

Introduction to Programming for Scientists

Prof. Steven Ludtke

N420, sludtke@bcm.edu

Team Learning 5

Simple Math and Text Handling

Preliminaries:

1) Make sure you have all of the necessary tools installed.

From the system command prompt (not from jupyter), run:

```
conda install matplotlib bokeh ipywidgets pandas
```

2) Test your installation

Launch a new Jupyter notebook (do not use any existing one you may have had open)

Make sure this doesn't raise errors:

```
from pylab import *  
import bokeh,pandas,ipywidgets
```

```
x=arange(0,4*pi,0.05)  
y=sin(x)  
plot(x,y)
```

NumPy (30 min):

1) arrays vs lists

Run this:

```
import numpy as np  
a=[1.0,2.0,4.0,8,12,16]  
b=np.array(a)
```

```
print(a)  
print(b)
```

You'll note that a and b look very similar. Try this:

```
print(repr(a))  
print(repr(b))
```

You'll see that a is a Python list but b is a NumPy array. Arrays are designed for math, and behave differently from lists in many ways, but can still be sliced, etc. For example:

```
print(a[2:4])  
print(b[2:4])
```

You will see if you look carefully that the second value in the first print is an integer, as specified in the original list, but the second value in b is a floating point number! Unlike lists, all values in a single NumPy array MUST be of the same data type. Now try this:

```
print(a*3)  
print(b*3)
```

When you multiply a list by 3, it makes 3 copies of the list. NumPy arrays are geared for math, so when you multiply by 3, it multiplies each element in the array by 3. Try this:

```
print(b)
print(np.sqrt(b))
```

So you can even do fairly complex mathematical operations on NumPy arrays. Could this cause problems if you said "from numpy import * " ?

2) arange

The 'np.arange' function behaves much like Python's 'range' function. The difference is that it can make ranges with fractional values, and produces NumPy arrays. This can be very convenient for generating functions. Say we want to generate a plot of x vs sin(x) over some range:

```
x=np.arange(0,pi*2.0,0.05)
y=sin(x)
print(x)
print(y)
```

Oops. That doesn't work... See if you can figure out what's wrong and fix it, then look at the result.

3) reading files

If you have a text file containing tabular data, it's very easy to read this data into a NumPy array. Download 'curve.txt' from the class website to your computer. Open curve.txt with a text editor to see what's in the file.

Then back in Jupyter:

```
a=np.loadtxt("curve.txt")
print (a)
```

Oops, unless you just happened to download the file to the same location on your computer where you launched Jupyter from, this will fail. The trick is to provide a complete path to where curve.txt is on your computer. Fix it and try again.

You will note that when you print(a), it only shows you a small part of the array if it's large. All of the data is there, it just doesn't want to fill your screen with numbers.

4) writing files

Similarly, if we want to save the data you generated above with arange, you can do that too. First we need to merge x and y into a single 2 column array:

```
cmb=np.column_stack((x,y))
print(cmb)
```

Note that this array is small enough it will print the whole thing.

Then we can write cmb to a file (look at it after saving, but you'll have to find it):

```
np.savetxt("mydata.txt",cmb)
```

Bokeh (30 min)

Both matplotlib and bokeh provide great tools for plotting. In general, matplotlib is more capable than Bokeh with many more options. However, matplotlib only generates fixed figures in Jupyter, whereas Bokeh produces dynamic plots within the web browser, so we will focus on that today. If you need to generate publication quality PDF files, matplotlib is likely the better choice!

1) To start, try this. We will plot the x/y data we generated above

```
from bokeh.plotting import figure, output_notebook, show
```

```
output_notebook()    # output to Jupiter
```

```
p = figure()        # create a new plot
p.line(x, y, legend="y=sin(x)")
p.circle(x, y, fill_color="white", size=5)
show(p)
```

You should see a plot appear in your web browser showing the $\sin()$ curve we generated above in step 2.

2) Experiment with the available widgets in the upper right of the plot. The different buttons allow you to pan, zoom, etc.

3) Let's display the data from the file we read previously:

```
print(a)
```

and again, you should see a partial list of the contents of the array. If you don't go back up to step 3 above.

4) When you call `p.line()`, `p.circle()` or any of the other plot methods, it expects to get an array containing all of the x values and another array with all of the y values:

```
print(x)
print(y)
```

then:

```
print(a[0])
print(a[1])
print(a[2])
```

Do you see the difference? We read the data from the file, but `a[0]` is an x/y pair, not a list of all of the x values.

5) numpy to the rescue! To do this, we need to consider the array as if it were a matrix (ie - linear algebra). The matrix we have is $2 \times n$ and we'd like it to be $n \times 2$ instead. Luckily there is a simple math operation that does this, the matrix transpose:

```
b=a.transpose()  
print(b[0])
```

Much better, now `b[0]` contains all of the x values from the file.

6) For your final exercise, figure out how to plot "b" in bokeh now that we have it in the correct form. Note that you do not need to repeat the import command or the `output_notebook` command, but you will need to make a new `figure()`.

7 (extra credit) See if you can plot the curve from the file "b" and $\sin(x)$ covering the same range as "b" on the same plot.