**Lecture 3 (week 2.2)**

Thursday 15th 2016

Introduction to NumPy

The fundamental library needed for scientific computing with Python is called NumPy.

This Open Source library contains:

* a powerful N-dimensional array object
* advanced array slicing methods (to select array elements)
* convenient array reshaping methods

and it even contains 3 libraries with numerical routines:

* basic linear algebra functions
* basic Fourier transforms
* sophisticated random number capabilities

Today’s class is about learning how to create and manipulate arrays, look at some of the most common methods applied on arrays. The more advanced NumPy concepts (such as array versus matrices) will be discussed at the seventh week of class during the “A deeper look at NumPy” lecture.

NumPy's main *object* is the homogeneous multidimensional array called *ndarray*.

* This is a table of elements (usually numbers), all of the same type, indexed by a sequence of positive integers. Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets, but are not limited to these.
* Dimensions usually called axes, number of axes is the rank

Example:

**In [1]: import numpy**

**In [2]: a = numpy.array([1,3,5,7,9])**

**In [3]: b = numpy.array([3,5,6,7,9])**

**In [4]: c = a + b**

**In [5]: print(c)**

**[ 4 8 11 14 18]**

Instead of typing **numpy** everytime we use an array, it is common to rename it, e.g. as np with the following modified import command:

**In [6]: import numpy as np**

**In [7]: a = np.array([1,3,5,7,9])**

**In [8]: b = np.array([3,5,6,7,9])**

**In [9]: c = a + b**

**In [10]: print(c)**

**[ 4 8 11 14 18]**

A numpy array is NOT a list:

**In [1]: a=[1,2,3,4]**

**In [2]: a**

**Out[2]: [1, 2, 3, 4]**

**In [3]: print(a)**

**[1, 2, 3, 4]**

**In [4]: type(a)**

**Out[4]: list**

**In [5]: import numpy as np**

**In [6]: a=np.array([1,2,3,4])**

**In [7]: a**

**Out[7]: array([1, 2, 3, 4])**

**In [8]: type(a)**

**Out[8]: numpy.ndarray**

Python lists will be introduced later in the course.

Meaning of rank, row, column, length and axis:

[7, 5, -1] An array of rank 1 i.e. It has 1 axis of length 3

[ [ 1.5, 0.2, -3.7] , An array of rank 2 i.e. It has 2 axes, the first

[ 0.1, 1.7, 2.9] ] length 2, the second of length 3

NumPy arrays attributes:

* **ndarray.ndim**
  + the number of axes (dimensions) of the array i.e. the rank.
* **ndarray.shape**
  + the dimensions of the array.
* **ndarray.size**
  + the total number of elements of the array, equal to the product of the elements of shape.
* **ndarray.dtype**
  + an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. NumPy provides many, for example bool\_, character, int\_, int8, int16, int32, int64, float\_, float8, float16, float32, float64, complex\_, complex64, object\_.
* **ndarray.itemsize**
  + the size in bytes of each element of the array. E.g. for elements of type float64, itemsize is 8 (=64/8), while complex32 has itemsize 4 (=32/8) (equivalent to ndarray.dtype.itemsize).
* **ndarray.data**
  + the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

If looking at the attributes of the array **a** above, we get:

**In [9]: a**

**Out[9]: array([1, 3, 5, 7, 9])**

**In [10]: a.ndim**

**Out[10]: 1**

**In [11]: a.shape**

**Out[11]: (5,)**

**In [12]: a.size**

**Out[12]: 5**

**In [13]: a.dtype**

**Out[13]: dtype('int64')**

**In [14]: a.itemsize**

**Out[14]: 8**

**In [15]: a.data**

**Out[15]: <memory at 0x10f5ad7c8>**

NumPy array creation and use

**In [17]: a=np.array([[1, 2, 3], [3, 6, 9], [2, 4, 6]])** *# creates a 2-dimensional array*

**In [18]: a**

**Out[18]:**

**array([[1, 2, 3],**

**[3, 6, 9],**

**[2, 4, 6]])**

**In [19]: a.ndim** *# check attribute ndim*

**Out[19]: 2**

**In [24]: print(a[0])** *# prints a row*

**[1 2 3]**

**In [25]: print(a[1,2])** *# prints an element of the array*

**9**

**In [26]: print(a[1, 1:3])** *# and slices*

**[6 9]**

**In [27]: print(a[:,1])**

**[2 6 4]**

NumPy array methods

With the same array **a** above:

**In [30]: a**

**Out[30]:**

**array([[1, 2, 3],**

**[3, 6, 9],**

**[2, 4, 6]])**

**In [31]: a.sum()**

**Out[31]: 36**

**In [32]: a.mean()**

**Out[32]: 4.0**

**In [33]: a.std()**

**Out[33]: 2.4037008503093262**

**In [34]: a.max()**

**Out[34]: 9**

**In [35]: a.min()**

**Out[35]: 1**

As we have seen before, you can access the list of methods (and more) with the help command:

**In [11]: a = np.array([1, 2, 3])**

**In [12]: help(a)**

help returns WAY more information than you need, and lot of this information is too advanced to be explained at this stage of the course. But if you scroll down the help menu, you’ll see the methods above are listed, as well as a few others that are useful.

An exhaustive list of numpy methods is given there:

<http://docs.scipy.org/doc/numpy/reference/routines.html>

NumPy mathematics:

You can use the conventional operators +, -, /, \* for numpy arrays. You can also use methods for all mathematical operations:

**In [37]: np.sqrt(a)**

**Out[37]:**

**array([[ 1. , 1.41421356, 1.73205081],**

**[ 1.73205081, 2.44948974, 3. ],**

**[ 1.41421356, 2. , 2.44948974]])**

Sorting:

**In [44]: x = np.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])**

**In [45]: x**

**Out[45]: array([ 4.5, 2.3, 6.7, 1.2, 1.8, 5.5])**

**In [46]: np.sort(x)**

**Out[46]: array([ 1.2, 1.8, 2.3, 4.5, 5.5, 6.7])**

A list of numpy mathematical function available is given there:

<http://docs.scipy.org/doc/numpy/reference/routines.math.html>

Online NumPy tutorial

Read the NumPy sections (Arrays, array indexing, datatypes, array math, **do NOT read** broadcasting, this is a more advanced concept which I will cover in our second numpy class later in october) there:

<http://cs231n.github.io/python-numpy-tutorial/#numpy>

Note that the above tutorial was written for Python 2. We are using Python 3, the main difference for us is the syntax of the print statement which is **print()** in Python 3 and not just **print** as in Python 2.