# SciPy

Written by Dave Pawlowski, November 11, 2012

## 1 Introduction

The basic python programming language is great for a wide variety of tasks. However, in lacks some basic features that make it possible to effectively do intensive scientific computing. For example, it is not possible to do simple array or matrix mathematics using standard Python. Instead, many scientists and engineers use the SciPy package (http://www.scipy.org/) to gain access to common algorithms and functions inside Python. SciPy is not part of most standard Python distributions, and typically needs to be installed separately on a system. Any system that is meant for scientific computing, however, should have SciPy already installed.

## 2 Using Scipy

If SciPy is installed, you import it just as you would any other Python package:

```
>>> import scipy as sp
```

will import the scipy package, and allow you to refer to it using sp for short.

SciPy has several subpackages that need to be imported separately if you want to use them. Table 1 provides a list of available subpackages.

To gain access to the linear algebra package, you would execute:

```
>>> from scipy import linalg
```

## 3 The array class

SciPy has absolved another numerical Python packaged called Numpy, because the two packages were so commonly used together. When importing SciPy, all of the Numpy functions and classes become available to you, in the scipy namespace.

One of the most useful aspects of Scipy for scientific programming is the introduction of the array class of variables. SciPy makes this class available so that it can define functions that operate on

| **Subpackage** | **Description** |
|---|---|
| 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 |
| maxentropy | Maximum entropy methods |
| 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 |
| weave | C/C++ integration |

Table 1: Scipy subpackages

arrays, yet will not affect list objects. Creating an array is simple:

```
>>> arr = sp.array([1,2,5,4,6])
```

assuming that you imported SciPy under the sp namespace. In this example, we converted a list to an array. You could have just as easily created the list [1,2,5,4,6] first, and then called the array function to convert it. Unlike with a list, which can be populated with any combination of variable types, arrays all have to have the same variable type. This means that if you try to create an array such as:

```
>>> newarr = sp.array([1,'g',5,'d'])
```

Scipy will cause the numbers 1 and 5 to be converted to the strings '1' and '5', since it is not possible to convert the letters to ints.

The reason that it is nice to have the array object is that it is easy to create multidimensional arrays. For example, a common task for computer programmers is to create an array of zeros that will be populated at a later point in time. While this is possible using a list object, it is not trivial, and requires several lines of code. Using SciPy, it is simple:

```
>>> a = sp.zeros(100)
```

will create a 1-dimensional array filled with zeros. Its just as easy to make a 2-D array:

```
>>> a = sp.zeros((100,50))
```

Two sets of parenthesis are necessary because the functions zeros takes other optional arguments, such as data type. This lets Python know that the 100 and 50 are part of the first arguments, the shape of the array.

Scipy also allows you to easily create an array of ones:

```
>>> b = sp.ones((5,5,2))
```

which is a 3-D array with sides of length 5, 5, and 2 respectively. Indexing an array is done much the same way that indexing a list is performed. Figure 1 shows the anatomy of an array.

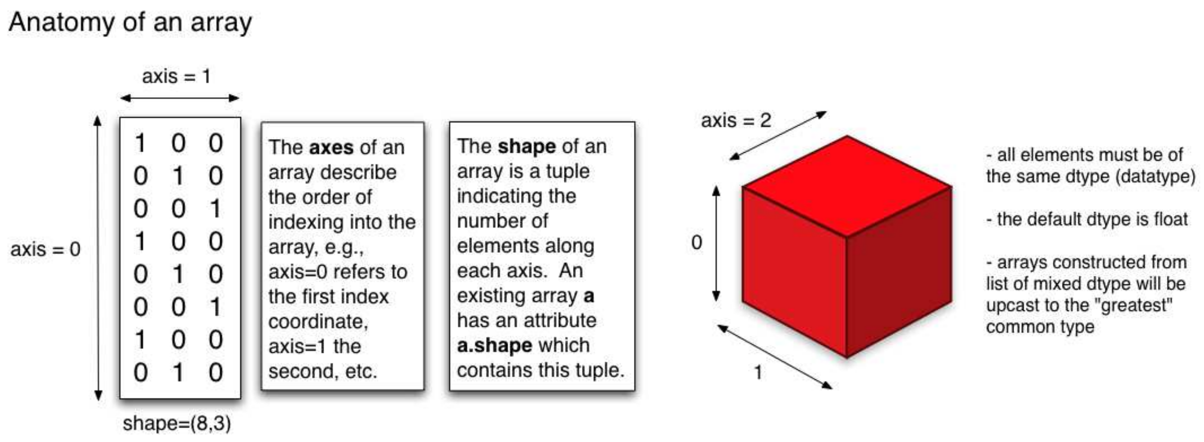
Anatomy of an array
axis = 1
axis = 0
1 0 0
0 1 0
0 0 1
1 0 0
0 1 0
0 0 1
1 0 0
0 1 0
shape=(8,3)
The **axes** of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.
The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which contains this tuple.
axis = 2
0
1
- all elements must be of the same dtype (datatype)
- the default dtype is float
- arrays constructed from list of mixed dtype will be upcast to the "greatest" common type

Figure 1: Anatomy of a SciPy array

Let's say that you have a 2-D 5x5 element array, and that you want to alter all the elements that correspond to the 3rd place in the 1st dimension:

```
>>> c = sp.zeros([5,5])
>>> print c
>>> c[3,:] = 20
>>> print c
```

This is accomplished using the colon character, which basically represents all elements in a column. Similarly, you can print, or otherwise reference, a range of elements in a particular dimension:

```
>>> print c[3,1:4]
```

will print c[3,1], c[3,2] and c[3,3]. Remember, Python handles ranges slightly different than most other languages.

Every array has an attribute shape that returns the dimensions of an array:

```
>>> print b.shape
```

Note that this is a touple, which makes sense as you certainly don't want the shape of b to be able to change unless you change b itself.

## 3.1 Array Math

Scipy makes available basic array math as part of the array class of objects. Let's define a couple arrays to do some math on:

```
>>> arr = sp.array([i for i in range(100)])
```

will create a 100 element array that starts at 0 and increments by 1 up to 99. Similarly you could do the same thing using:

```
>>> arr = sp.linspace(0,99,100)
```

or

```
>>> arr = sp.arange(100)
```

though the data type of the array elements will depend on the method used.

Let's also import a few constants and use them.

```
>>> import scipy.constants as const
>>> arr2 = sp.sin(arr * const.pi/360)
```

This gives us part of a sin wave. We can add arr and arr2 together element-wise using the plus sign:

```
>>> print arr + arr2
```

Subtraction works the same way.

```
>>> print arr - arr2
```

These operations work for an N-dimensional array, as long as they have the same dimensions.

```
>>> print arr * arr2
```

will multiply the arrays together element-wise (not using matrix-multiplication!). Division works the same way.

### 3.2 Some array functions

You've met a few useful functions that work on arrays: zeros, ones, sin. There are many, many others. For an extensive list, see http://www.scipy.org/Numpy_Example_List_With_Doc. Here are a few that you should know right away.

```
>>> s = sp.sum(arr)
```

will add up all of the elements in a given array and return a scalar. If we make a multi-dimensional array:

```
>>> newarr = sp.zeros((100,2))
>>> newarr[:,0] = arr
>>> newarr[:,1] = arr
>>> s0 = sp.sum(arr,axis=1)
```

will sum the array along a particular axis, and return and array of that has N-1 dimensions.

```
>>> a = sp.mean(newarr)
```

will return the mean of an array. The axis argument is also optional.

Another useful function is where:

```
>>> a = sp.where(condition, x, y)
```

which returns an array with the same shape as the condition, where values from x are inserted in positions where the condition is True and values from y where the condition is False.

```
>>> a = sp.where(arr < 50, 1, 0)
```

If you leave off the x and y arguments, the where function returns the indices where the condition was met.

Again, this is just a snippet of the functions available for arrays. If there is something that you need to do that is reasonably common, chances are it is included in SciPy.

## 4 A Note on subpackages

A few of the subpackages are worth pointing out here, at least at a cursory level.

### 4.1 The special subpackage

SciPy provides access to a number of special functions in the special subpackage.

```
>>> from scipy import special
```

This subpackage contains tools to define bessel functions, spherical harmonics, Legendre polynomials, orthogonal polynomials.

### 4.2 The integration subpackage

There are several integration functions provided as part of the SpiPy distribution in the integration subpackage. You can choose from a variety of integration schemes (such as the trapezoidal and Simpsons rule). In addition, there are functions that can be used to integrate ordinary differential equations.

### 4.3 The interpolation subpackage

SciPy makes several general interpolation methods available for data in any dimension. Common functions are interp1d (1-D interpolation). griddata (multivariate data interpolation), and several spline functions.