Scientific Computing: Lecture 4

- Arrays and Numpy
- Vectorization of functions with arrays
- File I/O with Numpy
- First steps in visualizing data with Pylab
- Exercises

Class Notes

- Read Chap. 3 and get started on Chap. 4.1 and 4.2 by next Friday
- How is the screencast working? Shared in “ScreenCasts” Box folder.
- HW00 is due tonight and HW01 will be posted.

Numpy array objects

- Python lists are flexible, but SLOW to loop over.
- In scientific computing we often deal with looping over large sequences of data.
- Numpy module offers another data type (object) called an array which is MUCH faster and more efficient.
- Plus adds many standard operations to data [sum, average, stddev,…..]
- Can pass an array to a regular python function and it will return the array with the operation performed on each element in the array -> called vectorization.

Numpy array objects

```
import numpy as np
a1=np.zeros(100) #creates an array of 100 zeros (floating)
I1=np.eye(10) # 10x10 array with I_{i=j}=1. & 0 otherwise.
a2=np.array( [1,2,3,4,5 ] ) #converts a list to array
x=np.linspace(0,10,100) #array of 100 floats b/t 0 & 10
x2=np.arange(0,100,0.1) #array b/t 0 & 10 in steps of 0.1
xy=np.array([1,1],[2,4],[3,9],[4,16]] ) #2D array
```

- Array sizes can not be changed, but shape can.
- All elements must be of the same type (float, int,..)
- Pieces can be accessed by slices like lists.

```
x=xy[:,0] # x becomes the 1st column of xy  (‘:’ means all)
y=xy[:,1] # First index is row #, second is column #
xy[2,1] # returns -> 9 (3rd row, 2nd column)
xy[0,1] = 4.0 # reassigns this element
xy2 = xy # just makes xy accessible by another name
xycopy = xy.copy() #makes a 2nd independent array
```

Array Manipulation

- Many “matrix-like” operations are available for numpy arrays.
- Here are some common ones. See docs for more

```
import numpy as np
x=np.linspace(0,10,100)
y=np.sin(x)*np.exp(-x)
xy=np.array([x,y])
xy.shape   #returns tuple (2,100) – 2 rows, 100 columns
xyT=xy.transpose() # inverts rows and columns
xyT.shape() #returns tuple (100,2)
y.max() #returns max value in y (y.min() also)
y.argmax() #return the index of the max value.
x.dot(y) #returns dot product of 2 1D arrays
x.tolist() # converts the array to a list object
xy.flatten() #turn multidim array into 1D array
```

Clipping Arrays – boolean indexing

- Sometimes we need to remove data in an array above or below some value (oultliers, extreme noise, poles in functions,…
- Can do this with Boolean indexing.
- Could also do this by looping over each value in the array, but it’s much slower!

```
import numpy as np
x=np.linspace(0,10,100)
y=np.sin(x)
yPos = y[y>0]
#returns a new array with only values in y that are >0
y[y<0]=0.
#changes y so that any negative values are replaced by 0.
boolarray= y > 0
# returns a boolean array (True or False) of
#length y where T or F results from the test condition
```

Functions and File I/O with numpy

- Much easier to read in / write out data to files in numpy!

- Must be columns of numbers (can force to skip rows).

- Example: read in a file with 2 columns of numbers, square the 2nd column, and write both columns to a new file.

- Regular old python functions can act on whole arrays at once rather than just a single number at a time. Much faster than calling the function repeatedly in a loop!

```
def squareit(x):
       return x**2
data = np.loadtxt(‘oldfile.dat’)
x=data[:,0]
y=data[:,1]
#Pass an array to a funct and it returns an array
y2= squareit(y) # OR can just use y2=x**2
np.savetxt(‘newfile.dat’,(x,y2))
```

Remote files with urllib2

- Neat trick is to open remote files from the internet using module urllib2.
- `urllib2.urlopen(url_path_to_file)` returns a file-like object which can then be used to read data on a remote machine.

```
import urllib2 as url
import numpy as np
address='http://www.phy.olemiss.edu/~jgladden/sci_comp/
handouts/data.dat’
infile=url.urlopen(address)
x,y = np.loadtxt(infile,unpack=True)
```

numpy in the background - pylab

- Numpy provides a common data structure (arrays) for almost all scientific libraries in python.

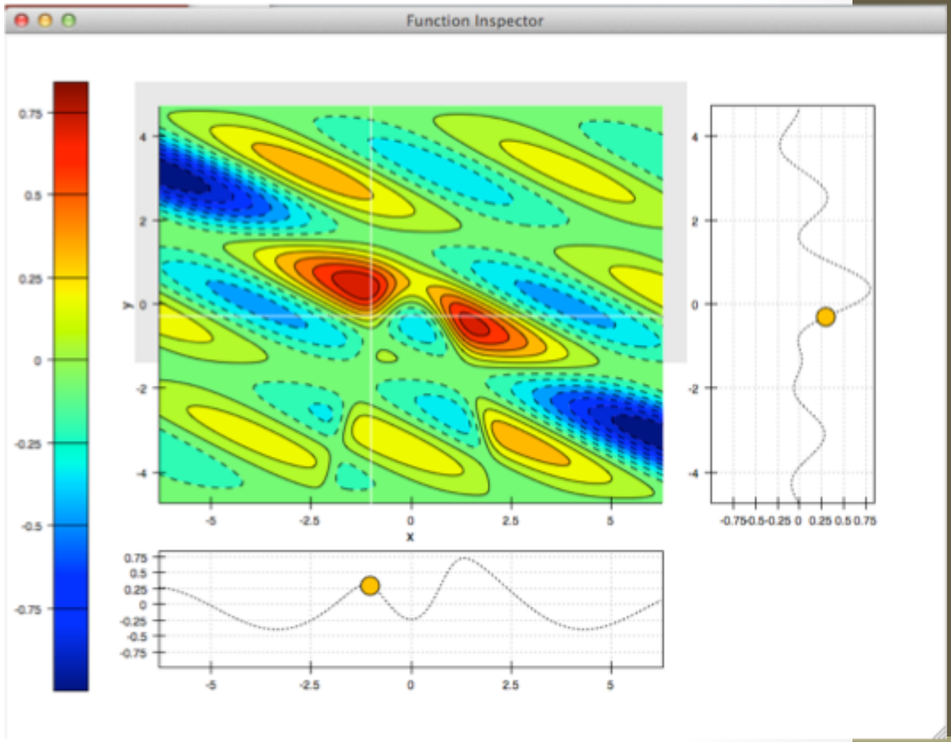
- Pylab (matplotlib is the core graphics engine) is an extensive plotting graphics library with numpy at it’s core.

- Other very nice visualization packages are Chaco (powerful interactivity with plots) and Mayavi (excellent 3D)

- But Pylab is very user friendly yet has power for those who need it, is very widely used and actively developed, makes nice looking plots, and has a syntax very similar to Matlab.

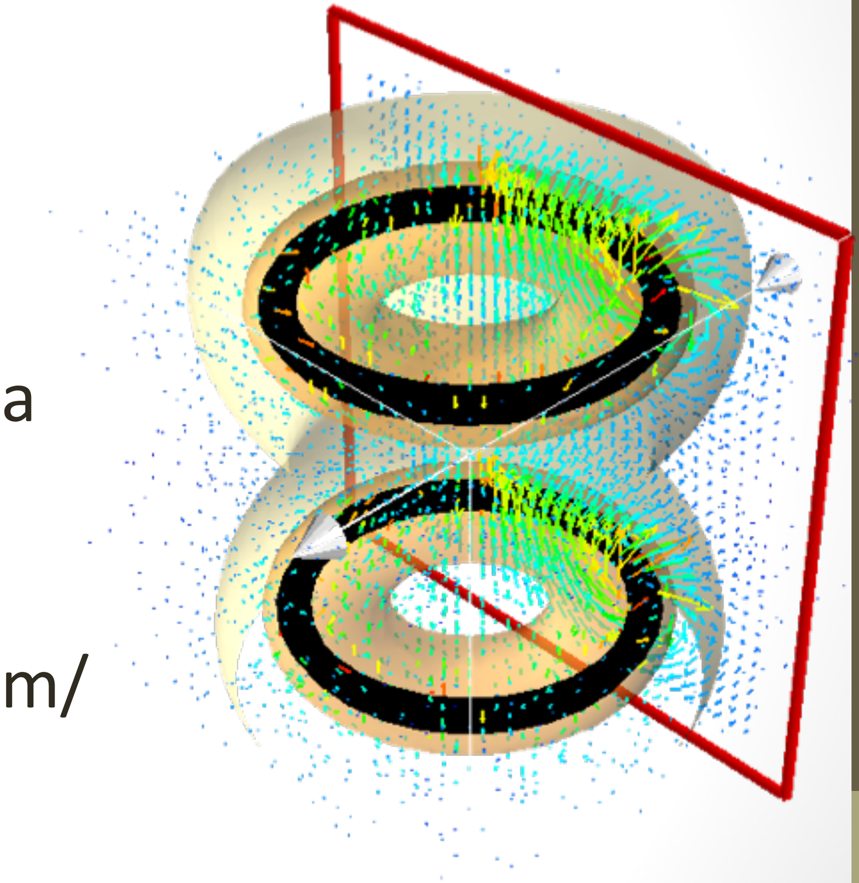
Other options: Chaco

- Chaco offers tools for a HIGH degree of interactivity with your data.
- Cost is a rather steeper learning curve compared to pylab (matplotlib).
- Nice demos with code at the Canopy Chaco website



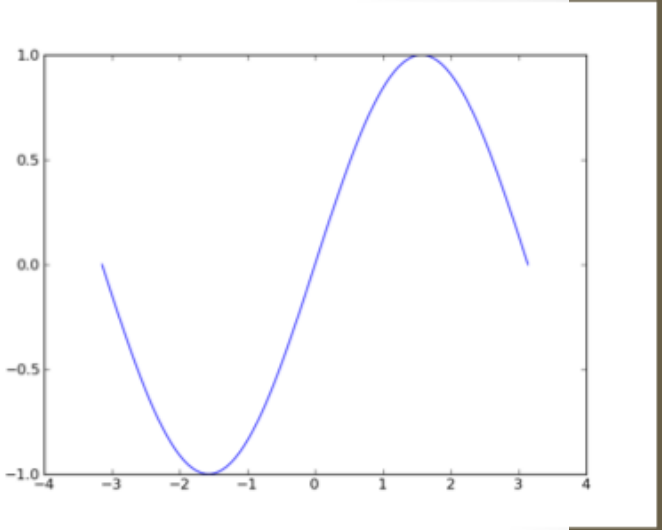
Other options: Mayavi

- Mayvi can actually be a stand alone package, but uses Python under the hood.
- Very sophisticated 3D data visualization tools.
- Examples at: http://docs.enthought.com/mayavi/mayavi/

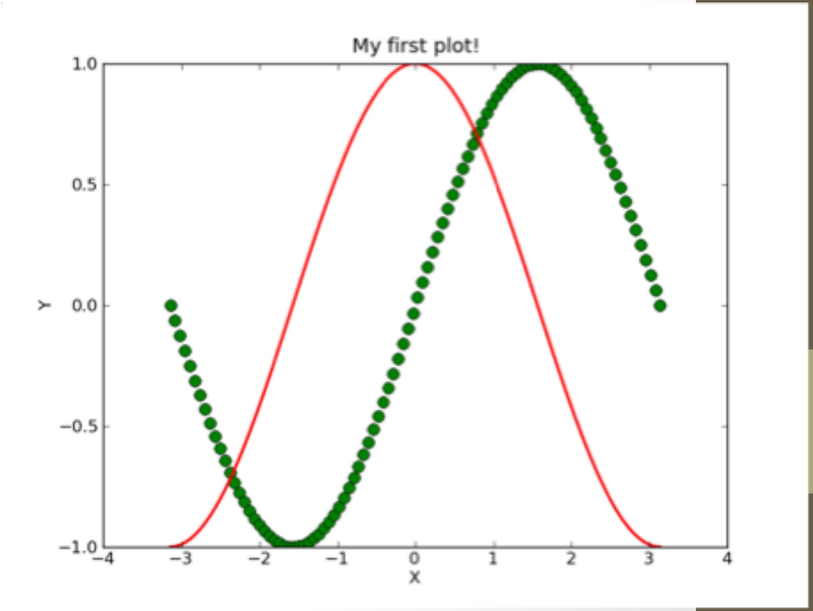


pylab Examples

```
from pylab import *
#numpy will automatically be
#loaded as ‘np’
x=linspace(-np.pi,np.pi,100)
y=sin(x)
plot(x,y) #plot comes from pylab
show()    #displays figure
```



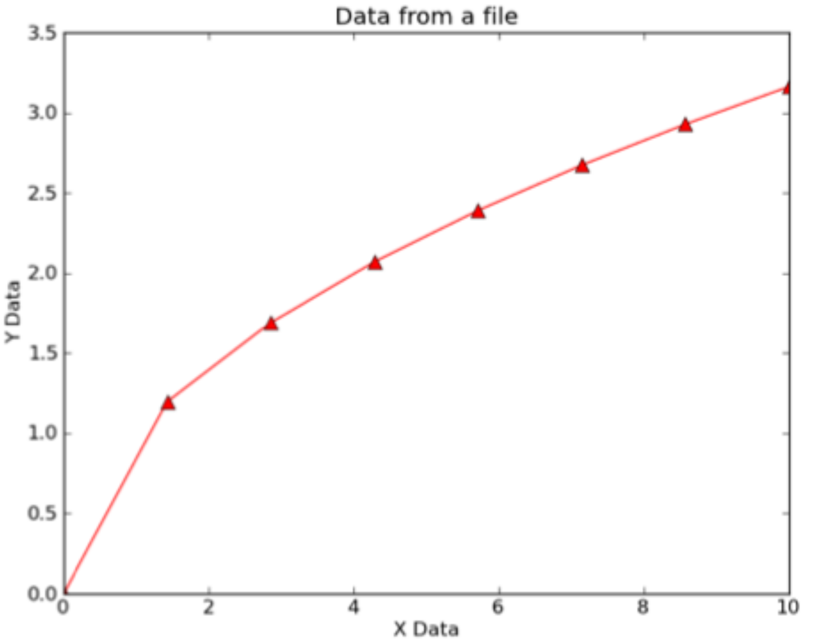
```
from pylab import *
x=linspace(-np.pi,np.pi,100)
y=sin(x)
y2=cos(x)
plot(x,y,’go’,ms=8)
plot(x,y2,’r-’,lw=2)
xlabel(‘X’)
ylabel(‘Y’)
title(‘My first plot!’)
show()
```



Plotting data from a file

- Use numpy loadtxt() to load data from a file.
- pylab.plot takes arrays as data containers
- MANY options! type help(plot) to see.

```
from pylab import *
data=np.loadtxt(‘mydata.dat’)
x=data[:,0]
y=data[:,1]
plot(x,y,’r-^’,ms=8)
xlabel(‘X Data’)
ylabel(‘Y Data’)
title(‘Data from a file’)
savefig(‘myplot.png’)
show()
```



matplotlib and pylab

- pylab is a convenient interface to the true graphics engine – known as matplotlib.
- Underlying matplotlib can be accessed to display plots in GUI apps, interact with plots, …
- Community has been pushing away from “from pylab import *” method.
- Preferred method is

```
import matplotlib.pyplot as plt
plt.plot(x,y,’o’)
plt.show()
```

plotfile: Quick and dirty

- A new(ish) feature in pylab is `plotfile` to directly and quickly plot data in a file.
- Source file can have labels in 1st row and multiple columns of data.

```
from pylab import *
plotfile(‘mydata2.dat’,(0,1,2),delimiter=‘ ‘)
```

- This makes 2 plots which share a x-axis. (0,1,2) means put data in 1st (0) column on x-axis, data in 2nd (1) column on y_1-axis and data in 3rd (2) column on y_2-axis. Columns are delimited by white space (could be commas, colons,…)
- Structure of ‘mydata2.dat’ is:

```
Time_sec   Temperature_C        Pressure_atm
0.0        22.8                 0.80
1.0        24.9                 0.95
2.0        27.2                 1.34
3.0        30.3                 1.58
4.0        34.6                 1.89
```