Data Sciences using Python (05101305)

Dr.Kamini Solanki (Associate Professor)
Parul Institute of Computer Application - BCA

CHAPTER-4

Scientific computing with Python (scipy)

Introduction of scipy

- Scipy stand as scientific computing with python. It's a open source library for computing mathematical based operation which is not possible using numpy.
- Scipy is normally use for data science and data analytics based calculation.
- Scipy distributed under BSD license library.
- is pronounced as **Sigh pi**, and it depends on the Numpy, including the appropriate and fast N-dimension array manipulation.
- It provide many numerical integration based function for scientific calculation.
- It support **integration, gradient optimization, special functions, ordinary differential equation solvers, parallel programming tools**, and many more.
- **Why scipy and why not numpy?**

Installation of scipy

- Installing scipy using pip
  pip install scipy
- Installing scipy using anaconda
  - install anaconda first in you machine
  - open anaconda command prompt and enter below command
  - **conda install -c anaconda scipy (wait for some time package are going to download)**
- Installing scipy in macos
  - **sudo port install py35-numpy py35-scipy py35-matplotlib py35-ipython +notebook py35-pandas py35-sympy py35-nose**

  ** **For more about installation :**
  https://www.javatpoint.com/scipy-installation
  https://www.guru99.com/scipy-tutorial.html#4

Why scipy?

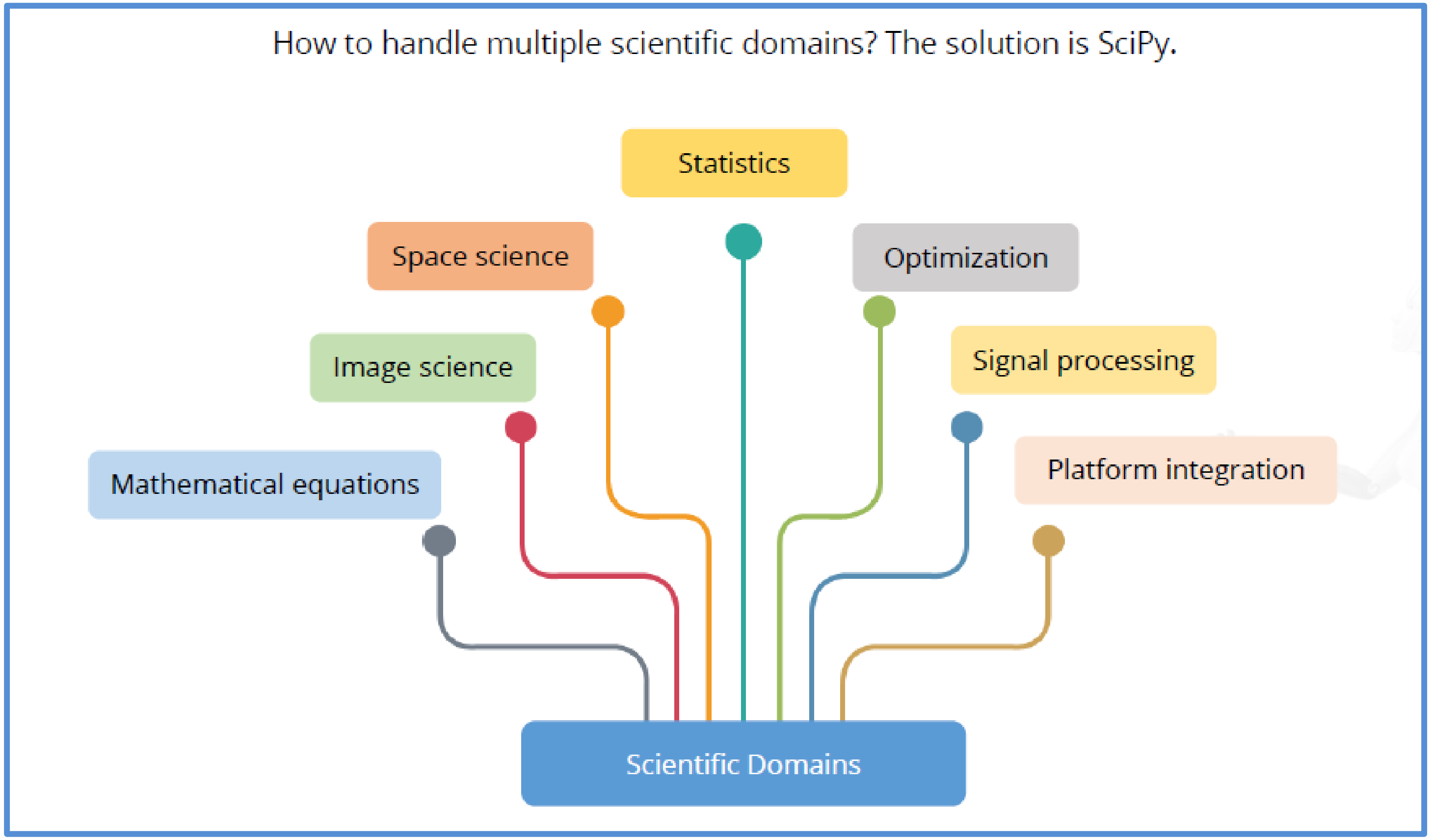


Image source : Simplilearn

Cont..

Scipy have many inbuilt sub packages which is use to handle integration problem, algebraic problem, statistics and many more problem.

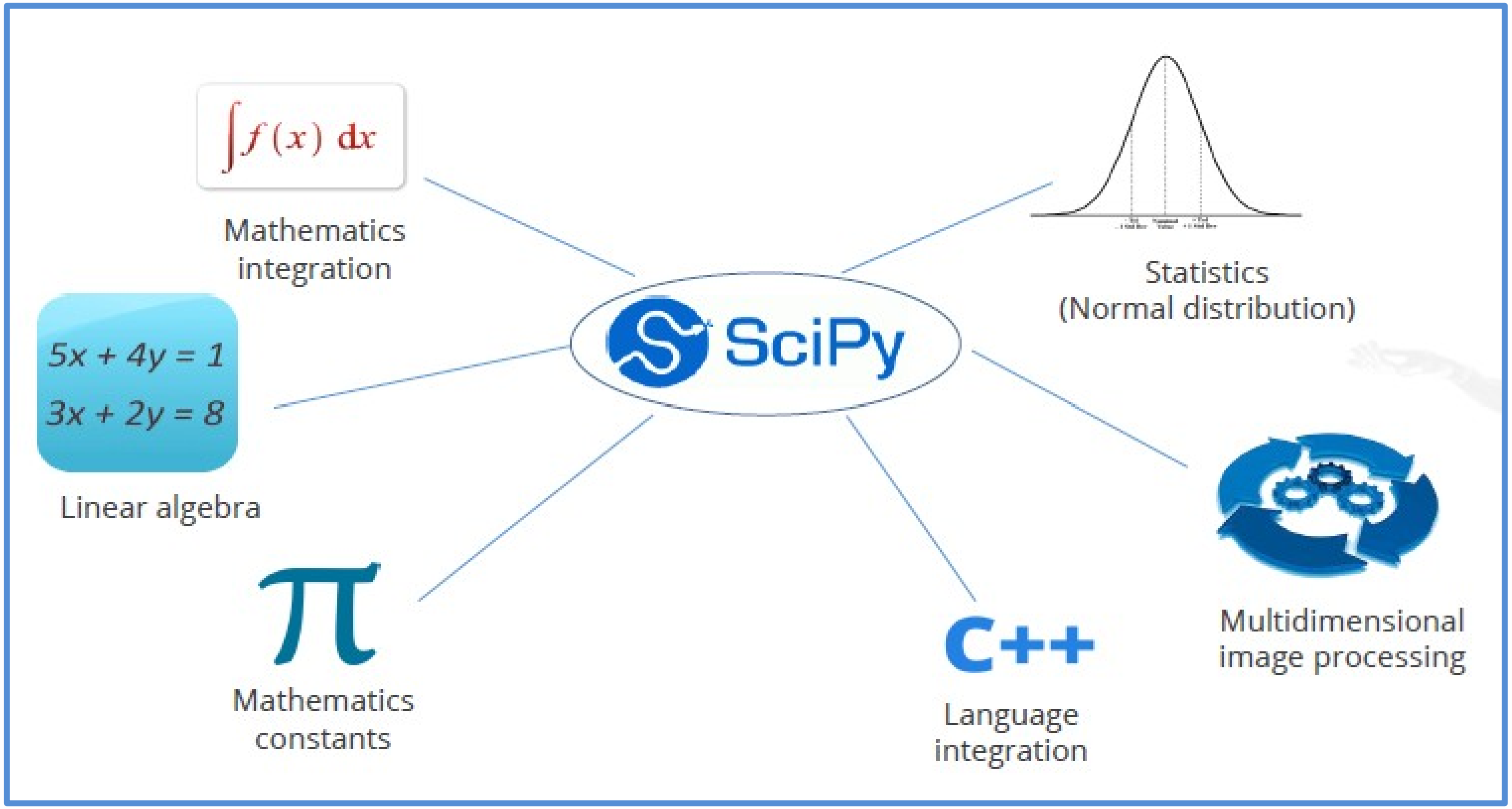


Image source : google.com

Characteristics of scipy

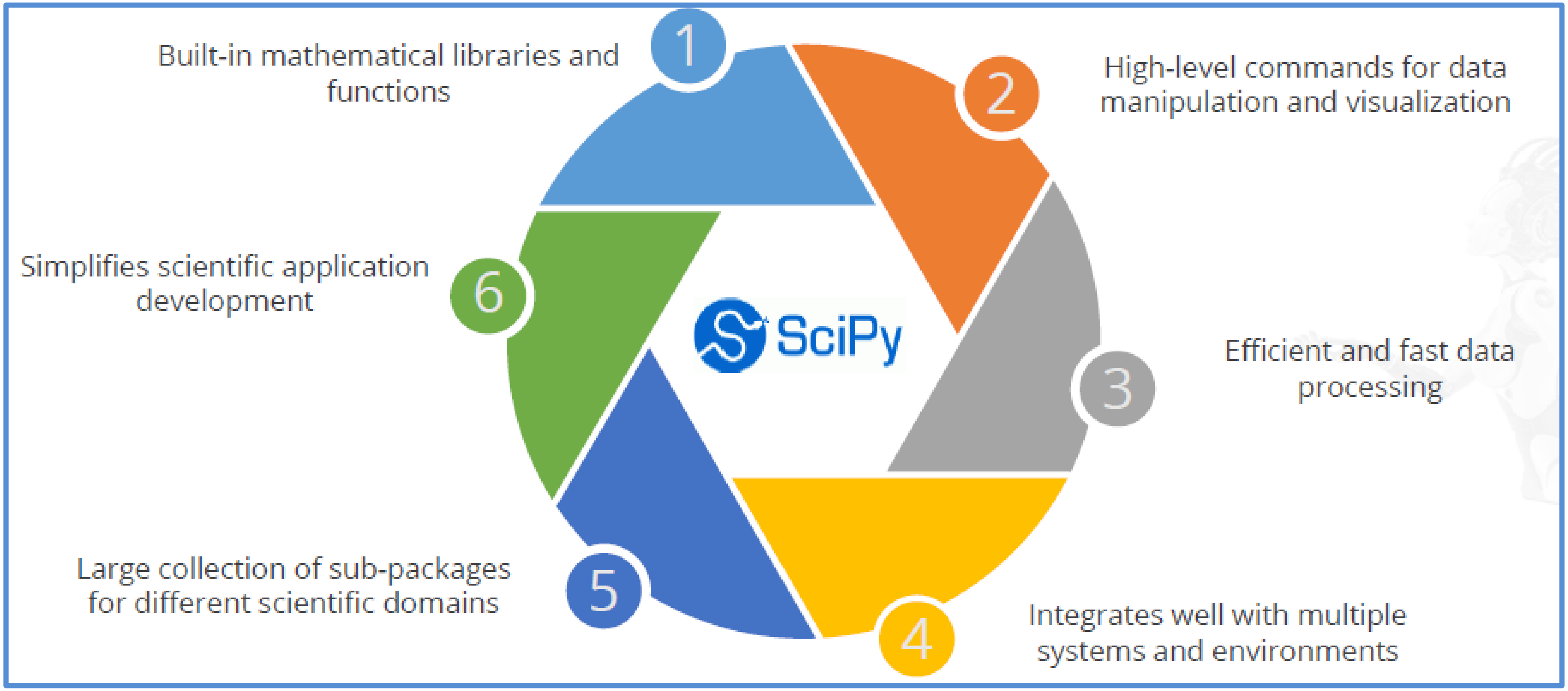
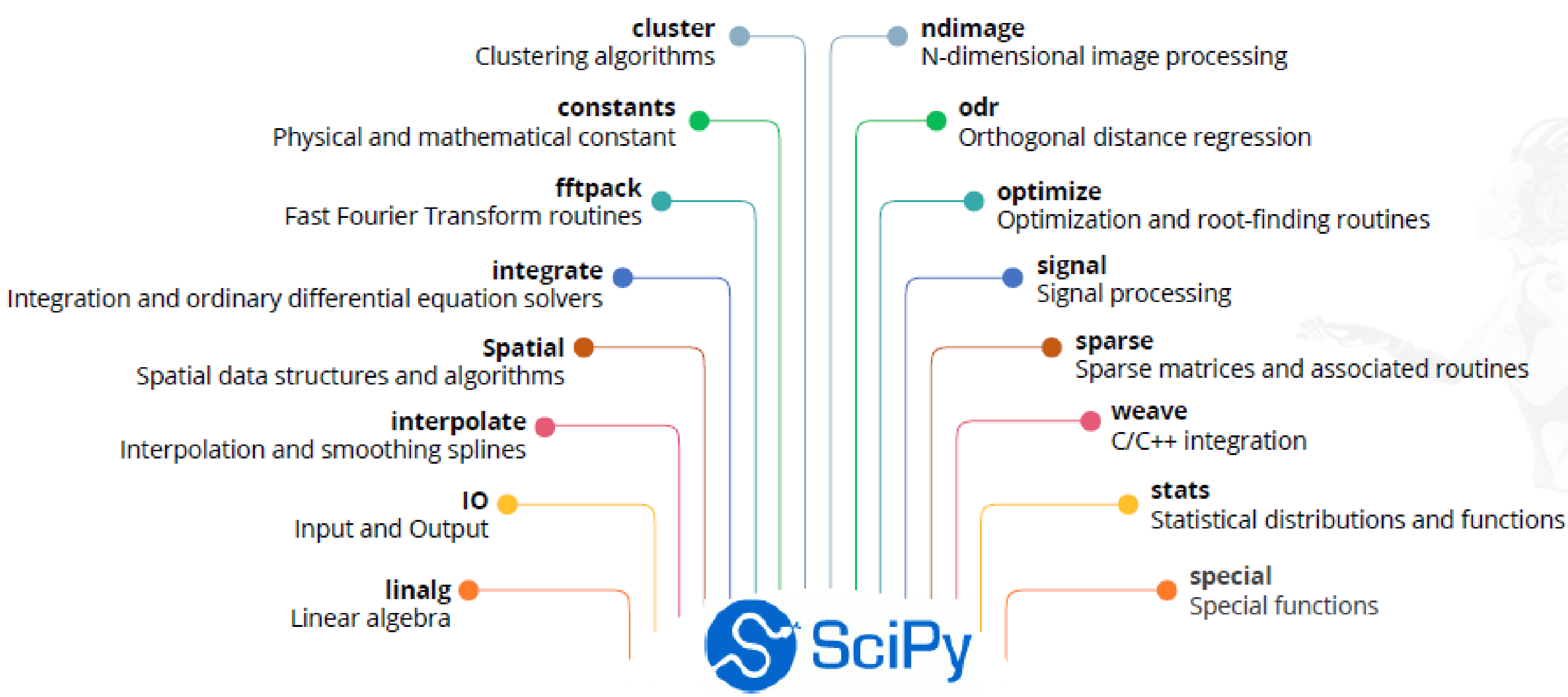


Image source : Simplilearn

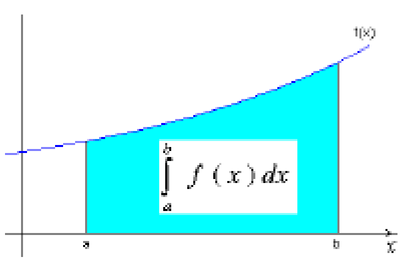
Sub-packages of scipy

SciPy has multiple sub-packages which handle different scientific domains.



Scipy subpackage – Integration

- The **scipy.integrate** sub-package provides a number of integration techniques including an ordinary differential equation integrator.
- Which is use to solve mathematical sequences and series, or perform function approximation.

General integration (quad)	General multiple integration (dblquad, tplquad, nquad)
integrate.quad(f, a, b) 	• integrate.dblquad() • integrate.tplquad() • integrate.nquad() The limits of all inner integrals need to be defined as functions.

Single Integrals

- For single integrals quad() function is use.

  Syntax : scipy.integrate.quad(f,a,b)

Where,

F is : function name for integration (user defined or inbuilt function)

A is : upper limit (numeric value)

B is : lower limit (numeric value)

- It provided to integrate a function of one variable between two points (upper limit and lower limit). The points can be **±∞(inf)** to indicate infinite limits. For example, suppose you wish to integrate a bessel function jv(2.5, x) along the interval [0,4.5] than.

$$\boxed{I = \int_0^{4.5} J_{2.5}(x)\, dx.}$$

Cont...

```
In [2]: import scipy.integrate as integrate
        import scipy.special as special
        #here we are creatig lambda function
        result = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)
        print("value of rasult is : ",result)

        value of rasult is :  (1.1178179380783244, 7.866317216380707e-09)

In [7]: from scipy.integrate import quad
        def integrand(x, a, b):  #UDF function for ax2 + b
            return a*x**2 + b
        a = 2
        b = 1
        ans = quad(integrand, 0, 1, args=(a,b))
        print("answer is : ",ans)

        answer is :  (1.6666666666666667, 1.8503717077085944e-14)
```

Cont...

```
In [13]: from scipy.integrate import quad
```

Import quad from integrate sub-package

```
In [14]: def integrateFunction(x):
             return x
```

Define function for integration of x

```
In [15]: quad(integrateFunction,0,1)
Out[15]: (0.5, 5.551115123125783e-15)
```

Perform quad integration for function of x for limit 0 to 1

```
In [16]: def integrateFn(x,a,b):
             return x*a+b
```

Define function for ax + b

```
In [17]: a=3
         b=2
```

Declare value of a and b

```
In [18]: quad(integrateFn,0,1,args=(a,b))
Out[18]: (3.5, 3.885780586188048e-14)
```

Perform quad integration and pass functions and arguments

Multiple integration

- Is use to find double and triple integration.
- Which is rapped within the **dblquad, tplquad and nquad** function.
- Here **dblquad()** is use for double integration.
- **tplquad()** is use for triple integration
- **nquad()** use for n integration.
- For double integration
- **Syntax : scipy.integrate.dblquad(func,a,b,gfun,hfun)**
- **Where**
- **Func** : name of function
- **a and b** are upper and lower limit
- **gfun and hfun** are names of the functions that define the lower and upper limit of the y variable

Cont...

In [8]:
```
#double integration using dblquad() function
import scipy.integrate
from numpy import exp
from math import sqrt
f = lambda x, y : 16*x*y
g = lambda x : 0
h = lambda y : sqrt(1-4*y**2)
i = scipy.integrate.dblquad(f, 0, 0.5, g, h)
print(i)
```

```
(0.5, 1.7092350012594845e-14)
```

In [9]:
```
from scipy.integrate import dblquad
area = dblquad(lambda x, y: x*y, 0, 0.5, lambda x: 0, lambda x: 1-2*x)
print(area)
```

```
(0.010416666666666668, 4.101620128472366e-16)
```

$$I = \int_{y=0}^{1/2} \int_{x=0}^{1-2y} xy \, dx \, dy = \frac{1}{96}.$$

Cont...

```
In [20]: import scipy.integrate as integrate
```

Import integrate package sub-package

```
In [21]: def f(x, y):
             return x + y
         integrate.dblquad(f, 0, 1,lambda x: 0, lambda x: 2)

Out[21]: (3.0, 3.3306690738754696e-14)
```

Define function for x + y

Perform multiple integration using the lambda built-in function

Scipy package optimization

- Is use to optimize or improve the performance of system mathematically by using some optimize algorithms.
- For optimizing scipy provides the list of algorithms like bfgs, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA, or SLSQP and many more.
- Is use for minimization of curve fitting, multidimensional or scalar and root fitting.
- **For more please refer**

https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html
https://www.javatpoint.com/scipy-optimize
https://www.tutorialspoint.com/scipy/scipy_optimize.htm

Nelder- Mead Simplex Algorithm

```
In [10]: #Nelder- Mead Simplex Algorithm
         #pvoide minimum() function
         #used for minimization of scalar function of one or more variabl
         import numpy as np
         import scipy
         from scipy.optimize import minimize
         #define function f(x)
         def f(x):
             return .2*(1 - x[0])**2
         scipy.optimize.minimize(f, [2, -1], method="Nelder-Mead")
```

```
Out[10]:  final_simplex: (array([[ 1.        , -1.27109375],
                [ 1.        , -1.27118835],
                [ 1.        , -1.27113762]]), array([0., 0., 0.]))
                    fun: 0.0
                message: 'Optimization terminated successfully.'
                   nfev: 147
                    nit: 69
                 status: 0
                success: True
                      x: array([ 1.        , -1.27109375])
```

Cont..

```
In [32]: import numpy as np
         from scipy import optimize
```

Import numpy and optimize from SciPy

```
In [33]: def f(x):
             return x**2 + 5*np.sin(x)
```

Define function for X^2 + 5 sin x

```
In [34]: minimaValue = optimize.minimize(f,x0=2,method='bfgs',options={'disp':True})

Optimization terminated successfully.
         Current function value: -3.246394
         Iterations: 4
         Function evaluations: 24
         Gradient evaluations: 8
```

Perform optimize minimize function using bfgs method and options

```
In [35]: minimaValueWithoutOpt = optimize.minimize(f,x0=2,method='bfgs')

In [36]: minimaValueWithoutOpt

Out[36]:      fun: -3.2463942726915382
         hess_inv: array([[ 0.15430551]])
              jac: array([ -8.94069672e-08])
          message: 'Optimization terminated successfully.'
             nfev: 24
              nit: 4
             njev: 8
           status: 0
          success: True
                x: array([-1.11051051])
```

Perform optimize minimize function using bfgs method and without options

Root finding

```
In [11]: #root() function is used to find the root of the nonlinear equation
         #various methods such as hybr (the default) and
         #the Levenberg-Marquardt method from the MINPACK
         import numpy as np
         from scipy.optimize import root
         def func(x):
             return x*2 +  3* np.cos(x)
         a = root(func, 0.3)
         print(a)

    fjac: array([[-1.]])
     fun: array([2.22044605e-16])
 message: 'The solution converged.'
    nfev: 10
     qtf: array([-8.13081602e-10])
       r: array([-4.37742668])
  status: 1
 success: True
       x: array([-0.91485648])
```

Cont...

```
In [118]: import numpy as np
          from scipy.optimize import root
          def rootfunc(x):
              return x + 3.5 * np.cos(x)
```

Define function for X + 3.5 Cos x

```
In [119]: rootValue = root(rootfunc, 0.3)
```

Pass x value in argument for root

```
In [120]: rootValue
Out[120]:     fjac: array([[-1.]])
               fun: array([  2.22044605e-16])
           message: 'The solution converged.'
              nfev: 14
               qtf: array([ -8.32889313e-13])
                 r: array([-4.28198145])
            status: 1
           success: True
                 x: array([-1.21597614])
```

Function value and array values

SciPy : Linear Algebra

- As scipy is handy package to perfrom datascience based computation it provide lot many functions to perform mathematical calculation.
- Scipy linear algebra provide one more package named as **scipy.linalg** for machine learning.
- It have advance algebraic function which working with machine learning concept.
- The **ATLAS LAPACK and BLAS** library, provides very fast linear algebra capabilities. Linear algebra accepts 2D array object and Gives output in 2D array object.
- Linear algebra solve given equation using the **solve()** method. Which gives result in the form of 2D array.
- The algebraic equation are in the following format
- **ax+by = c for the x and y are unknown value**

Solving linear equation

```
In [13]: #Lets solve below given lenear equation
         #1. x + 3y +10z = 10, 2x + 12y + 7z = 18, 5x + 8y + 8z = 30
         #Let's create array for these equation
         import numpy as np
         from scipy import linalg
         coeff_arr = np.array([[1, 3, 10], [2, 12, 7], [5, 8, 8]])
         sol_arr= np.array([[10], [18], [30]])
         # Solve the linear algebra
         ans = linalg.solve(coeff_arr, sol_arr)
         print("result is : ",ans)
         # Checking Results
         print("\n Checking results, Vectors must be zeros")
         print(coeff_arr.dot(ans)-sol_arr)

         result is :  [[4.55393586]
          [0.51311953]
          [0.39067055]]

          Checking results, Vectors must be zeros
         [[1.77635684e-15]
          [0.00000000e+00]
          [0.00000000e+00]]
```

One more example

Linear equations

$$2x + 3y + z = 21$$
$$-x + 5y + 4z = 9$$
$$3x + 2y + 9z = 6$$

```
In [88]: import numpy as np
         from scipy import linalg
```

Import linalg

```
In [89]: numArray = np.array([[2,3,1],[-1,5,4],[3,2,9]])

In [90]: numArrValue = np.array([21,9,6])

In [91]: linalg.solve(numArray,numArrValue)

Out[91]: array([ 4.95,  4.35, -1.95])
```

Use solve method

Finding determinate value

- The determinate value is calculate for the square matrix.
- Determinate value help to find inverse of matrix.
- It represent using |A| where A is my square matrix.
- Determinate calculate on the diagonals value of square matrix.
- For more about determinate : https://www.mathsisfun.com/algebra/matrix-determinant.html

For a 2×2 matrix (2 rows and 2 columns):

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The determinant is:

$$|A| = ad - bc$$

"The determinant of A equals a times d minus b times c"

Image Source : https://www.mathsisfun.com

Cont...

For a 3×3 matrix (3 rows and 3 columns):

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

The determinant is:

$$|A| = a(ei - fh) - b(di - fg) + c(dh - eg)$$

"The determinant of A equals ... etc"

It may look complicated, but **there is a pattern**:

$$a \times \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \times \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \times \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

Image Source : https://www.mathsisfun.com

Finding determinate for 2 X 2 matrix

```
In [15]: #for finding determinate linalg.det() function is use
         from scipy import linalg
         import numpy as np
         #Declaring the numpy array
         A = np.array([[5,9],[8,4]])
         print(A)
         #Passing the values to the det function
         x = linalg.det(A)
         #printing the result
         print("determinate of given matrix is :", x)

         [[5 9]
          [8 4]]
         determinate of given matrix is : -52.0
```

Finding Determinate for 3 X 3 matrix

```
In [16]: #for finding determinate linalg.det() function is use
         from scipy import linalg
         import numpy as np
         #Declaring the numpy array
         A = np.array([[10,5,8],[8,4,11],[12,4,6]])
         print(A)
         #Passing the values to the det function
         x = linalg.det(A)
         #printing the result
         print("determinate of given matrix is :", x)

         [[10  5  8]
          [ 8  4 11]
          [12  4  6]]
         determinate of given matrix is : 92.0
```

Like wise you can find for 4 X 4 for any n X n matrix where n is any integer number

Inverse of matrix

- Inverse of matrix represent with the reciprocal of number. If 8 is one number than reciprocal of 8 is 1/8 and it represent 8^{-1}.
- The Inverse of a Matrix is the same idea but we write it A^{-1}.
- Why inverse of matrix is required? because we can not divide the matrix but when we multiply actual matrix with it inverse matrix than we get identity matrix.

2x2 Matrix

OK, how do we calculate the inverse?

Well, for a 2x2 matrix the inverse is:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\underbrace{ad-bc}_{\text{determinant}}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Image Source : https://www.mathsisfun.com

Inverse of 2 x 2 matrix

```
In [22]: #inverse of matrix
         from scipy import linalg
         import numpy as np
         #Declaring the numpy array
         A = np.array([[5,9],[8,4]])
         print(A)
         inv_mat = linalg.inv(A)
         print("inverse of matrix is : ",inv_mat)
         print("checking the answer, mutiplication should be identity")
         ans = A * inv_mat
         ans = ans.astype(int)
         print(ans)

[[5 9]
 [8 4]]
inverse of matrix is :  [[-0.07692308  0.17307692]
 [ 0.15384615 -0.09615385]]
checking the answer, mutiplication should be identity
[[0 1]
 [1 0]]
```

Inverse of 3 X 3 matrix

```
In [23]: A = np.array([[1,5,9],[8,4,10],[12,4,5]])
         print(A)
         inv_mat = linalg.inv(A)
         print("inverse of matrix is : ",inv_mat)
         print("checking the answer, mutiplication should be identity")
         ans = A * inv_mat
         ans = ans.astype(int)
         print(ans)

[[ 1  5  9]
 [ 8  4 10]
 [12  4  5]]
inverse of matrix is :  [[-0.08474576  0.04661017  0.05932203]
 [ 0.33898305 -0.43644068  0.26271186]
 [-0.06779661  0.23728814 -0.15254237]]
checking the answer, mutiplication should be identity
[[ 0  0  0]
 [ 2 -1  2]
 [ 0  0  0]]
```

Singular-Value Decomposition (SVD)

- SVD Is use for matrix decomposition mean reduce the matrix.
- **A = U . Sigma . V^T**
- Where **A is the real m x n** matrix that we wants to decompose,
- **U** is an m x m matrix,
- **Sigma** (often represented by Greek letter Sigma) is an m x n diagonal matrix, **V^T** is the transpose of an n x n matrix where T is a superscript.

Image Source : https://www.mathsisfun.com

Cont...

```
In [25]: from numpy import array
         from scipy.linalg import svd
         # define a matrix
         A = array([[1, 2], [3, 4], [5, 6]])
         # SVD
         U, s, VT = svd(A)
         print("Unitery Matrix")
         print(U)
         print("Sigma / Square root of Eigenvale")
         print(s)
         print("VH value ")
         print(VT)

         Unitery Matrix
         [[-0.2298477   0.88346102  0.40824829]
          [-0.52474482  0.24078249 -0.81649658]
          [-0.81964194 -0.40189603  0.40824829]]
         Sigma / Square root of Eigenvale
         [9.52551809 0.51430058]
         VH value
         [[-0.61962948 -0.78489445]
          [-0.78489445  0.61962948]]
```

Cont...

```
In [103]: import numpy as np
          from scipy import linalg
```

Import linalg

```
In [104]: numSvdArr = np.array([[3,5,1],[9,5,7]])
```

Define matrix

```
In [105]: numSvdArr.shape
Out[105]: (2L, 3L)
```

Find shape of ndarray which is 2X3 matrix

```
In [106]: linalg.svd(numSvdArr)
```

Use svd function

```
Out[106]: (array([[-0.37879831, -0.92547925],
                  [-0.92547925,  0.37879831]]),
           array([ 13.38464336,   3.29413449]),
           array([[-0.7072066 , -0.4872291 , -0.51231496],
                  [ 0.19208294, -0.82977932,  0.52399467],
                  [-0.68041382,  0.27216553,  0.68041382]]))
```

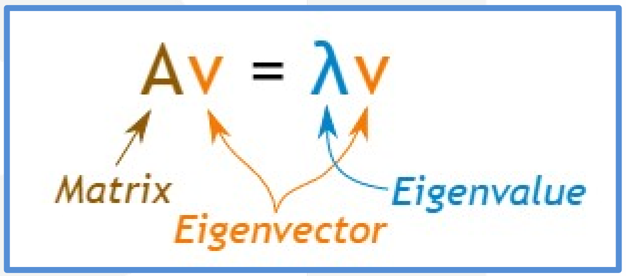
U (Unitary matrix)

Sigma or square root of eigenvalues

VH is values collected into unitary matrix

Calculate Eigenvalues and Eigenvector

- Finding Eigenvalue and Eigenvector are the most common problem for any linear algebraic equation.
- It is calculated for square matrix i.e. (2X2 , 3X3 or any square matrix)
- Is calculated using :



- Where A is any square matrix
- V is any n X 1 vector for real numbers
- λ is a scalar (which may be either real or complex)
- For more :
  https://lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html
  https://www.mathsisfun.com/algebra/eigenvalue.html

How to find Eigenvalue for the given matrix

Example: Solve for λ:

Start with $| A - \lambda I | = 0$

$$\left| \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0$$

Which is:

$$\begin{vmatrix} -6-\lambda & 3 \\ 4 & 5-\lambda \end{vmatrix} = 0$$

Calculating that determinant gets:

$$(-6-\lambda)(5-\lambda) - 3\times4 = 0$$

Which then gets us this Quadratic Equation:

$$\lambda^2 + \lambda - 42 = 0$$

And solving it gets:

$$\lambda = -7 \text{ or } 6$$

And yes, there are **two** possible eigenvalues.

Find eigenvector for eigenvalues

Example (continued): Find the Eigenvector for the Eigenvalue **λ = 6**:

Start with:

$$Av = \lambda v$$

Put in the values we know:

$$\begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = 6\begin{bmatrix} x \\ y \end{bmatrix}$$

After multiplying we get these two equations:

$$\begin{aligned} -6x + 3y &= 6x \\ 4x + 5y &= 6y \end{aligned}$$

Cont...

Bringing all to left hand side:

$$\begin{aligned} -12x + 3y &= 0 \\ 4x - 1y &= 0 \end{aligned}$$

Either equation reveals that **y = 4x**, so the **eigenvector** is any non-zero multiple of this:

$$\begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

And we get the solution shown at the top of the page:

$$\begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} -6\times1+3\times4 \\ 4\times1+5\times4 \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix}$$

... and also ...

$$6 \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix}$$

So $Av = \lambda v$

Let’s find it using python scipy

In [1]:
```
#importing the scipy and numpy packages
from scipy import linalg
import numpy as np
#Declaring the numpy array
a = np.array([[3,2],[4,6]])
#Passing the values to the eig function
l, v = linalg.eig(a)
#printing the result for eigenvalues
print(l)
#printing the result for eigenvectors
print(v)
```

```
[1.29843788+0.j 7.70156212+0.j]
[[-0.76164568 -0.39144501]
 [ 0.64799372 -0.9202015 ]]
```

SciPy Sub Package - Statistics

- As for data science and data analytics statistics plays very important role so sicpy have rich set of function to perform calculation based on statistics.
- All the function are located into **scipy.stats** package.
- A list of random variables available can also be obtained from the **docstring** for the stats sub-package.
- **We can get to know list of function supported by stats package using info(stats).**

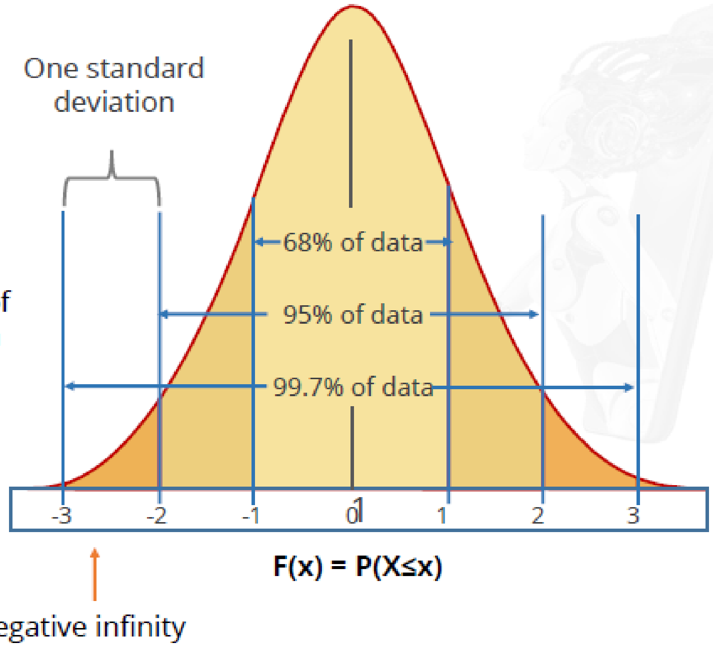
Function	Description
rv_continuos	It is a base class to construct specific distribution classes and instances for continuous random variable.
rv_discrete	It is a base class to construct specific distribution classes and instances for discrete random variables.
rv_histogram	It can be inherited from **rv_continuous** class. It generates a distribution given by a histogram.

Normal Continuous Random Variable

CDF or Cumulative Distribution Function provides the cumulative probability associated with a function.

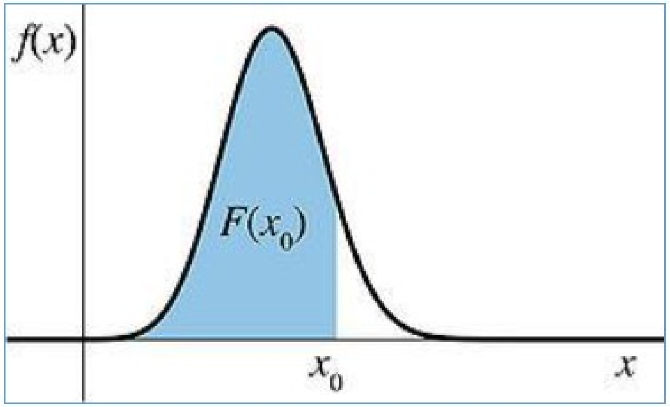
Age Range	Frequency	Cumulative Frequency
0-10	19	19
10-20	55	74
21-30	23	97
31-40	36	133
41-50	10	143
51-60	17	160

Total number of persons within this age



Cont...

Probability Density Function, or **PDF,** of a continuous random variable is the derivative of its Cumulative Distribution Function, or CDF.



$$f(x) = \frac{dF(x)}{dx}$$

Derivative of CDF

Cont...

```
In [1]: # two general distribution classes which have been implemented for encapsulating
        # is continuous random variables and discrete random variable
        # calculating Cumulative Distribution Function
        from scipy.stats import norm
        import numpy as np
        print(norm.cdf(np.array([3,-1., 0, 1, 2, 4, -2, 5])))

        [0.9986501  0.15865525 0.5        0.84134475 0.97724987 0.99996833
         0.02275013 0.99999971]

In [4]: #To get the median of the distribution, we can use the Percent Point Function (PPF),
        #PPF is the inverse of the CDF.
        from scipy.stats import norm
        print(norm.rvs(size = 4))
        #output for rvs is change with every execution
        #for getting fix value use seed() function

        [ 0.55359238 -1.55145107 -0.68957377 -0.0369672 ]
```

Cont...

Shown here are functions used to perform Normal Distribution:

```
In [108]: from scipy.stats import norm
```

Import norm for normal distribution

```
In [110]: norm.rvs(loc=0,scale=1,size=10)
```

rvs for Random variables

```
Out[110]: array([-0.16337774,  0.39039561,  0.85642826,  0.30134358, -1.86009474,
                 -0.29621603,  0.03863757,  0.23727056, -1.42395316, -0.5730162 ])
```

```
In [112]: norm.cdf(5,loc=1,scale=2)
```

cdf for Cumulative Distribution Function

```
Out[112]: 0.9772498680518208
```

```
In [113]: norm.pdf(9,loc=0,scale=1)
```

pdf for Probability Density Function for random distribution

```
Out[113]: 1.0279773571668917e-18
```

loc and **scale** are used to adjust the location and scale of the data distribution.

SciPy Sub-Package: Weave and IO

- Scipy have many module, classes and function to read and write operation for file and data from various source.
- **But scipy.io package** provide wide range of function to for working on file read write operation. It support many file formats like below.
- **Idl, Matlab, Matrix Market, Arff, Wave, Netcdf, etc.**
- But for now we only concentrate on wave file type.
- For more you can visit : https://www.tutorialspoint.com/scipy/scipy_input_output.htm
- https://www.javatpoint.com/scipy-input-and-output

Cont...

The weave package provides ways to modify and extend any supported extension libraries.

Features of Weave Package:

- Includes C/C++ code within Python code
- Speed ups of 1.5x to 30x compared to algorithms written in pure Python

Two main functions of weave::

- inline() compiles and executes C/C++ code on the fly
- blitz() compiles NumPy Python expressions for fast execution

example

```
In [1]: import scipy.io as sio
        import numpy as np
        #Save a mat file
        vect = np.arange(10)
        sio.savemat('array.mat', {'vect':vect})
        #Now Load the File
        mat_file_content = sio.loadmat('array.mat')
        print(mat_file_content)

        {'__header__': b'MATLAB 5.0 MAT-file Platform: nt, Created on: Thu Dec 31 20:56:35 2020', '__version__': '1.0', '__globals__':
        [], 'vect': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])}

In [2]: #array consisted with the information. If we want to inspect the contents of a MATLAB file without
        #reading the data into memory
        #use the whosmat command
        import scipy.io as sio
        mat_file_content = sio.whosmat('array.mat')
        print(mat_file_content)

        [('vect', (1, 10), 'int32')]
```

Parul® University

DIGITAL LEARNING CONTENT

For more about scipy

- https://www.tutorialspoint.com/scipy/index.htm
- https://docs.scipy.org/doc/scipy/reference/tutorial/
- https://www.guru99.com/scipy-tutorial.html#4
- https://scipy-lectures.org/
- https://www.javatpoint.com/scipy-installation
- https://www.journaldev.com/18106/python-scipy-tutorial
- https://www.w3schools.com/python/scipy_intro.asp

DIGITAL LEARNING CONTENT