Learning Python

Getting results for beamlines and scientific programming

Using python: Simple plotting with Matplotlib

Outline of topics to be covered

Creating graphics

- One of the nice aspects of programming in Python is that high quality display of manipulate-able graphics (line plots & images) is quite easy.

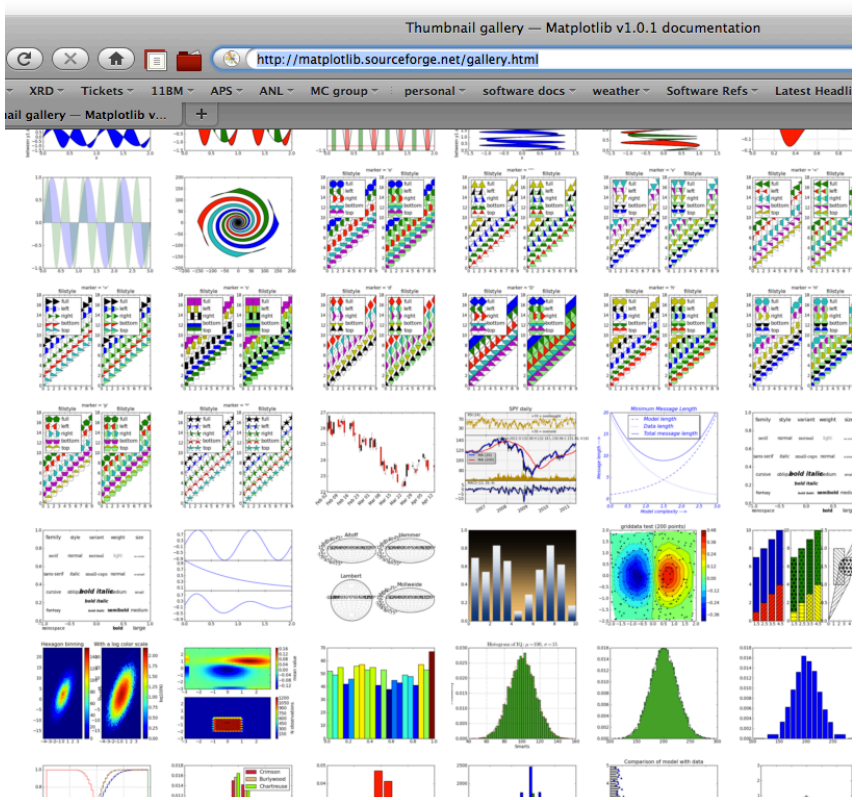
- I will show you how to use a graphics library called matplotlib for this, but Python offers other choices. The plotting package chaco is very good and is preferred by the folks at Enthought for more sophisticated scripts.

Matplotlib

matlibplot: publication quality graphics as well as quick-and easy graphics with direct coupling to beamline operations and/or high-level mathematical computation library

- Information on matplotlib:
  - Download the ~1000 page matplotlib user's guide and manual from http://matplotlib.sourceforge.net/contents.html
  - book: *Matplotlib for Python Developers* by Sandro Tosi (Packt Publishing, 2009)

- Note there is a nice gallery of hundreds of sample plots along with the example code to generate them: http://matplotlib.sourceforge.net/gallery.html



Three ways of using matplotlib

- I find matplotlib a bit hard to follow because it was written to be used in three ways:
  (1) Interactively (simple),
  (2) Standalone graphics (single windows) with simple commands (slightly harder).
  (3) Object-level plotting in matplotlib
  (4) Use of matplotlib together with a GUI package (wxPython, Qt,…) for more complex graphics (to be presented later)

- ***Interactive use***: ipython shell with pylab, provides a "clone" of
- ***Simple Scripting***: write simple scripts that display nice plots; minimal interaction with plot or GUI integration
  – do this by using the matplotlib **pyplot** package
- ***Object-level matplotlib scripting***: this allows for the most sophisticated use of matplotlib: animation, combined use of GUIs & graphics,…

ipython for interactive computation

ipython provides an interactive environment for computation and plotting

- Use of –pylab causes the pylab module to be loaded (matplotlib.pylab & numpy) as import *
- Code to compute a Bessel function
  - N.B. mgrid, sqrt and shape are numpy routines
  - Perhaps better to type longer names: numpy.sqrt, etc.

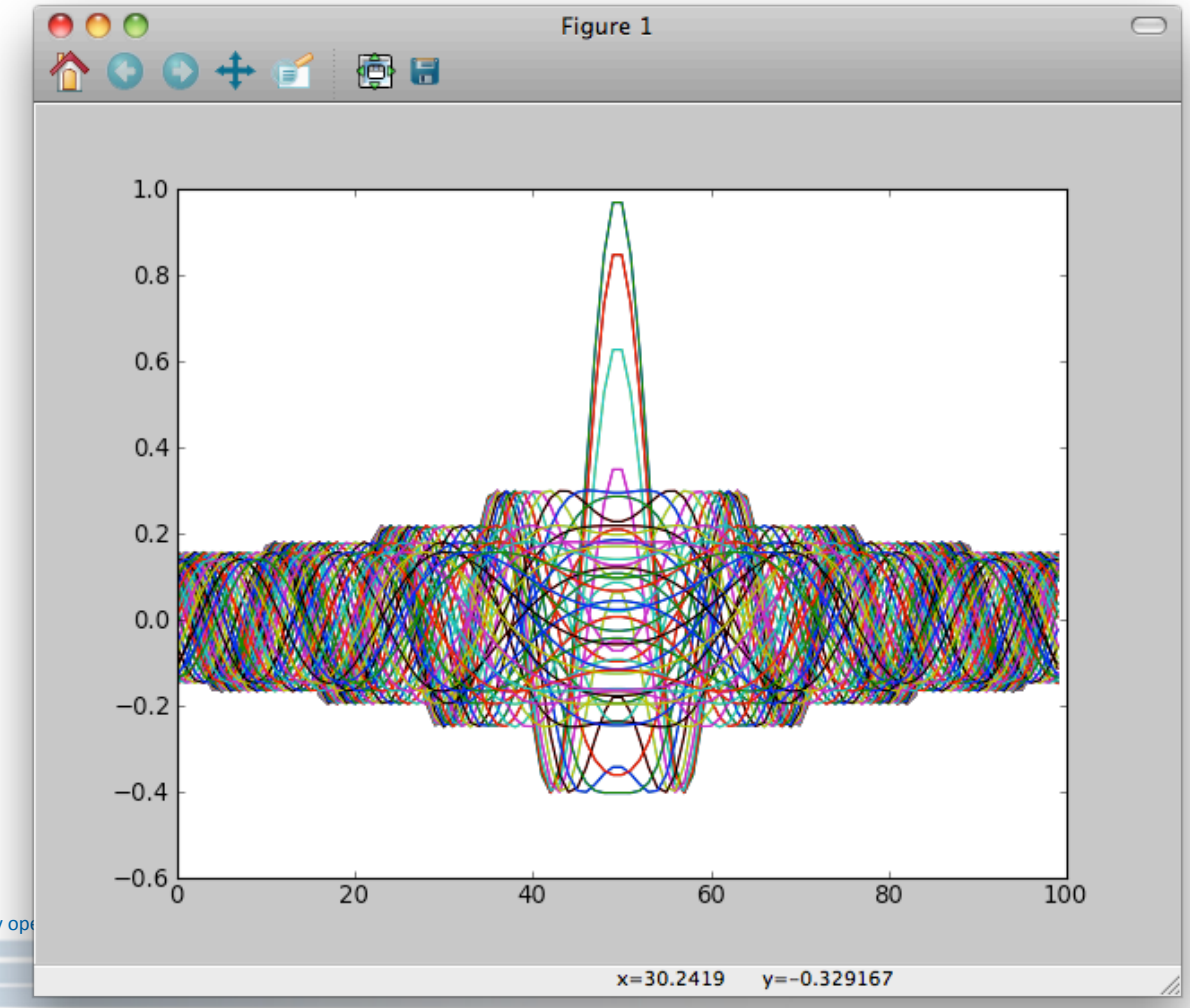
```
BHT3:tmp toby$ ipython -pylab
Enthought Python Distribution -- http://code.enthou
Welcome to pylab, a matplotlib-based Python environ
  For more information, type 'help(pylab)'.

In [1]: x,y = mgrid[-25:25:100j,-25:25:100j]

In [2]: r = sqrt(x**2 + y**2)

In [3]: shape(x)
Out[3]: (100, 100)

In [4]: shape(r)
Out[4]: (100, 100)

In [5]: import scipy.special

In [6]: B = scipy.special.j0(r)
```

ipython for plotting

```
In [9]: plot(B)
Out[9]:
[<matplotlib.lines.Line2D object at 0x5a42550>,...
```

ipython provides easy access to matplotlib plotting routines

It makes life easier by calling show() (updating the screen) for you

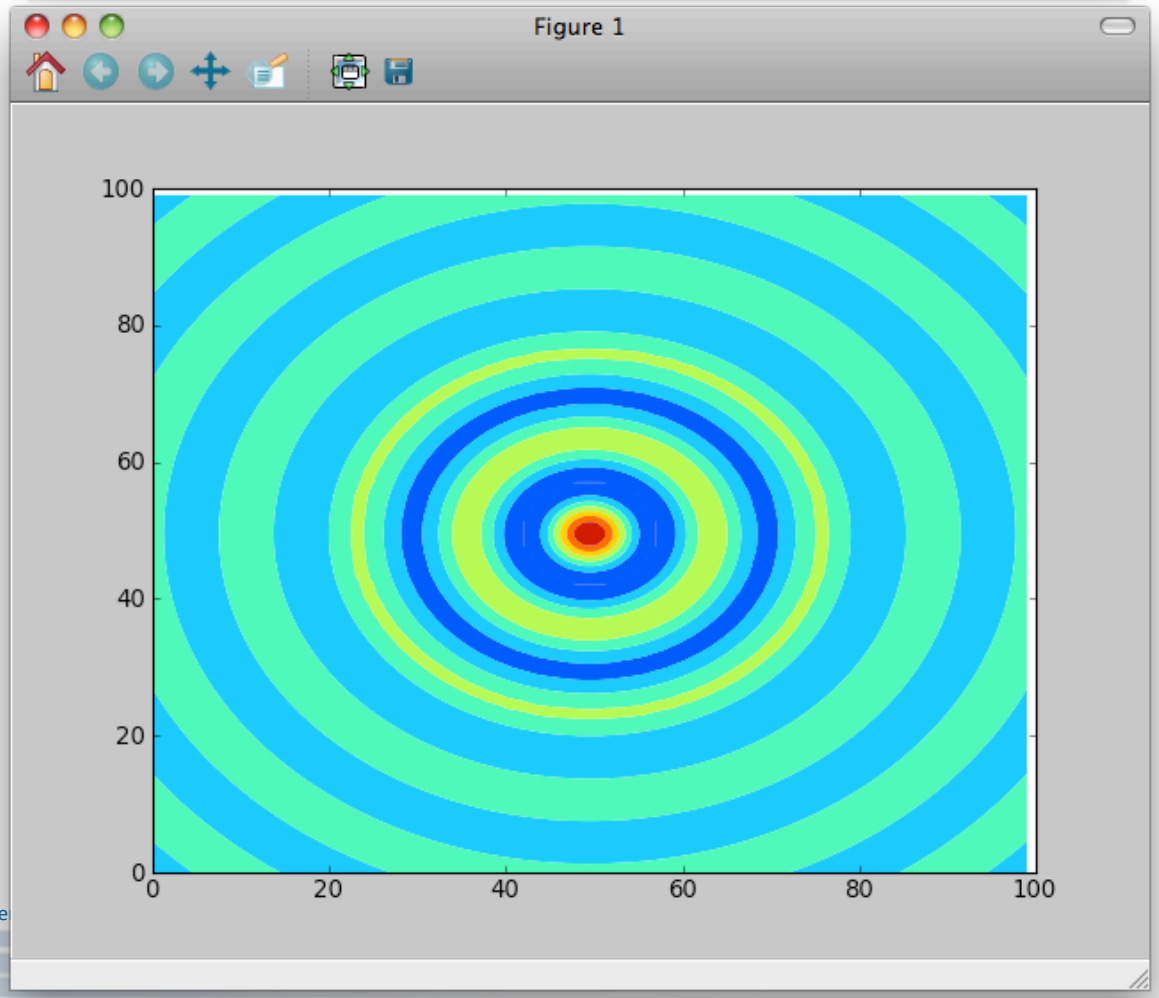


ipython for plotting

```
In [10]: clf()

In [11]: contourf(B)
Out[11]: <matplotlib.contour.ContourSet
instance at 0x5d949b8>
```

The plot can be updated as you go. Note use of clf (clear figure) – otherwise the plots are superimposed

Note that full names are matplotlib.pyplot.clf and matplotlib.pyplot.contourf



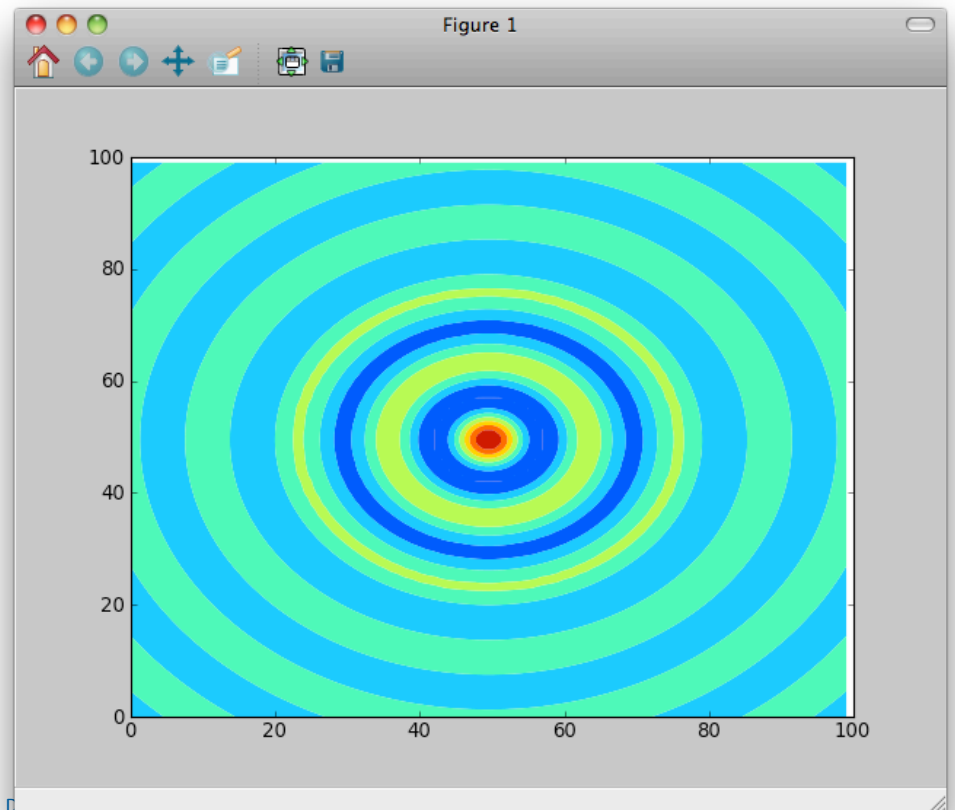
Simple plotting with pyplot

- For simple plotting inside a script we can copy the previous commands to a file and change them to use pyplot .
  - Note that now the pyplot show command is needed or nothing happens.

```
toby$ python pyplot_demo1.py
```

This approach works fine for simple plotting, but the print statement is not executed until after the figure window is closed – this is a problem for more interactive code.

```
import scipy.special
import numpy as np
import matplotlib.pyplot as plt
x,y = np.mgrid[-25:25:100j,-25:25:100j]
r = np.sqrt(x**2 + y**2)
B = scipy.special.j0(r)
plt.contourf(B)
plt.show()
print 'done'
```



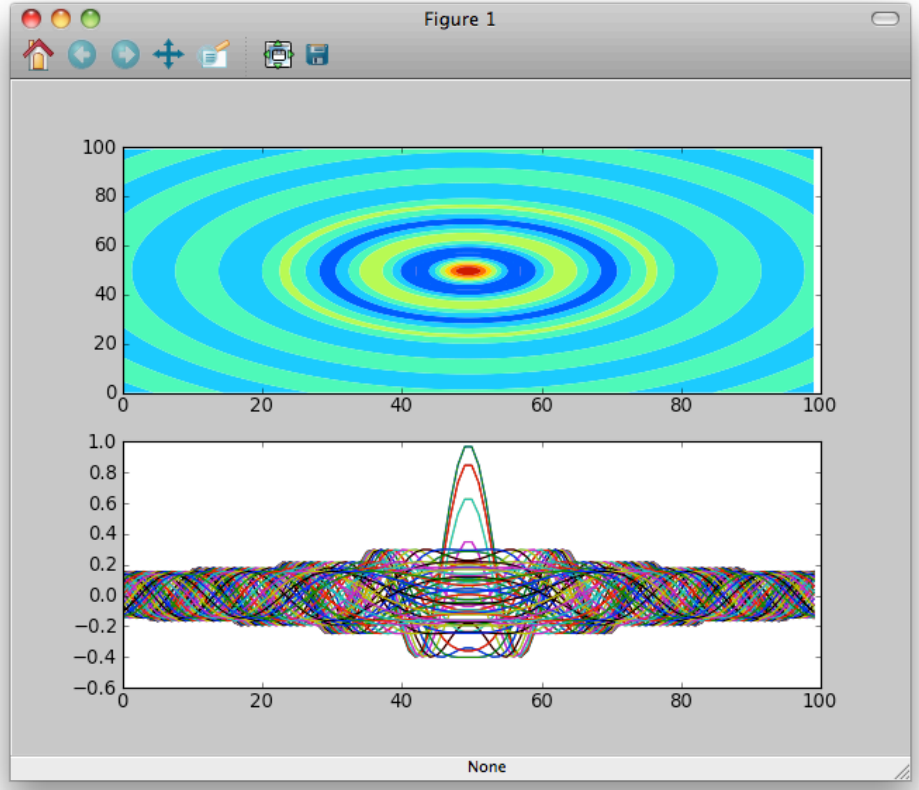
Use of matplotlib objects

- Programming graphics directly in matplotlib requires understanding more about the hierarchy of objects that matplotlib creates:
  - Canvas: highest level object that contains a figure
  - Figure: inside canvas, can hold one or more axes
  - Axes: describes a rectangular region of the figure where drawing is done
- In the example below, we use pyplot to set up the canvas and figure (fig) for us, but then use the objects directly. Function gca (get current axes) creates axes if none exist and function contourf creates the contour map. Finally we use pyplot to show the graphics

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.gca()
obj = ax1.contourf(B)
plt.show()
```

Side-by-side plots

- Rather than using gca to create a single set of axes, we can divide up the figure into multiple axes using `add_subplot(r, c, n)`
  r = rows, c = cols, n = num
  - Note that subplots are numbered from left to right and then top to bottom

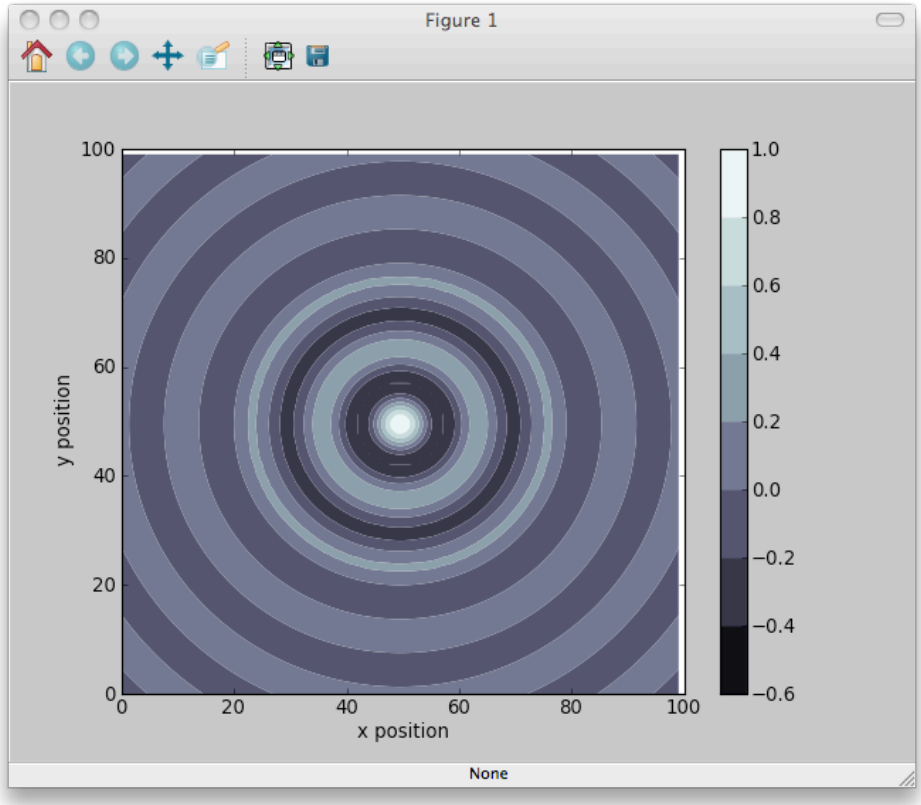



```
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.add_subplot(2,1,1)
ax1.contourf(B)
ax2 = fig.add_subplot(2,1,2)
ax2.plot(B)
plt.show()
```

Modifying the plotting objects

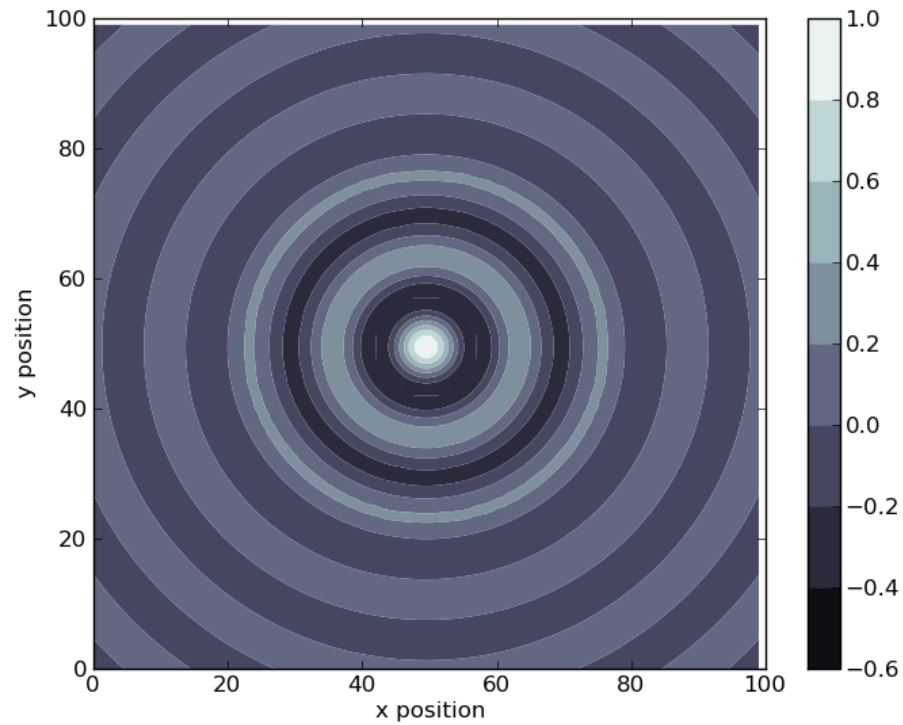
- By creating objects directly, we can modify them more easily. Below we change the color map used to display the contours, add axis labels and we call a method of the figure to display a legend for the color scale.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.gca()
cnt = ax1.contourf(B)
cnt.set_cmap(mpl.cm.bone)
ax1.set_ylabel('y position')
ax1.set_xlabel('x position')
fig.colorbar(cnt)
plt.show()
```



Plotting into a file

- If we want to produce output in a file, we use the matplotlib `use` function to specify that plotting is to occur in memory rather than on a screen. Finally we use the pyplot `savefig` to specify the file, format, (dpi, size…)

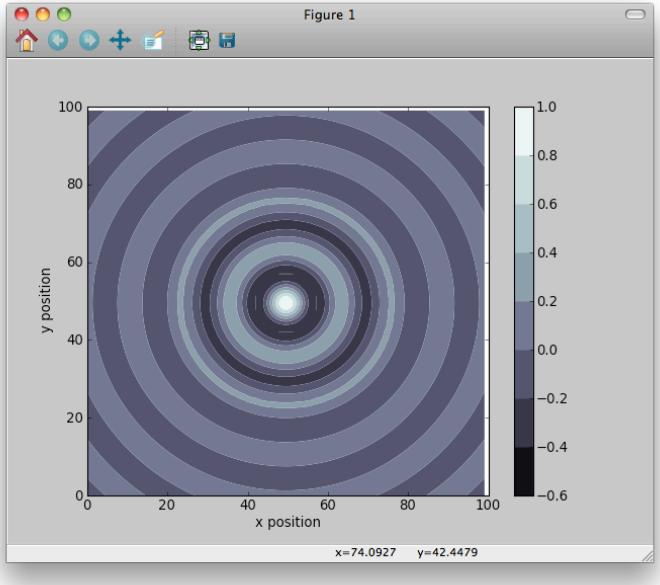


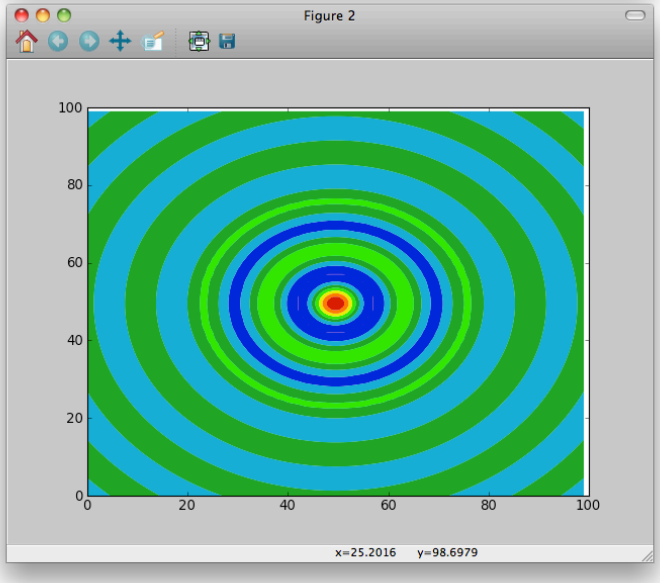

```
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.gca()
cnt = ax1.contourf(B)
cnt.set_cmap(mpl.cm.bone)
ax1.set_ylabel('y position')
ax1.set_xlabel('x position')
fig.colorbar(cnt)
plt.savefig('x.png', format='png')
```

Plotting in multiple windows

- Figures can be numbered, causing plotting to be placed in different windows

```
import matplotlib as mpl
import matplotlib.pyplot as plt

fig1 = plt.figure(1)
ax1 = fig1.gca()
cnt1 = ax1.contourf(B)
cnt1.set_cmap(mpl.cm.bone)
ax1.set_ylabel('y position')
ax1.set_xlabel('x position')
fig1.colorbar(cnt1)

fig2 = plt.figure(2)
ax2 = fig2.gca()
cnt2 = ax2.contourf(B)
cnt2.set_cmap(mpl.cm.spectral)

plt.show()
```





Plotting with tabbed windows

It is convenient to use a single window for plotting different things. This can be done by giving each plot a different tab.

Module plotnotebook.py (on the Python Interest Group site http://www.google.com/search?ie=UTF-8&oe=UTF-8&sourceid=navclient&gfns=1&q=pig) makes this fairly straightforward, without having to learn any wxPython and minimal matplotlib.

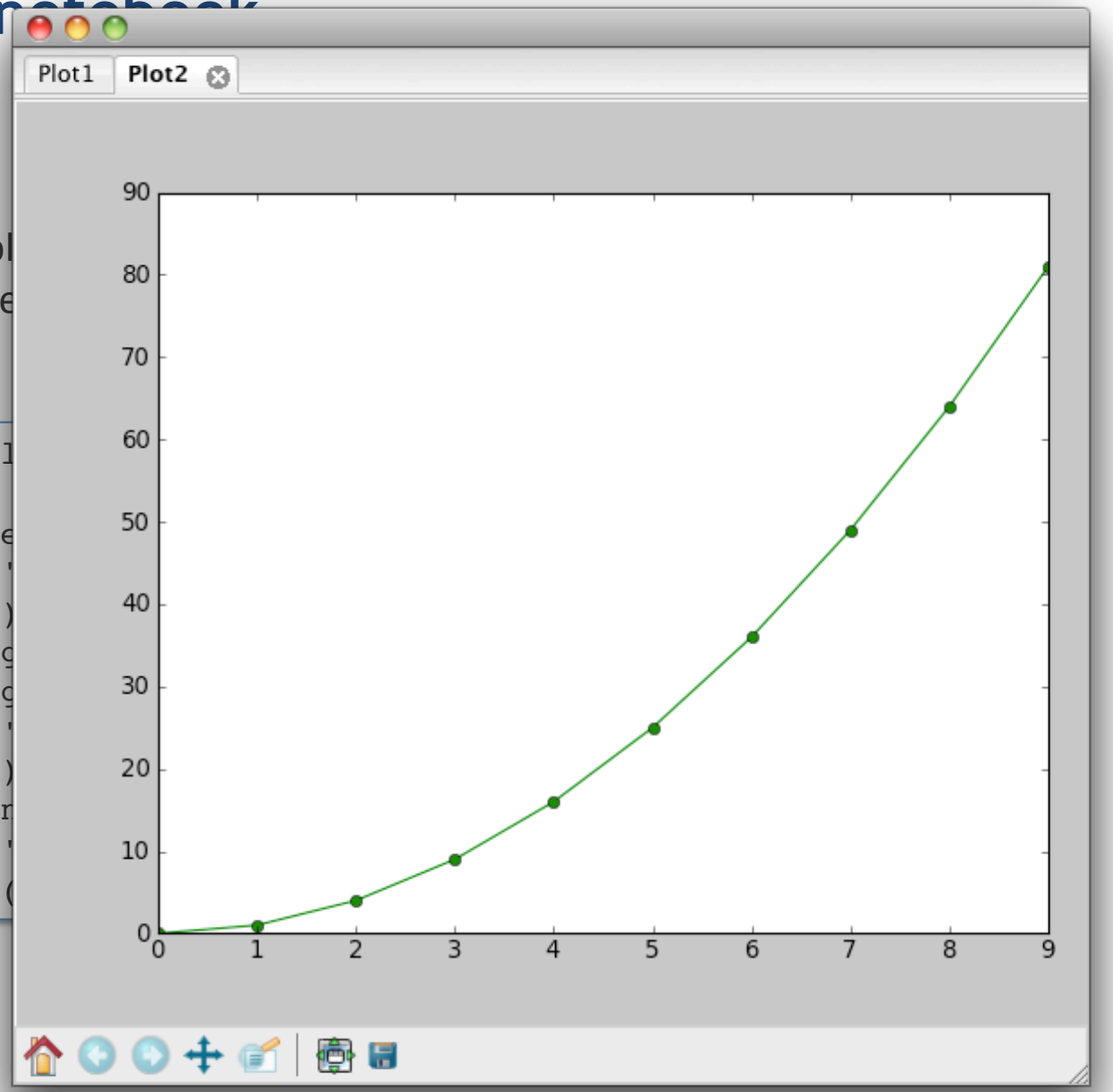
Routines:

- Initialization is done when the module is imported.
- function MakePlotWindow() creates a plot window (returns window object).
  - <window>.AddPlotTab(name="name") creates a tab in the plot window, returns the plot object
- function ShowPlots() must be called as the last line in the script.

Using the plotnotebook

- Below is some sampl
  two tabs. In a later le
  to the plots.

```
# two tab demo using pl
import plotnotebook
win = plotnotebook.Make
tab1 = win.AddPlotTab('
ax1 = tab1.figure.gca()
ax1.plot(range(10),rang
ax1.plot(range(10),rang
tab2 = win.AddPlotTab('
ax2 = tab2.figure.gca()
x2 = [i**2 for i in ran
ax2.plot(range(10),x2,'
plotnotebook.ShowPlots(
```



How does plotnotebook work?

The code that creates plot windows and tabs will make more sense after we have covered creating classes (objects), but ignoring how the PlotNotebook routine in the module actually works, we can see four parts to this module:

1. imports: pull in the modules we will need
2. create a wx applications and initialize a global variable (framelist)

```
# initialization code: create a basic Wx application
app = wx.PySimpleApp()
framelist = [] # list of frames (windows) created here
```

3. When MakePlotWindow is called (slightly simplified below) framelist saves the name(s) of the created window objects for later use

```
def MakePlotWindow(DisableDelete=False, size=(600,600)):
    'Create a plot window ...'
    plotter = PlotNotebook(id, DisableDelete)
    global framelist
    framelist.append(plotter.frame)
    return plotter
```

How does plotnotebook work? final section

4. The final section of code calls a wxPython function for each window that has been created and then starts the wxPython event loop. By burying the wx stuff in the module, the user of the plotnotebook module does not need to know anything about it or how it works.

```
def ShowPlots():
    '''Call ShowPlots after all plots are defined... '''
    for frame in framelist:
        frame.Show()
    app.MainLoop()
```