Archanavyas
Apr 20 · 9 min read · Listen

Mathematics deals with a huge number of concepts that are very important but at the same time, complex and time-consuming. However, Python provides the full-fledged SciPy library that resolves this issue for us. In this SciPy tutorial, you will be learning how to make use of this library along with a few functions and their examples.

So let's get started. :)

SciPy — Python Library

SciPy

SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems, such as MATLAB, IDL, Octave, R-Lab, and SciLab.

The additional benefit of basing SciPy on Python is that this also makes a powerful programming language available for use in developing sophisticated programs and

base subroutines and classes have been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy.

Comparison Between SciPy and Numpy

Point of Difference	NumPy	SciPy
Type of operations	Performs basic operations such as sorting, indexing, etc. It is mostly used when working with data science and statistical concepts.	Used for complex operations such as algebraic functions, various numerical algorithms, etc.
Functions	Contains a variety of functions but these are not defined in depth.	Contains detailed versions of the functions like linear algebra that are completely featured.
Arrays	NumPy Arrays are multi-dimensional arrays of objects which are of the same type i.e. homogeneous.	SciPy does not have any such array concepts as it is more functional. It has no constraints of homogeneity.
Base Language of creation and speed	NumPy is written in **C** and so has a faster computational speed.	SciPy is written in **Python** and so has a slower execution speed but vast functionality.

Comparison Between SciPy and NumPy

Let's get started by installing SciPy on your Machine.

Installing SciPy

Before learning more about the core functionality of SciPy, it should be installed in the system.

Install on Windows and Linux

Here are a few methods that can be used to install SciPy on Windows or Linux.

Install SciPy using pip

We can install the SciPy library by using the pip command. Pip is basically a recursive acronym which stands for 'Pip Installs Packages'. It is a standard package manager which can be installed in most of the operating systems. To install, run the following




Note: Use pip to install SciPy in Linux.

Install SciPy using Anaconda

We can also install SciPy packages by using Anaconda. First, we need to download the Anaconda navigator and then open the anaconda prompt type the following command:

```
conda install -c anaconda scipy
```

Install on Mac

The mac doesn't have the preinstall package manager, but you can install various popular package managers. Run the following commands in the terminal it will download the SciPy as well as matplotlib, pandas, NumPy.

```
sudo port install py35-numpy py35-scipy py35-matplotlib py35-ipython
+notebook py35-pandas py35-sympy py35-nose
```

Also, you can use Homebrew to install these packages. But keep in mind that it has incomplete coverage of the SciPy ecosystem:

```
brew install numpy scipy ipython jupyter
```

Sub packages in SciPy

Many dedicated software tools are necessary for Python scientific computing, and SciPy is one such tool or library offering many Python modules that we can work with in order to perform complex operations.




cluster Clustering algorithms

constants Physical and mathematical constants

fftpack Fast Fourier Transform routines

integrate Integration and ordinary differential equation solvers

interpolate Interpolation and smoothing splines

io Input and Output

linalg Linear algebra

ndimage N-dimensional image processing

odr Orthogonal distance regression

optimize Optimization and root-finding routines

signal Signal processing

sparse Sparse matrices and associated routines

spatial Spatial data structures and algorithms

special Special functions

stats Statistical distributions and functions

However, for a detailed description, you can follow the official documentation. By the end of this tutorial, you will learn how to use scipy to do the most common mathematical and scientific computations using some of the above functions/sub-packages.

These packages need to be imported exclusively prior to using them. For example:




Because of their ubiquitousness, some of the functions in these subpackages are also made available in the `scipy` namespace to ease their use in interactive sessions and programs. In addition, many basic array functions from `numpy` are also available at the top-level of the `scipy` package.

Finding Documentation

SciPy and NumPy have documentation versions in both HTML and PDF format available at https://docs.scipy.org/, which cover nearly all available functionality. However, this documentation is still work-in-progress and some parts may be incomplete or sparse. As we are a volunteer organization and depend on the community for growth, your participation — everything from providing feedback to improving the documentation and code — is welcome and actively encouraged.

Python's documentation strings are used in SciPy for online documentation. There are two methods for reading them and getting help. One is Python's command `help` in the `pydoc` module. Entering this command with no arguments (i.e. `>>> help`) launches an interactive help session that allows searching through the keywords and modules available to all of Python. Secondly, running the command *help(obj)* with an object as the argument displays that object's calling signature, and documentation string.

Let's take a closer look at some of the SciPy functions.

Basic functions

help( )

- without any parameters
- using parameters

Here is an example that shows both of the above methods:

```
from scipy import cluster
help(cluster)      #with parameter
help()             #without parameter
```

When you execute the above code, the first help() returns the information about the *cluster* submodule. The second help() asks the user to enter the name of any module, keyword, etc for which the user desires to seek information. To stop the execution of this function, simply type 'quit' and hit enter.

info( )

Returns information about desired function, modules etc.

```
import scipy
scipy.info(cluster)
```

Source( )

Returns the source code only for objects written in python. This function does not return useful information in case the methods or objects are written in any other language such as C. However in case you want to make use of this function, you can do it as follows:

```
scipy.source(cluster)
```

Special functions (`scipy.special`)

The main feature of the **`scipy.special`** the package is the definition of numerous




for general use as an easier interface to these functions is provided by the `stats` module. Most of these functions can take array arguments and return array results following the same broadcasting rules as other math functions in Numerical Python. Many of these functions also accept complex numbers as input. For a complete list of the available functions with a one-line description type `>>> help(special)`. Each function also has its own documentation accessible using help.

Exponential & Trigonometric functions

SciPy's Special Function package provides a number of functions through which you can find exponents and solve trigonometric problems.

For Example:

```
from scipy import special
a = special.exp2(3)
print(a)

b = special.exp10(4)
print(b)

c = special.sindg(90)
print(c)

d = special.cosdg(60)
print(d)

e= special.tandg(45)
print(e)
```

Output:

```
8.0
10000.0
1.0
0.49999999999999994
1.0
```

Bessel functions of real order(`iv , jn, zeros`)




$$x^2 \frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - \alpha^2)y = 0$$

Among other uses, these functions arise in wave propagation problems, such as the vibrational modes of a thin drum head. **Here is an example of a circular drum head anchored at the edge:**

```
# Bessel Function of first kind of real order and complex argument
c = special.jv(2,4)
print(c)
```

Output:

```
0.3641281458520728
```

There are many other functions present in the special functions package of SciPy that you can try for yourself.

Integration (`scipy.integrate`)

The **`scipy.integrate`** sub-package provides several integration techniques ranging from ordinary differential integrator to using trapezoidal rules to compute integrals, SciPy is a storehouse of functions to solve all types of integrals problems.

General Integration:

We can use the quad() function to find the single integration. The name comes from the fact that integration is sometimes called the quadrature.

The function **`quad`** is provided to integrate a function of one variable between two points. The limits can be ±∞ (± `inf`) to indicate infinite limits.

For example :

```
i = integrate.quad(lambda x:special.exp10(x),0,1)
print(i)
```

Output:

```
(3.9086503371292665, 4.3394735994897923e-14)
```

Double Integration:

We can use the dblquad() function to find the double integral of a function.A double integral, as many of us know, consists of two real variables. The dblquad() function will take the function to be integrated as its parameter along with 4 other variables which define the limits and the functions *dy* and *dx*.

For example :

```
# Double Integrate Function
from scipy import integrate
e = lambda x, y: x*y**2
f = lambda x: 1
g = lambda x: -1
integrate.dblquad(e,0,2,f,g)
```

Output:

```
(-0.0, 4.405142707569776e-14)
```

SciPy provides various other functions to evaluate triple integrals, n integrals, Romberg Integrals, etc that you can explore further in detail. To find all the details about the required functions, use the help function.

Linear Algebra

Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.

In addition to all the functions from numpy.linalg, scipy.linalg also provides a number of other advanced functions. Also, if numpy.linalg is not used along with ATLAS LAPACK and BLAS support, scipy.linalg is faster than numpy.linalg.

Finding the Inverse of a Matrix:

Mathematically, the inverse of a matrix A is the matrix B such that AB=I where I is the identity matrix consisting of ones down the main diagonal denoted as B=A-1. In SciPy, this inverse can be obtained using the *linalg.inv* method.

For Example:

```
# Linear Algebra
import numpy as np
from scipy import linalg
a = np.array([[1,2],[3,4]])
b = linalg.inv(a)
print(b)
```

Output:

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Finding the Determinants:

The value derived arithmetically from the coefficients of the matrix is known as the determinant of a square matrix. In SciPy, this can be done using a function *det*.

For Example:

```
# Linear Algebra
import numpy as np
from scipy import linalg
A = np.array([[1,2],[3,4]])
B = linalg.det(A)
print(B)
```




```
-2.0
```

Fourier Transform Functions

Fourier analysis is a method that deals with expressing a function as a sum of periodic components and recovering the signal from those components. The *fft* functions can be used to return the discrete Fourier transform of a real or complex sequence.

For Example:

```
from scipy.fftpack import fft, ifft
x = np.array([0,1,2,3])
y = fft(x)
print(y)
```

Output:

```
[ 6.-0.j -2.+2.j -2.-0.j -2.-2.j]
```

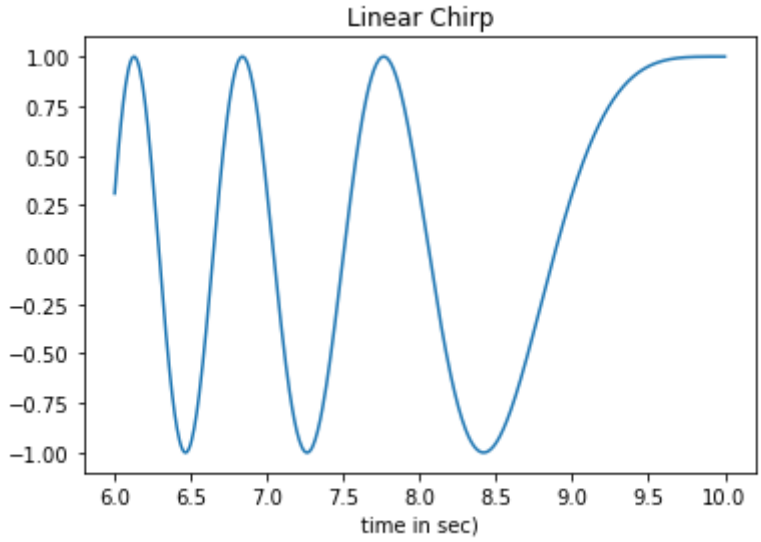
Waveforms

The scipy.signal subpackage also consists of various functions that can be used to generate waveforms. One such function is *chirp*. This function is a frequency-swept cosine generator

For Example:

```
from scipy.signal import chirp, spectrogram
import matplotlib.pyplot as plt
t = np.linspace(6, 10, 500)
w = chirp(t, f0=4, f1=2, t1=5, method='linear')
plt.plot(t, w)
plt.title("Linear Chirp")
plt.xlabel('time in sec)')
plt.show()
```



File IO

The scipy.io package provides a number of functions that help you manage files of different formats such as MATLAB files, IDL files, Matrix Market files, etc.

To make use of this package, you will need to import it as follows:

```
import scipy.io as sio
```

These were some SciPy library sub-packages. Follow the documentation to learn more about all of the features.

This SciPy tutorial has come to an end. I hope you have understood everything clearly.

Any kind of confusion in this tutorial? Please mention it in the comments section of this "SciPy Tutorial" blog and we will get back to you as soon as possible.

Presented by : Hetvi Soni, Khushi Tala & Archana Vyas

Charotar University of Science and Technology