Multidimensional Arrays

Terminology

- A *scalar* is a single item (real/float, integer, character/string, complex, etc.)
- An *array* contains data of the **same type** with each scalar element addressed by *indexing* into the array.
- An array has one or more *dimensions*. The *bounds* are the lowest and highest indexes. The *rank* is the number of dimensions.

NumPy

Use a tuple for dimensions:

```
    A=np.empty((N,M))
```

I can't find a maximum rank, might be when you run out of memory.

- Python is dynamically typed so usually we don't need to declare a type, but in a few cases we should or must:

```
Z=np.zeros((3,4),dtype=complex)
M=np.array([True, True, False, False],dtype=bool)
```

Note: Nearly everything to do with arrays is from numpy. See documentation at

http://www.scipy.org/Tentative_NumPy_Tutorial

Loop Bounds and Indices

- Python starts numbering at 0 and you can’t change that.
- Address elements using square brackets

```
A[i,j,k]
```

Python Array Construction

- Allocation and initialization can be done in one step.

```
A=np.array([(1,2,3),(4,5,6)])
A=np.zeros( ( 2,3) )
A=np.ones( ( 4, 5, 6) )
```

`A=np.eye(2)` 2x2 (identity only defined for square arrays)

Array Ranks

- *You can* declare arrays Nx1 and 1xN as well as N.   Sometimes you may wish to do this but it's not mandatory.
- There is a distinction between Nx1 and 1xN
- Python/NumPy considers these to be rank 2 and not rank 1 arrays.
- It will treat a rank-1 array as either row or column appropriately so normally we don't explicitly make an array Nx1 or 1xN, we just use a rank-1 size N array.

Orientation

- “Orientation” refers to how the array is stored *in memory*, not to any mathematical properties.
- Python is *row-major* oriented. Array elements are stored by rows in memory.
- Loop indices should reflect this whenever possible (when you need loops).
- Innermost first. Left to right. (May not matter much since loops are slow.)

  A[i,j,k] loop order is for i/for j/for k

Python (NumPy) Array Operations

- Arithmetic and many math functions are overloaded and operate elementwise.
- dot(a,b) multiplies via linear-algebra definition.
- Transpose is a.T

Shamelessly Stolen From NumPy Page

- ndarray.ndim the number of axes (dimensions) of the array. In the Python world, the number of dimensions is often referred to as *rank*.
- ndarray.shape the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with *n* rows and *m* columns, shape will be (n,m). The length of the shape tuple is therefore the rank, or number of dimensions, ndim.
- ndarray.size the total number of elements of the array. This is 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 a bunch of them, 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. For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). It is equivalent to ndarray.dtype.itemsize.

Frequently Used NumPy Intrinsics

- all, any,where
- append, delete, insert,resize (you can expand an array after the fact but this will be slow)
- arange
- array
- compress
- copy
- ones,zeros,empty
- fromfile,loadtxt
- reduce,repeat,reshape
- shape,size
- rollaxis,swapaxes,transpose
- abs, cos, sin, tan <several others>
- average, mean, median,std
- ceil, floor
- dot
- sum, prod
- min, max
- argmin, argmax
- nan,isnan
- inf,isinf
- linspace
- lstsq

NumPy Matrix Class

- NumPy has a matrix class that is different from a NxN array in that it has different operations defined on it. In particular, * means matrix multiplication and not elementwise multiplication. * must return a matrix.
- To do elementwise multiplication use
  multiply(a,b)
- a.I : inverse and a few others are defined (inversion might be SLOW)
- On the whole I don't recommend you use matrix. Stick to arrays.

Array Slicing

`A[S1:E1,S2:E2]`

This ACTUALLY goes from S1 to E1-1 and S2 to E2-1 as usual. So E1 and/or E2 can exceed the bound (by 1).

`A[:,1]` This is the second column

Contour Plotting in Matplotlib

- http://matplotlib.sourceforge.net/examples/pylab_examples/contour_demo.html

Three Dimensional Plotting

```
from mpl_toolkits.mplot3d
 import Axes3D from matplotlib
 import cm from matplotlib.ticker
import LinearLocator, FixedLocator, FormatStrFormatter import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d’)
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.jet, linewidth=0,
antialiased=False)
ax.set_zlim3d(-1.01, 1.01)
ax.w_zaxis.set_major_locator(LinearLocator(10))
ax.w_zaxis.set_major_formatter(FormatStrFormatter('%.03f'))
fig.colorbar(surf, shrink=0.5, aspect=5) plt.show()
```