Scientific Computing with Python

Arrays

- Similar to lists, but not as versatile
- All elements must be of the same type
- The size cant be changed after creation
- Not part of standard Python....

- Much faster for computation!

Numerical Python Arrays

- Stored in the scipy or numpy module
  - import numpy as np
- Additional math functionality in these modules as well
- Vectorization is included
- Given a list r:

```
a = array(r)
```

- Or create an array of zeros of size N:

```
a = zeros(n)
```

Creating Arrays

- Create an array from p to q with steps of n:

  a = np.arange(p, q, n)

- Alternatively, create an array of length n from p to q

  a = np.linspace(p, q, n)

Array Slicing

- Array indexing is similar to that of lists
  - We slice them in a similar manner too

```
b = a[1:-1]
b[2] = 0.1
```

- The sub-array, *b*, is **not a copy of *a***, but rather references the elements of *a*
- Thus, b[2] = 0.1 changes both *a* and *b*

Copying Arrays

- There are times when we want to copy an array and modify the copy without affecting the original
- Use the array's copy routine

```
>>> a = x.copy()
>>> a[-1] = 9
>>> a
array([ 1.,  2.,  9.])
>>> x
array([ 1.,  2.,  3.])
```

Vectorization

- Forget Loops!
- Given an array x, compute some function on each element of x
- Loop:

```
r = zeros(len(x))
for i in xrange(len(x)):
    r[i] = sin(x[i])*cos(x[i])*exp(-x[i]**2) + 2 + x[i]**2
```

- Vectorized:

```
r = sin(x)*cos(x)*exp(-x**2) + 2 + x**2
```

Vectorization

- Loop:

```
x = zeros(N);  y = zeros(N)
dx = 2.0/(N-1) # spacing of x coordinates
for i in range(N):
    x[i] = -1 + dx*i
    y[i] = exp(-x[i])*x[i]
```

- Vectorized:

```
x = linspace(-1, 1, N)
y = exp(-x)*x
```

Computational Efficiency

- For an array x, which is more efficient?
  - Built-in routine: sum(x)
  - Numpy routine: np.sum(x)
  - Array routine: x.sum()

- We can use the Unix 'time' command or the Python 'timeit' module

Array Unitary Functions

Try to use these whenever possible!

- Prod
- Sum
- Mean
- Var
- Std
- Round
- Conj
- Trace
- Max
- Min

Vector Operations

- Numpy arrays can be treated as vectors
  - Vectorization supports scalar multiplication
  - Vector multiplication stored in routines:
    - numpy.dot(vec1, vec2)
    - numpy.cross(vec1, vec2)
- Boolean comparison
  - Returns a new array of booleans
  - Built-in functions:
    - Any: returns True if any elements are True
    - All: returns True if all elements are True
    - isreal, isscalar, allclose, isclose, etc.
  - http://docs.scipy.org/doc/numpy/reference/routines.logic.html

Practicals

1) Create an array of cos(x) and a second array of sin(x), spaced in increments of 0.1. Using these, create a third array of $\cos(x)^2 + \sin(x)^2$

2) Write a function that computes:

$$f(x) = x^3 + xe^x + 1$$

And apply this function to each element in an array

3) Write a function that takes in two vectors and prints whether they are parallel, perpendicular, or neither. Assume 3D arrays.

Higher-Dimensional Arrays

- numpy can handle multidimensional arrays
  - numpy.zeros(tuple_of_dimensions)
  - numpy.reshape(array, tuple_of_dimensions)
- Matrix class for two dimensions
  - Has built-in matrix operations (like multiplication and whatnot)
  - Attributes T (transpose), H (Hermitian), and I (inverse)
- Can be indexed as N nested lists or in a C multi-dimensional array format

Numpy Matrices

- Can be created directly from numpy array routines
  - Zeros, ones, ones_like
  - Eye, identity, tri
- Access different matrix sections
  - diag
  - tril, triu
- Convert back to 1D array
  - mat_obj.faltten()
- Treats arrays as 1xN matrices

Linear Algebra

- Matrices are important!
- Linear algebra packs: wrappers for lapack
  - scipy.linalg
  - numpy.linalg
- Eigenvectors/egienvalues found with eig command
  - Eval, eVec = linalg.eig(A)
  - Other optimizations can be used

Practicals

4) Create a an array between 0 and 100 with N*M steps. Convert this into and NxM matrix.

5) Write a function that takes a 1x2 array and an angle. Multiply the array by the rotation matrix R and return the rotated array

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

6) Create a matrix A and show that U_inverse*A*U = diagonal matrix

- U is matrix of eigenvectors
- Print eigenvalues and compare to your answer

Random Numbers

- Modules numpy.random or scipy.random
- Most common function is random
  - numpy.random.random([size])
    - Returns random floats in the half open interval [0.0, 1.0)
- Other distributions as well
  - Poisson, uniform, triangular, you name it!
  - See documentation for everything!
    - http://docs.scipy.org/doc/numpy/reference/routines.random.html

Other Routines

- Integration
  - Module scipy.integrate
  - See Day4/ode.py (Lorentz Flow)
- Fourier Transform
  - Module scipy (or numpy).fft
  - Automatically uses fast Fourier Transform
- Curve Fitting
  - Module scipy.optimize
    - Function 'curve_fit'