Luca Heltai
PhD AMMA, PFC, MHPC, DSSC

Python Short Course
Lecture 11: Numpy Overview

International School for Advanced Studies

NumPy Modules

- NumPy has many of the features of Matlab, in a free, multiplatform program. It also allows you to do intensive computing operations in a simple way
- Basic Module: Array Constructors
  - ones, zeros, identity
  - arange
- LinearAlgebra Module: Solvers
  - Singular Value Decomposition
  - Eigenvalue, Eigenvector
  - Inverse
  - Determinant
  - Linear System Solver

Simple Numeric Constructors

- Arrays are slightly different from lists. They can only contain one type of data structure, and they are much faster to work with numerically. All operations are **element wise**

```
>>> from numpy import *
>>> s = arange(0,2*pi,0.1)
>>> print s
[0., 0.1, ... 6.2]
>>> sin(s) #numpy.sin maps onto arrays
[0., 0.099833, ... -0.0830894]
```

Shape and reshape

```
>>> a = zeros((3,3))
>>> print a
[[0.,0.,0.],
 [0.,0.,0.],
 [0.,0.,0.]]
>>> print a.shape
(3,3)
>>> reshape(a,(9,)) # could also use a.flat
>>> print a
[0.,0.,0.,0.,0.,0.,0.,0.,0.]
```

Arrays and Constructors

```
>>> a = ones((3,3))
>>> print a
[[1., 1., 1.],
 [1., 1., 1.],
 [1., 1., 1.]]
>>> b = zeros((3,3))
>>> b = b + 2.*identity(3) #"+" is overloaded
>>> c = a + b
>>> print c
[[3., 1., 1.],
 [1., 3., 1.],
 [1., 1., 3.]]
```

Overloaded operators

```
>>> b = 2.*ones((2,2)) #overloaded
>>> print b
[[2.,2.],
 [2.,2.]]
>>> b = b+1    # Addition of a scalar is
>>> print b    #   element-by-element
[[3.,3.],
 [3.,3.]]
>>> c = 2.*b   # Multiplication by a scalar is
>>> print c    #   element-by-element
[[6.,6.],
 [6.,6.]]
```

More on overloaded operators

```
>>> c = 6.*ones((2,2),Float)
>>> a = identity(2)
>>> print a*c
[[6.,0.],              # ARGH! element-by-element!
 [0.,6.]]
>>> a.dot(c)
[[6.,6.],
 [6.,6.]]
```

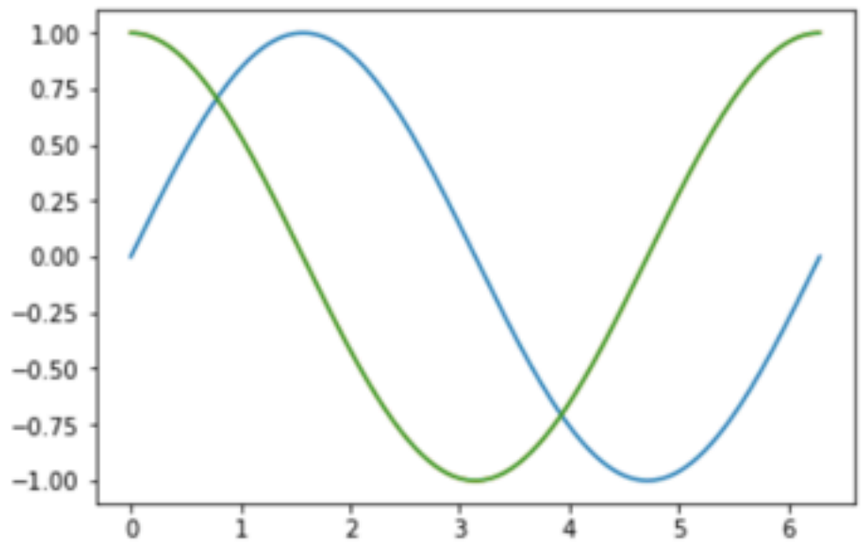
Array functions

```
In [1]: %pylab inline
Populating the interactive namespace from numpy and matplotlib

In [2]: from numpy.linalg import *

In [3]: a = zeros((3,3)) + 2.*identity(3)
        print(a)
[[ 2.  0.  0.]
 [ 0.  2.  0.]
 [ 0.  0.  2.]]

In [4]: inv(a)
Out[4]: array([[ 0.5,  0. ,  0. ],
               [ 0. ,  0.5,  0. ],
               [ 0. ,  0. ,  0.5]])

In [5]: det(a)
Out[5]: 7.9999999999999982

In [6]: det(inv(a))
Out[6]: 0.12500000000000003

In [7]: diag(a)
Out[7]: array([ 2.,  2.,  2.])

In [8]: transpose(a) # same as a. :-)
Out[8]: array([[ 2.,  0.,  0.],
               [ 0.,  2.,  0.],
               [ 0.,  0.,  2.]])
```

Finite Difference Approximation

In [24]:
```
x = linspace(0,2*pi,1025)
f = sin
f_prime = cos
y = f(x)
h = 1e-6
y_prime = (f(x+h)-f(x))/h

plot(x,y)
plot(x,y_prime)
plot(x,f_prime(x))
```

Out[24]: [<matplotlib.lines.Line2D at 0x11043f1d0>]



In [25]:
```
max(abs(y_prime-f_prime(x)))
```

Out[25]: 5.0004445047486749e-07

Numeric Python References

- http://www.numpy.org/NumPy Web Site

Tight Binding References

W. A. Harrison. *Electronic Structure and the Properties of Solids.* Dover (New York, 1989).

D. J. Chadi and M. L. Cohen. "Tight Binding Calculations of the Valence Bands of Diamond and Zincblende Crystals." Phys. Stat. Solids. **B68,** 405 (1975).

http://www.wag.caltech.edu/home/rpm/projects/tight-binding/
My tightbinding programs. Includes one that reproduces Harrison's method, and one that reproduces Chadi and Cohen's methods (the parameterization differs slightly).

http://www.wag.caltech.edu/home/rpm/python_course/tb.py
Simplified (and organized) TB program.