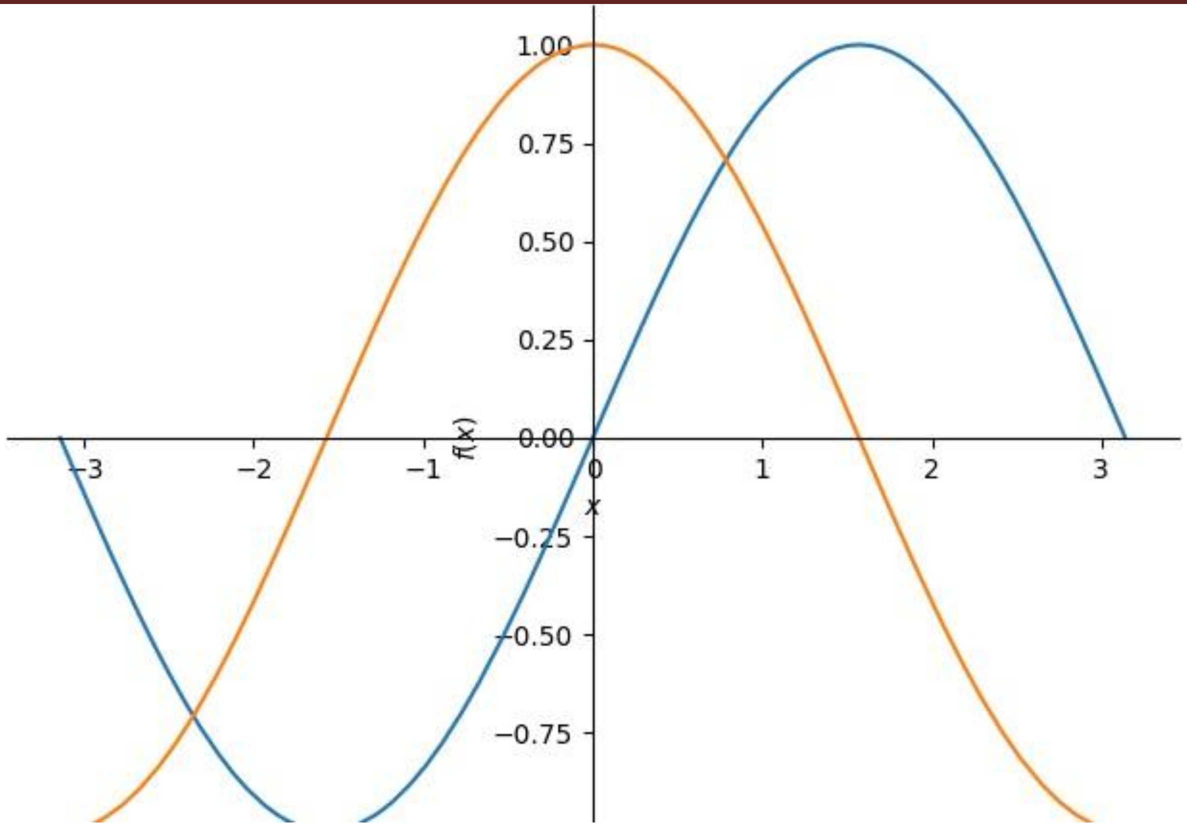
Out[ ]: $\left\{\left(\frac{5}{2}, -\frac{3}{2}\right)\right\}$

In [ ]:
```
a,b=symbols('a  b')
nonlinsolve([a**2 + a, a - b], [a, b])
```

Out[ ]: {(−1, − 1), (0, 0)}

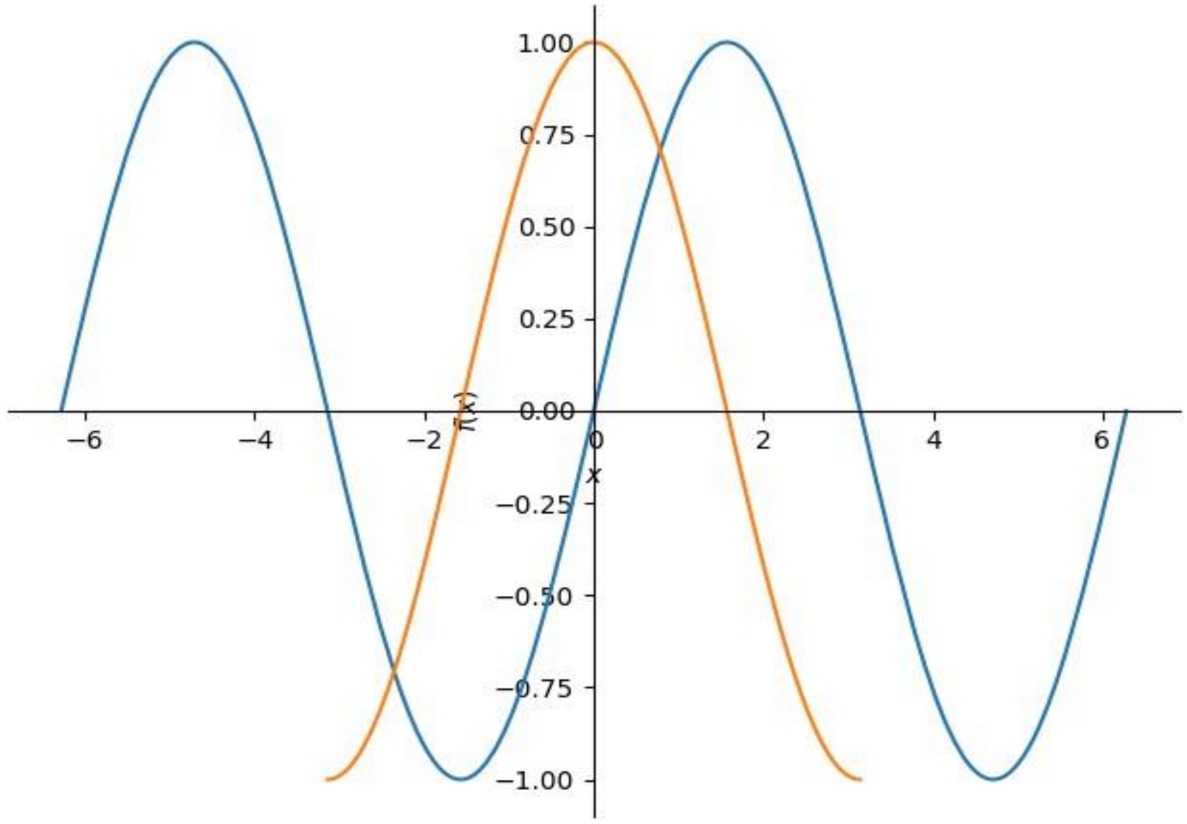
In [ ]:
```
from  sympy.plotting  import  plot
from       sympy       import       *
x=Symbol('x')
plot(x**2,  line_color='red')
plot( sin(x),cos(x), (x, -pi, pi))
```





In [ ]:

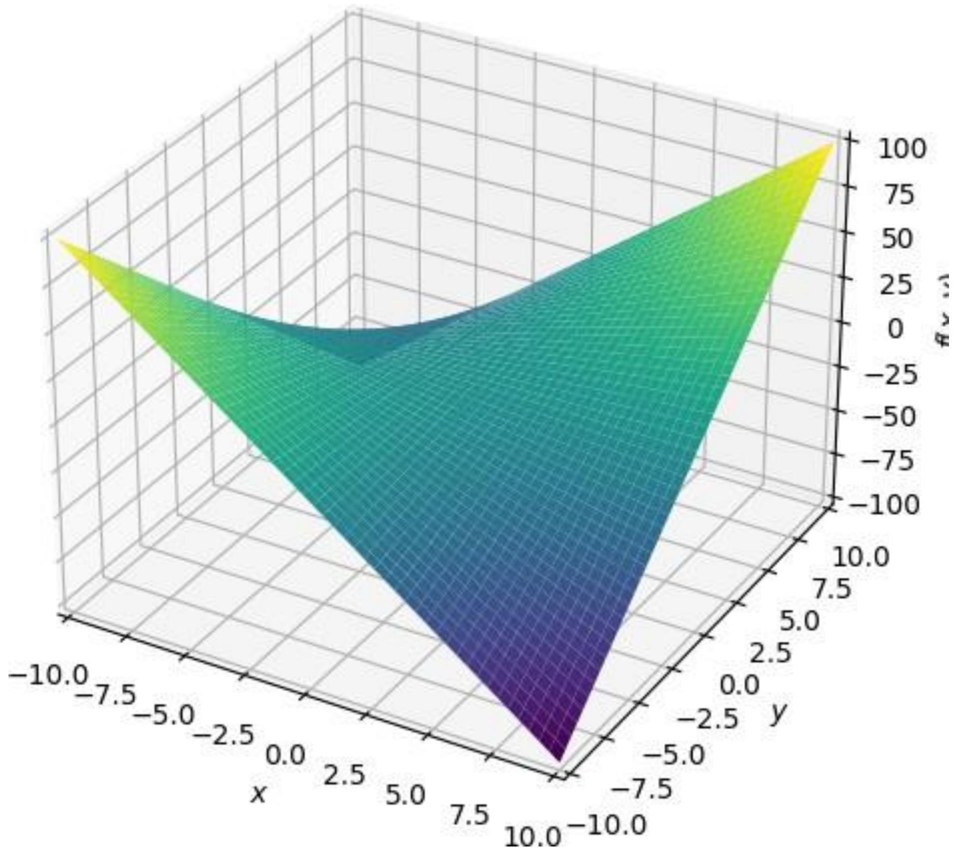
```
plot( (sin(x),(x, -2*pi, 2*pi)),(cos(x), (x, -pi, pi)))
```



Out[ ]: <sympy.plotting.plot.Plot at 0x7fd33978efb0>

In [ ]:

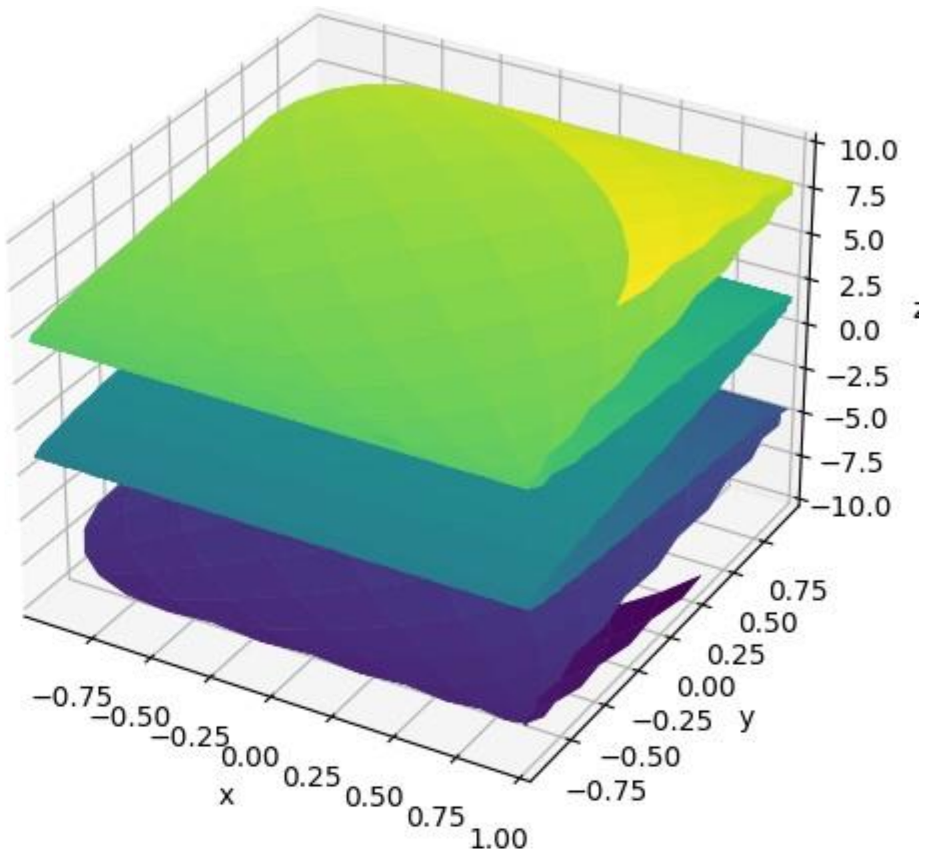
```
from sympy.plotting import plot3d
x,y=symbols('x y')
plot3d(x*y, (x, -10,10), (y, -10,10))
```



Out[ ]: <sympy.plotting.plot.Plot at 0x7fd3394d7520>

In [ ]:

```
#plot3d_parametric_surface(xexpr, yexpr, zexpr, rangex, rangey, kwargs)


from          sympy.plotting          import          plot3d_parametric_surface
plot3d_parametric_surface(cos(x+y), sin(x-y), x-y, (x, -5, 5), (y, -5, 5))
```



Out[ ]: <sympy.plotting.plot.Plot at 0x7fd339273430>

In [ ]:
```
print("\n\n")

pprint(      Integral(sqrt(1/x),x)       )
print("\n\n")

print((Integral(sqrt(1/x),x)))
```

$$\int \sqrt{\frac{1}{x}}\, dx$$

```
⌠
⎮     ___
⎮    ╱ 1
⎮   ╱  ─  dx
⎮ ╲╱   x
⌡
```

```
Integral(sqrt(1/x), x)
```

In [ ]:

```
x=Symbol("x")
print(x)
print(diff(x**4,   x,   x))
print(diff(x**4,   x,   2))
diff
```

```
x
12*x**
2
12*x**2
```

Out[ ]:

```
<function  sympy.core.function.diff(f,  *symbols,  **kwargs)>
```

finding the maximum and minimum for the unction f(x)

Step 1: Have your function, f(x), and find the derivative, f′(x).

Step 2: Set the derivative, f′(x), equal to zero.

Step 3: Solve for x — the result(s) are your critical point(s).

Step 4: Find the second derivative, i.e., find f’’(x). If f’’(x) < 0 then it’s a maximum. If f’’(x) > 0 then it’s a minimum

In [ ]:

```
#find the maximium and minimumfrom
sympy import * x=Symbol('x')

f=x**2 df=diff(f,x)
d2f=diff(df,x)
df_sol=solveset(Eq(df,0),              x)
print(df_sol)
for          i          in          df_sol:
    d2f_val=d2f.evalf(subs={x:i})
    if(d2f_val<0):
        print(i,"is   maximum")
    if(d2f_val>0):
        print(i,"is  minimum")
```

```
{0}
        0 is minium
```

Saddle point:

A saddle point, in the context of mathematics and optimization, refers to a critical point of a function where the function reaches a local extremum (maximum or minimum) in one direction and a different local extremum in another perpendicular direction. In other words, it's a point where the function curves upwards along one axis and downwards along another axis, creating a shape resembling a saddle.

Mathematically, for a function of two variables, a point (x, y) is a saddle point if it satisfies the condition that the partial derivatives of the function with respect to both x and y are zero at that point, and the second partial derivatives have opposite signs. This implies that the function has neither a local minimum nor a local maximum at that point, but rather a point of inflection.

Saddle points are important in optimization and the study of functions because they can present challenges for optimization algorithms. Traditional gradient-based optimization methods might be stuck at saddle points instead of finding the global or local extrema that are desired. However, directions along which the function decreases, making it possible for optimization algorithms to escape them and continue toward a more desirable solution, often surround saddle points.

In some cases, saddle points can be crucial in understanding the behavior of functions, especially in fields like economics, physics, and engineering where functions model complex systems. Saddle points can also be points of instability in dynamic systems, where small perturbations can lead to significant changes in behavior.

Exercises:

```
1. solve for A and B. if tan(A+B)=sqrt(3) and tan(A-B)=1/sqrt(3)
2. find the value of cos(x)cos(90-x)-sin(x)sin(90-x)
3. sin [(B + C)/2] = cos A/2. given A+B+c=180
4. (1 + tan2A)/(1 + cot2A) =  tan2A
5. a^x + log x . sinx differentiate, integrate  and plot it.
6. the tangent line to f(x)=7x^4+8x^6+2x at x=-1.
```

Exercises:

1 (derivatives)Try atleast 2 problems for each section(4 sections) from #https://www.math-exercises.com/limits-derivatives-integrals/derivative-of-a-function

2. (extereme points)Try atleast 4 problems from https://www.math-exercises.com/analysis-of-functions/local-extrema-of-a-function

3. Find the area of circle and triangle

4. Indefinite integrals - try atleast 2 from each section - https://www.math-exercises.com/limits-derivatives-integrals/indefinite-integral-of-a-function

5. Definite integrals - try atleast 2 from each section - https://www.math-exercises.com/limits-derivatives-integrals/definite-integral-of-a-function

Viva questions:

1. Define derivative and Integration.
2. What is the significance of sympy library?
3. How do we find maximum and minimum of a function.
4. Differentiate definite and indefinite integrals.

EX.NO 10 :

Application of definite integrals to area & volume

#https://www.youtube.com/watch?v=X-MVjALEAN4

#https://tutorial.math.lamar.edu/Problems/CalcI/AreaBetweenCurves.aspx

Finding the area of circle.

```
from sympy import *

x=Symbol('x')

print(integrate(sin(x)**2,(x,0,pi/2)))

integrate(cos(x)**2,(x,0,pi/2))
```

```
pi/4
```

Exercises:

Find the area of rectangle, triangle.

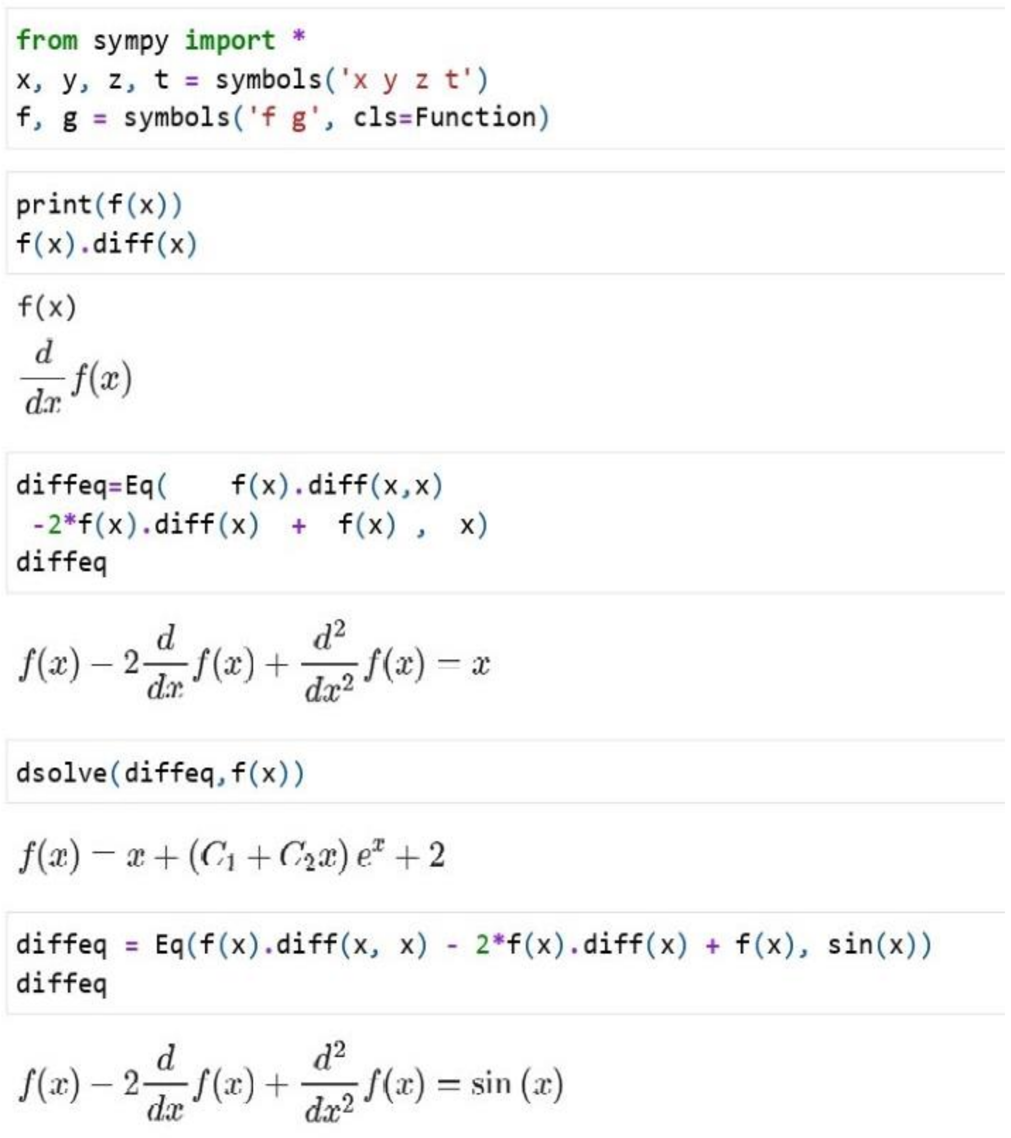
Find the volume of cube, cone, sphere.

Viva questions:

1. Explain how definite integrals can be used to find the area under a curve.
2. How can definite integrals be used to find the total distance traveled by an object over a given interval?
3. Explain the concept of using definite integrals to find the volume of a solid.
4. Explain the use of calculating the area and volume in 3D printing.

EX NO. 11

Solving differential equations



```
from sympy import *
x, y, z, t = symbols('x y z t')
f, g = symbols('f g', cls=Function)
```

```
print(f(x))
f(x).diff(x)
```

```
f(x)
```

$$\frac{d}{dx} f(x)$$

```
diffeq=Eq(    f(x).diff(x,x)
 -2*f(x).diff(x)  +  f(x) ,  x)
diffeq
```

$$f(x) - 2\frac{d}{dx} f(x) + \frac{d^2}{dx^2} f(x) = x$$

```
dsolve(diffeq,f(x))
```

$$f(x) - x + (C_1 + C_2 x)\, e^x + 2$$

```
diffeq = Eq(f(x).diff(x, x) - 2*f(x).diff(x) + f(x), sin(x))
diffeq
```

$$f(x) - 2\frac{d}{dx} f(x) + \frac{d^2}{dx^2} f(x) = \sin(x)$$

Viva questions:

1. What is a differential equation?
2. Explain the difference between an ordinary differential equation (ODE) and a partial differential equation (PDE).
3. How are differential equations used to model real-world phenomena?
4. Define the order of a differential equation.