SUMMER SCHOOL 2021

Scipy demo

Andrew Gill

Summary of commands used in this lecture

Command	Meaning
scipy.integrate.quad	a generally useful numerical integration function
scipy.integrate.trapz	trapezium-rule integration
scipy.fftpack.fft	Perform a Fast Fourier Transform on a numpy array of data (usually a time-series)
scipy.fftpack.ifft	Perform an inverse Fast Fourier Transform on a numpy array of frequency domain data

Introduction

Scipy and Numpy are supposed to be used together. Scipy can be thought of an addon for numpy to allow it to do still more specialised things.

There are a few sub-modules in Scipy which faithfully imitate Numpy, but attempt to do a better job behind the scenes, and add in a few extra features. Examples are the linalg and fft modules.

Scipy also has a number of other submodules that deal with tasks like numerical integration, interpolation, numerical optimization, [mathematical] graph manipulation, sparse matrix operations, advanced sound and image processing and advanced file-IO, to name a few.

Because these are used in diverse and specialized fields, it is neither easy nor sensible to attempt a thorough overview of Scipy's capabilities. Therefore, I will demonstrate a few of the more generally useful Scipy features, and leave it to you, the reader, to consult the excellent documentation to discover how to use those parts of Scipy which are relevant to your research.

Numerical integration with Scipy

Numerical integration is straightforward in scipy.
The quad (short for quadrature) function is shown here. This is a wrapper for the powerful Fortran Quadpack. There are other functions for different numerical integration methods, such as trapz for trapezium-rule integration.

```
import numpy as np
import scipy.integrate as integrate

def fn(x):
    return np.sin(x)-x*np.cos(x/2.0)-x**2

print(integrate.quad(fn,0,10))
```

Note that one needs to define a function that, when given an array x, returns an array of the values of the function of x. So in this case, I'm numerically calculating the integral of sin(x)-x*cos(x/2)-x^2

If you run the above script, you will see that Quad returns a tuple of two values. The first value is the result of the numerical integration. The second number, which is usually very small, is the estimated maximum error relative to the exact solution.

FFTs with Scipy

Fast Fourier Transforms are very useful for analysing cyclic or oscilating phenomena. Mathematically, Fourier Transforms are are similar to laplace transforms, except that the transform includes complex numbers in order to transform from the time into the frequency domain. FFT is an algorithm to do Fourier Transforms numerically in a way that is very computationally efficient (hence the "Fast"). They are extensively used in electronic engineering and scientific fields such as radio astronomy. They are also used in image compression (like the JPEG format).

Numpy and Scipy both include submodules for doing FFTs and inverse FFTs (which are transforming frequency domain data back to time-data).
When last I checked, there was some argument about which was actually better in practice. However, we will be using the fft submodule from scipy in this case.
Instead of reading in some data from a laboratory accelerometer or radio telescope as we would probably do in practise, we will generate some noisy data with cyclic components at different frequencies, then see if we can pick out these frequencies using an fft, knowing where they are supposed to be:

```
import numpy as np
from scipy.fftpack import fft
import matplotlib.pyplot as pl

samplerate=0.01
max_t=10

x = np.arange(0,max_t,samplerate)
y = np.sin(x*2*np.pi)+2.5*np.sin(x*3*2*np.pi)
+0.8*np.sin(x*7*2*np.pi)+ \
    0.5*np.sin(x*31*2*np.pi)+1.2*np.random.ranf(len(x))

freqdom=2*abs(fft(y,n=len(y)))/len(y)
freqrange=np.linspace(0,1/samplerate,len(x))

pl.subplot(2,1,1)
pl.plot(x,y,"b-")
pl.xlabel("Time (sec)")
pl.ylabel("Signal")
pl.subplot(2,1,2)
pl.plot(freqrange[:int(len(x)/2)],freqdom[:int(len(x)/2)],"k-")
pl.xlabel("Frequency (hz)")
pl.ylabel("Energy")
pl.savefig("fft_plots.png")
pl.show()
```

The actual fft is done in line 11, reproduced below for your convenience:

```
freqdom=2*abs(fft(y,n=len(y)))/len(y)
```

Note that some processing needs to be done on the output of the FFT in order to make it suitable for plotting. This tutorial is not intended to instruct the uninitiated in the mathematics of Fourier Transforms, but for those already initiated into the mysteries, it will be apparent that in this case, we are only looking at the amplitude, not the phase, which could be extracted by using the numpy.angle function (which determines the phase-angle of a complex number). Also note that the inverse fft is performed using the ifft function.

Going further

As mentioned earlier, this is a very brief demonstration of two parts of scipy. There is a more thorough online tutorial dealing with each submodule with examples here:

https://docs.scipy.org/doc/scipy/reference/tutorial/

Working through the examples for the section in which you are interested should be sufficient to get you started in using the methods that apply to your research.