L03_Matplotlib

December 15, 2015

1 Matplotlib

Kevin Stratford kevin@epcc.ed.ac.uk Emmanouil Farsarakis farsarakis@epcc.ed.ac.uk
Other course authors: Neelofer Banglawala Andy Turner Arno Proeme

[Matplotlib] What is matplotlib? I
Matplotlib is a plotting library for Python
"make the easy things easy and the hard things possible"

- Capable of:
- interactive and non-interactive plotting
- Producing publication-quality figures
- Large amount of functionality:
- scientific and statistical plots, heatmaps
- surfaces, map-based plotting and more...

[Matplotlib] What is matplotlib? II
Matplotlib is Closely integrated with NumPy

- Use numpy functions for reading data
- As data is in numpy, matplotlib can plot it easily
- Documentation:
- http://matplotlib.org/

[Matplotlib] What is matplolib? III

- People often want to have a quick look at data in a plain text file
- Gnuplot/Excel often used for this but matplotlib can provide a simple, feature-rich replacement
- Manipulate data interactively and replot
- Can save the session to keep record of what you did if required

Creating high-quality plots is easy in matplotlib!

- 2D plotting only (VTK for 3D plots)
- grew out of MATLAB

- mostly pure Python but makes heavy use of numpy and is efficient with very large arrays...
- requirements for developer when looking for a plotting package (can I think of other requirements?)– Plots should look great - publication quality. One important requirement for me is that the text looks good (antialiased, etc.)
- Postscript output for inclusion with TeX documents
- Embeddable in a graphical user interface for application development
- Code should be easy enough that I can understand it and extend it
- Making plots should be easy

- fig.show() <– when to use this? in scripts only?

- launching an IPython shell with the `--pylab` option tells matplotlib to plot figures to the screen i.e. activates "interactive" plotting mode

matplotlib –> pyplot Explain pylab = pyplot + numpy, to create convenient environment for interactive plotting BUT no longer recommended as this is like : from pylab import * and universal imports are discouraged. So either

ipython –matplotlib : who, %who –> interactive namespace empty for both ipython –pylab : %who –> interactive namespace empty, who –>

fig = figure() –> don't need this line in ipython generally, but will if using ARCHER. figure will be generated by default, this gives a handle to figure, and helps distinguish when you have more than one figure plt.plot()

- test plot_this.py on ARCHER... see with and without X server, does script try to produce a figure? Why would we need to use Agg? Perhaps we don't when running non-interactively? Check... To support all of these use cases, matplotlib can target different outputs, and each of these capabilities is called a backend; the "frontend" is the user facing code, ie the plotting code, whereas the "backend" does all the hard work behind-the-scenes to make the figure. There are two types of backends: user interface backends (for use in pygtk, wxpython, tkinter, qt, macosx, or fltk; also referred to as "interactive backends") and hardcopy backends to make image files (PNG, SVG, PDF, PS; also referred to as "non-interactive backends")

MAKE CLEAR some backends suppress need for diplay (don't confuse with 'interactive')?

When in non-interactive mode, figures don't display anyway. Calling Agg ensures nice rendering of figure, don't need to suppress X server display? BE CLEAR why do you need "use("Agg") in a script???

[Matplotlib] Basic concepts I

- Everything is assembled by Python commands

- Create a figure with an axes area (this is the plotting area)

- Can create multiple plots in one figure

[Matplotlib] Basic concepts II

- Only one figure (or axes) is active at a given time (i.e. current figure, current axes)

- In an IPython shell, can plot to the screen (interactive mode) or save to image (non-interactive mode)

- Can use the show() command in, for example, a Python script to display the plot

matplotlib.pyplot contains the high-level functions we need to do all the above and more

Notes * Matplotlib is the whole package; matplotlib.pyplot is a module in matplotlib; and pylab is a module that gets installed alongside matplotlib.

- Pyplot provides the state-machine interface to the underlying object-oriented plotting library. The state-machine implicitly and automatically creates figures and axes to achieve the desired plot. For example

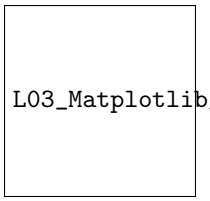
- pylab is a convenience module that bulk imports matplotlib.pyplot (for plotting) and numpy (for mathematics and working with arrays) in a single name space. Although many examples use pylab, it is no longer recommended.

[Matplotlib] Basic plotting
Launch an IPython shell, import pyplot and numpy
Notes 9. explain "%matplotlib" and say that you will explain later what this does 1. basic plot, line 2. use symbols (change pts), different colour 3. add x again with different line colour 4. show fig = plt.figure() –> plt.title('A Title'); 5. then plt.plot() again 6. add second plot to plt.plot()

```
In [3]: # add 'inline' option if using a notebook
        %matplotlib inline
        import matplotlib.pyplot as plt; import numpy as np

In [4]: xmin=0; xmax=10; pts = 50;
        x = np.linspace(xmin, xmax, pts);
        y = np.cos(x);

In [5]: # line, markers, 2 plots, fig, title then plot
        plt.plot(x,y,'ro'); # , x, y, 'g-'# #
```



[Matplotlib] Saving images to file

- Saving to image file is simple using savefig
- File format is determined from the extension you supply
- Resolution set using dpi option
- Commonly supports: png, jpg, pdf, ps

Note 1. Students to do basic plotting exercise (up to saving image). Give them 10-15 mins to do this

```
In [ ]: # save image to file in different formats
        plt.savefig("cos_plot.pdf");
        plt.savefig("cos_plot.png", dpi=300);  # higher resolution (dpi)
```

Time to create some plots. Please complete Basic Plotting (pages 1 - 7) of the Matplotlib exercise.
[Matplotlib] What is a backend? (++) I
Matplotlib consists of two parts, a frontend and a backend:

- Frontend : the user facing code i.e the plotting code
- Backend : does all the hard work behind-the-scenes to make the figure

By offering different backends, Matplotlib can support a wide range of different use cases and output formats.
[Matplotlib] What is a backend? (++) II
There are two types of backend:

- User interface, or "interactive", backends
- Hardcopy, or "non-interactive", backends to make image files
- e.g. Agg (png), Cairo (svg), PDF (pdf), PS (eps, ps)
- These are known as rendering engines and determine how your image is drawn

[Matplotlib] What is a backend? (++) III

- Check which backend is being used with: matplotlib.get_backend()
- Default backend on ARCHER is Qt4Agg
- Switch to a different backend with matplotlib.use(...)
- Must issue command before importing matplotlib.pylot (or %matplotlib)

[Matplotlib] What is interactive mode? (++) I
Here we mean that a figure displays to screen as soon as you call either plt.figure() or plt.plot().

- Furthermore, the displayed figure does not prevent you from issuing commands in the IPython shell. This means you can update the figure and see the resulting changes immediately.

[Matplotlib] What is interactive mode? (++) II
In contrast, in "non-interactive" mode, the figure will not display to screen, unless you call show(). This is what happens when you create figures in scripts.

- If you show the figure, it "blocks" any further commands being issued in the shell until you have to closed the figure.

[Matplotlib] What is interactive mode? (++) III
To confuse matters, an "interactive" backend does not guarantee your figures will automatically display to screen. Matplotlib has a Boolean variable in its configuration file (the matplotlibrc files, more of that later) that sets the interactivity.

- You can query this with: matplotlib.is_interactive()

[Matplotlib] What is interactive mode? (++) IV
In most cases you don't need to worry about this.
The easiest way to ensure interactivity is to either:

- launch an IPython shell with the –matplotlib option or
- to issue the magic command %matplotlib within the IPython shell before issuing any other command.

Notes

- Actually, interactivity (plotting to screen) not guaranted by a particular backend. It is set by matplotlibrc variable

What is interactive mode? * Use of an interactive backend (see What is a backend?) permits–but does not by itself require or ensure–plotting to the screen. Whether and when plotting to the screen occurs, and whether a script or shell session continues after a plot is drawn on the screen, depends on the functions and methods that are called, and on a state variable that determines whether matplotlib is in "interactive mode". The default Boolean value is set by the matplotlibrc file, and may be customized like any other configuration parameter (see Customizing matplotlib). It may also be set via matplotlib.interactive(), and its value may be queried via matplotlib.is_interactive(). Turning interactive mode on and off in the middle of a stream of plotting commands, whether in a script or in a shell, is rarely needed and potentially confusing, so in the following we will assume all plotting is done with interactive mode either on or off.

- IPython: on ARCHER, Qt4Agg, macosx on laptop -> neither will plot to screen, need fig.show(), plt.show() by default. Remember, show is a "blocking" call, need to close window to continue. Also, fig.show() will show only what you've done to date. So need to execute all figure commands before fig.show() else you'll only see empty or part completed figure.
- setting %matplotlib on macosx, makes it interactive (still macosx backend).
- setting %matplotlib on ARCHER will also produce plots to screen BUT might have X server issues, so we will not do this... (can do this if using your own laptop)
- To make this more confusing, if using notebook, you have the interactive backend set by default so will not need to use plt.show() or fig.show()
- A lot of documentation on the website and in the mailing lists refers to the "backend" and many new users are confused by this term. matplotlib targets many different use cases and output formats. Some people use matplotlib interactively from the python shell and have plotting windows pop up when they type commands. Some people embed matplotlib into graphical user interfaces like wxpython or pygtk to build rich applications. Others use matplotlib in batch scripts to generate postscript images from some numerical simulations, and still others in web application servers to dynamically serve up graphs.
- To support all of these use cases, matplotlib can target different outputs, and each of these capabilities is called a backend; the "frontend" is the user facing code, i.e., the plotting code, whereas the "backend" does all the hard work behind-the-scenes to make the figure. There are two types of backends: user interface backends (for use in pygtk, wxpython, tkinter, qt4, or macosx; also referred to as "interactive backends") and hardcopy backends to make image files (PNG, SVG, PDF, PS; also referred to as "non-interactive backends")
- To make things a little more customizable for graphical user interfaces, matplotlib separates the concept of the renderer (the thing that actually does the drawing) from the canvas (the place where the drawing goes). The canonical renderer for user interfaces is Agg which uses the Anti-Grain Geometry C++ library to make a raster (pixel) image of the figure. All of the user interfaces except macosx can be used with agg rendering, e.g., WXAgg, GTKAgg, QT4Agg, TkAgg. In addition, some of the user interfaces support other rendering engines. For example, with GTK, you can also select GDK rendering (backend GTK) or Cairo rendering (backend GTKCairo).
- need to explain interactive mode and backend mode, e.g. Agg
- got interactive (shell/notebook) and non-interactive (script)
- but also mean interactive in the sense that figure prints to screen
- be default this is the case for shell, not for non-interactive
- to switch it off in shell, set backend to non-interactive i.e. save to file and can choose formats
- to switch it on in non-interactive script, savefig FIRST then do fig.show() or plt.show() AFTER savefig (if you are saving figure)
- cannot call non-interactive backend if: matplotlib.pyplot already been called, must be before this!
- can call %matplotlib at any point after you have set non-interactive backend to make shell interactive again

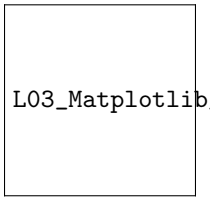
[Matplotlib] Plot customisations I

There are many ways to customise a plot. Play with the following properties.

Notes 0. Should explain difference between certain routines in figure, pylot, axes etc.? 1. Students to play about with plot properties. Give them 15 - 20 mins to do this

```
In [6]: # Ex: set the figure size and add a plot
        fig=plt.figure(figsize=(4,4));
        plt.plot(x,y,'c-')
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x109e666d0>]
```




```
In [ ]: # Ex: linewidth, and
        # linestyles: '-', '.-', ':', '--'
        plt.plot(x,y,'k-',linewidth=2.0)
```

[Matplotlib] Plot customisations II
Play around with plot markers.

```
In [ ]: # Ex: markers and their properties
        # unfilled markers: '.',+','x','1' to '4','|'
        plt.plot(x,y,'x',markersize=10)
```

```
In [ ]: # filled markers: 'o', 's','*','d','>','^','v', 'p', 'h'
        plt.plot(x,y,'8',markerfacecolor='None',markeredgecolor='g',
                 markersize=10)
```

[Matplotlib] Plot customisations III
Set x-axis and y-axis limits

```
In [ ]: # Ex: x,y, axis limits:
        plt.xlim((xmax*0.25,xmax*0.75));
        plt.ylim((np.cos(xmin*0.25),np.cos(xmax*0.75)));
        plt.plot(x,y,'mo-')
```

[Matplotlib] Plot customisations IV
Adjust title font properties

```
In [ ]: # Ex: title placement and font properties
        plt.plot(x,y,'x')
        plt.suptitle('A Centered Title', fontsize=20)
        # loc: center, left, right
        # verticalalignment: center, top, bottom, baseline
        plt.title('A Placed Title', loc='left', verticalalignment='top' )
```

[Matplotlib] Plot customisations V
Add tickmarks

```
In [ ]: # Ex: tick marks
        fig=plt.figure(figsize=(4,3.5)); plt.plot(x,y,'x');
        nticks = 5;
        tickpos = np.linspace(xmin,xmax,nticks);
        labels = np.repeat(['tick'],nticks);
        plt.xticks(tickpos, labels, rotation='vertical');
```

[Matplotlib] Plot customisations VI
Add annotations

```
In [ ]: # Ex++: arrows and annotations
        plt.plot(x,y,'x');
        atext='annotate this'; arrowtip=(1.5,0.5); textloc=(3, 0.75);
        plt.annotate(atext, xy=arrowtip, xytext=textloc,
                     arrowprops=dict(facecolor='black', shrink=0.01),)
```

[Matplotlib] Subplots

- There can be multiple plots, or subplots, within a figure
- Use subplot(nrows, ncols, plot number) to place plots on a regular grid
- The most recently created subplot is the current plot

[Matplotlib] Subplots II

- Can move between subplots by creating each subplot with a "handle" for each axes

```
In [ ]: (fig, axes) = plt.subplots(nrows=2, ncols=2);
        axes.size

        axes[0,0].plot(x,y,'g-');
        axes[1,1].plot(x,y,'r-');
```

[Matplotlib] subplot2grid (++)

- For more control over subplot layout, use subplot2grid
- Subplots can span more than one row or column

```
In [ ]: # Ex++: subplot2grid(shape, loc, rowspan=1, colspan=1)
        fig = plt.figure()
        ax1 = plt.subplot2grid((3, 3), (0, 0)); ax1.plot(x,y,'r-');
        ax2 = plt.subplot2grid((3, 3), (0, 1), colspan=2); ax2.plot(x,y,'g-');
        ax3 = plt.subplot2grid((3, 3), (1, 0), colspan=2, rowspan=2); ax3.plot(x,y,'b-');
        ax4 = plt.subplot2grid((3, 3), (1, 2), rowspan=2); ax4.plot(x,y,'c-');
```

[Matplotlib] Customise some subplots

- Go back to the Matplotlib exercise and create multiple customised plots (pages 8 - 11)

Note 1. Students to create a customised plot with multiple plots, give them 15 - 20 mins to do this
[Matplotlib] Other type of plots
http://matplotlib.org/gallery.html
Notes * Only just skimmed the surface of what is possible * the above slide looks terrible but it looks reasonable in a slideshow... promise... * xkcd figure in the bottom right corner...
[Matplotlib] Advanced : animation
Can even create animations (from Nicolas P. Rougier, https://github.com/rougier)

```
In [ ]: %reset
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [ ]: #from matplotlib import use
        ## animation doesn't work with macosx backend!
        #use("nbagg")
        #import earthquakes;
```

[Matplotlib] Images for publication I

- Matplotlib uses matplotlibrc configuration files to customize and set defaults for all kinds of properties (rc settings, rc parameters)
- Creating a custom matplotlibrc file in your local directory will override the default matplotlibrc file, and limit changes to that directory.

From Damon McDougall: http://bit.ly/1jIuuU0
[Matplotlib] Images for publication II

- You will most likely want different settings for each journal
- useful to keep a different matplotlibrc file for each journal

[Matplotlib] matplotlibrc : import rc file

- import a particular settings file with:

from matplotlib import rc_file rc_file('/path/to/my/matplotlibrc')
[Matplotlib] matplotlibrc : settings I
axes.labelsize : 9.0 # fontsize of the x any y labels xtick.labelsize : 9.0 # fontsize of the tick labels ytick.labelsize : 9.0 # fontsize of the tick labels legend.fontsize : 9.0 # fontsize in legend font.family : serif font.serif : Computer Modern Roman Marker size : lines.markersize : 3 text.usetex : True

- Last line means use TeX to format all text (only available with Agg, PS, PDF backends)

Notes * set font properties etc.
[Matplotlib] matplotlibrc : settings II
Here are some settings you can use to create a nice figure ratio
WIDTH = 500.0 # Figure width in pt (usually from LaTeX) FACTOR = 0.45 # Fraction of the width you'd like the figure to use widthpt = WIDTH * FACTOR inperpt = 1.0 / 72.27 # use the Golden ratio because it looks good golden_ratio = (np.sqrt(5) - 1.0) / 2.0 widthin = widthpt * inperpt heightin = widthin * golden_ratio figdims = [widthin, heightin] # Dimensions as list fig = plt.figure(figsize=figdims)
Notes * students will see what 'nice' means when they do the exercise
[Matplotlib] Include images in LaTeX
When you include the figure in the LaTeX source you should specify the scale factor as the width:

Complete the Matplotlib exercise and create a publication standard image (pages 12 - 13)
Note 1. Students to create publication image. Give students 15 - 20 mins to do this
Notes * some notes
[Matplotlib] Summary I

- Simple, interactive plotting
- integration with NumPy allows you to easily read data
- Plotting syntax is simple and concise
- Complex plotting types also available
- Can start from code for simple plots
- Many examples available online

[Matplotlib] Summary II

- Producing publication-ready images is relatively simple
- Easily customised for different scenarios
- The more you use matplotlib, the more you get out of it!
- Other packages
- Bokeh : interactive visualisation library.