Python as a plotting tool
(for RMP project of Finlator & Leung, in summer 2012)

This note contains some basic examples of using Python for plotting scientific figure. It is not meant to be complete. Instead it is intended for getting you started with this amazing tool. Once you get the hang of the basics, you are able to learn from online examples and manuals.

While it is assumed that the computer is a Linux machine, the examples should also be applicable to other OS with small changes. Besides the basic Python package, the required libraries are NumPy, SciPy and MatPlotLib. Installing PyLab is an easy way of setting up all the three libraries correctly.

There are three simple ways to use Python as a plotting tool. One can execute the commands in an interactive command prompt. That means one runs Python or iPython directly, and type in the commands. One can also save the commands in a Python script. Finally, one can run the commands in other types of script (most likely bash script in case of Linux). There are different reasons to do each of them, and they all have pros and cons. However, they all can produce the same plots. We will cover the first two usage in this note.

About the notations, an interactive command prompt of Python is >>> and a Linux command prompt is $. Both are at the beginning of a line. Do not include them when you type the command.

1. Example 1: plotting a function

Start Python by

$ python

and enter these commands:

from pylab import *

x = arange(-5.0, 5.0, 0.01)
y = x**2

plot(x, y)
show()
Question 1.1: Do you understand every line in this example? Try changing them to understand the changes.

Question 1.2: Here’s another way of writing the same commands:

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-5.0, 5.0, 0.01)
y = x**2

plt.plot(x, y)
plt.show()
```

Sometimes you may want to import module in similar way (e.g. there could be conflicts between the definitions of functions, especially if you go beyond PyLab).

Question 1.3: You may want to turn interactive mode on by using ion(). Then every `pyplot` command triggers a figure redraw. Sometimes, after a command error, auto redraw does not work properly. You can run draw() to force a redraw.

In some case you may want to turn the interactive mode off by ioff(). Why?

Question 1.4: You may find that it’s easier to use

```
$ ipython -pylab
```

if you want to run Python interactively. There are some enhanced features compared to simple Python prompt (one particular useful feature, at least for me, is that command history is saved between sessions).

2. Example 2: labels, Python script

First, save the following in a script (say, a text file called ex02.py). Use simple text editor (e.g. gedit or nano) instead of word processor (e.g. MS Word or OpenOffice Writer). The content of the script is:

```
#!/usr/bin/env python

from pylab import *

t = arange(0.0, 2.0*pi, 0.1)
```
y = sin(t) + 0.1 * sin(20.0*t)

xlabel('time (sec)')
ylabel('function')
title('Superposition of waves')

plot(t, y)
show()

Second, make the file executable by running

$ chmod u+x ex02.py

Third, run the script:

$ ./ex02.py

Question 2.1: What if you change the increment of the array from 0.01 to 0.1? What happens and why?

Question 2.2: What if there is an apostrophe in the title or label? There are at least two ways to get around it.

Question 2.3: The shebang line `#!/usr/bin/env python` indicates the interpreter of the script. You can also write a script (say, ex02.1.py) without the shebang:

```python
print "testing 123"
```

The file does not need to be executable. It is run by command like this:

$ python ex02.1.py

### 3. Example 3: line types, colors, ...

Plot a figure with these commands:

```python
from pylab import *

x = arange(0.0, 8.0, 0.01)
y1 = cos(2.*x)
y2 = cos(2.*x - pi/4.)
```
y3 = cos(2.*x - pi/2.)
y4 = cos(2.*x - 3.*pi/4.)

xlabel('x')
ylabel('y')

plot(x, y1, 'k-')
plot(x, y2, 'b--')
plot(x, y3, 'r:')
plot(x, y4, 'g-.')

show()

Question 3.1: Try colors c, m, y and w. Can you see a plot with color w? Why? (Tips: Use clf() to clean up plots before replotting; Use up arrow to recall used command so that you don’t have to type the whole line again; If up and down arrows do not provide you with used command, you can run Python by command:

$ rlwrap python

or reinstall/recompile Python with GNU readline support.)

Question 3.2: How do the curves look like if you use linestyles ., , o, s, ^, D, + or x?

Question 3.3: Try plot(x, y1, 'k-', linewidth=2.0). Now you know how to change line styles and width, but do we really need them? Isn’t it enough to use different colors for different lines? (Hint: imagine printing a plot in black and white)

4. Example 4: comment line, latex, output figure file

In Python, a single-line comment starts with a pound sign (#), and a multiple-line comment is placed between triple apostrophes (''') or trip quotation mark ("""). Certainly you can also use pound sign on each line of a multiple-line comment.

It is a good practice to add comment to your script to improve readability (may not be very useful for short script, but it is important in case the script is shared among people, or the script is long). Also, one simple comment of the (intended) usage of a script may save you the time to read through it.

Now try this example:
Plotting a function and the asymptotic lines
Last updated: Jun 2012

from pylab import *

x = arange(-5,5,0.01)

# function:
y1 = sqrt(1 + x**2/pi)
y2 = -y1

# asymptotic lines:
ya1 = x / sqrt(pi)
ya2 = -ya1

plot(x,y1,'k-')
plot(x,y2,'k-')
plot(x,ya1,'r:')
plot(x,ya2,'b:');

ylabel('$y$')
xlabel('$x$')
title(r'$y^2-x^2/\pi=1$') # title written as TeX equation

show()

Question 4.1: Try $\alpha$, $\sqrt{2}$, and $\int_0^1 x^2 \, dx$ in the label or title. These expressions are written in TeX (or more commonly referred as LaTeX, though they are not identical). Do you understand the notation? If not, google them. (Note: by adding r in the title command, the string is viewed as "raw", and it will not be "escaped". Sometimes there is no notable effect, but without it a LaTex expression may produce strange result.)

Question 4.2: Try the command `savefig('plot4.eps', format='eps')` and check whether there is a new file in the directory. You can view the file by using `gv` or `display`. The format is *encapsulated postscript*, which is a common format of figures in scientific writing.

For presentation (i.e. making powerpoint), it may be easier to use image formats such
as *jpg* or *png*. Can you figure out how to do it?

Question 4.3: If you are running *Python* interactively, you can click on one of the buttons to save the plot as your format of choice. Try it.

If there is such a simple way to save a plot, why do we need a command? (Hints: what if you *ssh* from another machine and the connection is slow? Do you still want graphical interface? Another scenario is that you have to replot similar figure again and again...)

5. **Example 5: import data file**

Type the following in a text file (say, *ex05.data.txt*):

```python
0.0 11.2
5.0 3.9
10.0 6.8
15.0 2.3
20.0 -1.2
```

Here are the commands to plot them:

```python
from pylab import *

data = loadtxt('ex05.data.txt')

plot(data[:,0], data[:,1])
show()
```

**Question 5.1:** What happen if you use `plot(data)` instead? If there are two inputed 1-dimensional arrays, `plot()` view them as *x* and *y*. Run `print data`, `print data[:,0]`, and `print data[:,1]` to see the outcome. Try to understand how `plot()` works.

**Question 5.2:** You can also import each column as an 1-dimensional array. So how do you do it?

6. **Conclusion**

Now you have learned the basics of using *Python* as a plotting tool. There are still a lot to learn. Consult online examples and manuals, and remember that Google is your friend.
As you are getting familiar with Python, you may also want to try making a log-log plot, changing the ticks, changing font size and type, adding legends, making subplots, adding error bar, importing multi-column data, and doing arithmetic operation on arrays. All these are useful in plotting scientific figures.

Some final words about how to write good code. Just like any computer language, there are always more than one way to use Python to achieve your goal. In a lot of case it is a matter of taste. So, develop good taste! Remember that computer code is read much more often than it is written. That means it may be a good idea to write a code in a long but clear way, instead of a short but obscured way. After all, you would want to able to understand or even reuse your own code in the future.