Why use SciPy

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

Package Name	Description
scipy.io	• File input/output
scipy.special	• Special Function
scipy.linalg	• Linear Algebra Operation
scipy.interpolate	• Interpolation
scipy.optimize	• Optimization and fit
scipy.stats	• Statistics and random numbers
scipy.integrate	• Numerical Integration
scipy.fftpack	• Fast Fourier transforms
scipy.signal	• Signal Processing
scipy.ndimage	• Image manipulation –

Numpy VS SciPy

Numpy:

- Numpy is written in C and use for mathematical or numeric calculation.
- It is faster than other Python Libraries
- Numpy is the most useful library for Data Science to perform basic calculations.
- Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

SciPy:

- SciPy is built in top of the NumPy
- SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
- Most new Data Science features are available in Scipy rather than Numpy.

SciPy - Installation and Environment Setup

You can also install SciPy in Windows via pip

```
Python3 -m pip install --user numpy scipy
```

Before start to learning SciPy, you need to know basic functionality as well as different types of an array of NumPy

The standard way of import SciPy modules and Numpy:

```
from scipy import special   #same for other modules
import numpy as np
```

Special Function package

- **scipy.special** package contains numerous functions of mathematical physics.
- SciPy special function includes Cubic Root, Exponential, Log sum Exponential, Lambert, Permutation and Combinations, Gamma, Bessel, hypergeometric, Kelvin, beta, parabolic cylinder, Relative Error Exponential, etc..
- For one line description all of these function, type in Python console:

```
help(scipy.special)
Output :
NAME
    scipy.special

DESCRIPTION
    ========================================
    Special functions (:mod:`scipy.special`)
    ========================================

    .. module:: scipy.special

    Nearly all of the functions below are universal functions and follow
    broadcasting and automatic array-looping rules. Exceptions are noted.
```

Cubic Root Function:

Cubic Root function finds the cube root of values.

Syntax:

```
scipy.special.cbrt(x)
```

Example:

```
from scipy.special import cbrt
#Find cubic root of 27 & 64 using cbrt() function
cb = cbrt([27, 64])
#print value of cb
print(cb)
```

Output: array([3., 4.])

Exponential Function:

Exponential function computes the 10**x element-wise.

Example:

```
from scipy.special import exp10
#define exp10 function and pass value in its
exp = exp10([1,10])
print(exp)
```

Output: [1.e+01 1.e+10]

Permutations & Combinations:

SciPy also gives functionality to calculate Permutations and Combinations.

Combinations - scipy.special.comb(N,k**)**

Example:

```
from scipy.special import comb
#find combinations of 5, 2 values using comb(N, k)
com = comb(5, 2, exact = False, repetition=True)
print(com)
```

Output: 15.0

Permutations –

```
scipy.special.perm(N,k)
```

Example:

```
from scipy.special import perm
#find permutation of 5, 2 using perm (N, k) function
per = perm(5, 2, exact = True)
print(per)
```

Output: 20

Linear Algebra with SciPy

- Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
- Performance of Linear Algebra is very fast compared to BLAS and LAPACK.
- Linear algebra routine accepts two-dimensional array object and output is also a two-dimensional array.

Now let's do some test with **scipy.linalg,**

Calculating **determinant** of a two-dimensional matrix,

```
from scipy import linalg
import numpy as np
#define square matrix
two_d_array = np.array([ [14,15], [13,12] ])
#pass values to det() function
linalg.det( two_d_array )
```

Inverse Matrix –

```
scipy.linalg.inv()
```

Inverse Matrix of Scipy calculates the inverse of any square matrix.

```
from scipy import linalg
import numpy as np
# define square matrix
two_d_array = np.array([ [14,15], [13,12] ])
#pass value to function inv()
linalg.inv( two_d_array )
```

Eigenvalues and Eigenvector – scipy.linalg.eig()

- The most common problem in linear algebra is eigenvalues and eigenvector which can be easily solved using **eig()** function.
- Now lets we find the Eigenvalue of (**X**) and correspond eigenvector of a two-dimensional square matrix.

Example,

```
from scipy import linalg
import numpy as np
#define two dimensional array
arr = np.array([[15,14],[16,13]])
#pass value into function
eg_val, eg_vect = linalg.eig(arr)
#get eigenvalues
print(eg_val)
#get eigenvectors
print(eg_vect)
```

.

Integration with Scipy – Numerical Integration

- When we integrate any function where analytically integrate is not possible, we need to turn for numerical integration
- SciPy provides functionality to integrate function with numerical integration.
- **scipy.integrate** library has single integration, double, triple, multiple, Gaussian quadrate, Romberg, Trapezoidal and Simpson's rules.

Example: Now take an example of **Single Integration**

$$\int_a^b f(x)dx$$

Here **a** is the upper limit and **b** is the lower limit

```
from scipy import integrate
# take f(x) function as f
f = lambda x : x**2
#single integration with a = 0 & b = 1
integration = integrate.quad(f, 0 , 1)
print(integration)
```

Output:

(0.33333333333333337, 3.700743415417189e-15)

Here function returns two values, in which the first value is integration and second value is estimated error in integral.

Example: Now take an example of **double integration.** We find the double integration of the following equation,

$$\int_0^{2/4} \cdot \int_0^{\sqrt{1-2y^2}} 64xy\, dx$$

```
from scipy import integrate
import numpy as np
#import square root function from math lib
from math import sqrt
# set  fuction f(x)
f = lambda x, y : 64 *x*y
```

```
# lower limit of second integral
p = lambda x : 0
# upper limit of first integral
q = lambda y : sqrt(1 - 2*y**2)
# perform double integration
integration = integrate.dblquad(f , 0 , 2/4,  p, q)
print(integration)
```

Output:

(3.0, 9.657432734515774e-14)

You have seen that above output as same previous one.

Summary

- SciPy(pronounced as "Sigh Pi") is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing.
- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy is built in top of the NumPy