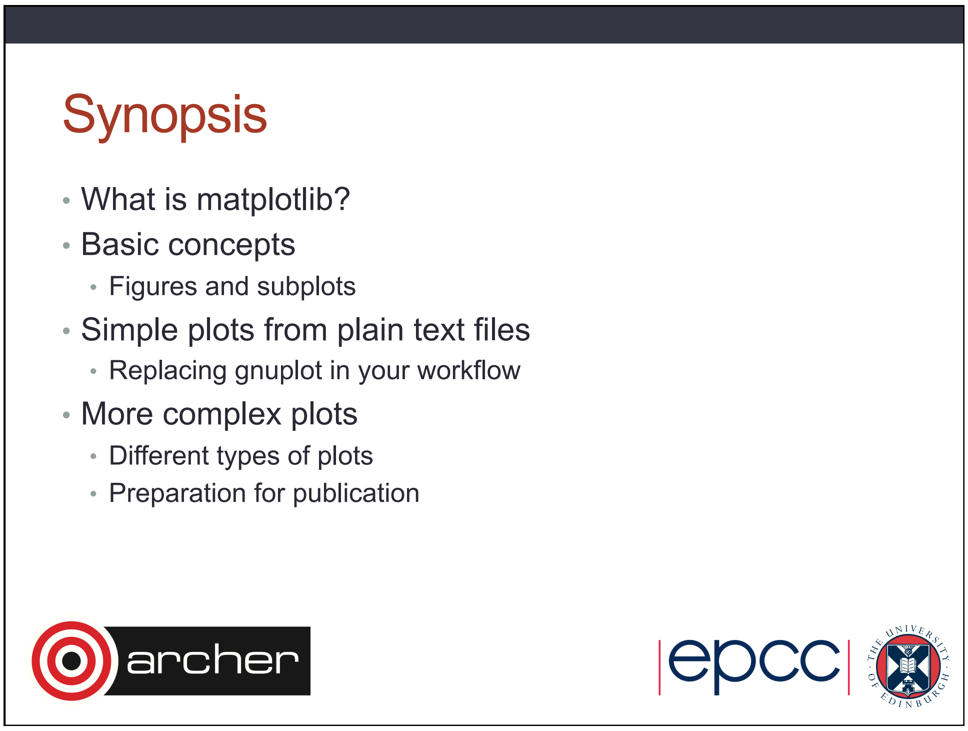
matplotlib

Andy Turner, ARCHER CSE Team
a.turner@epcc.ed.ac.uk

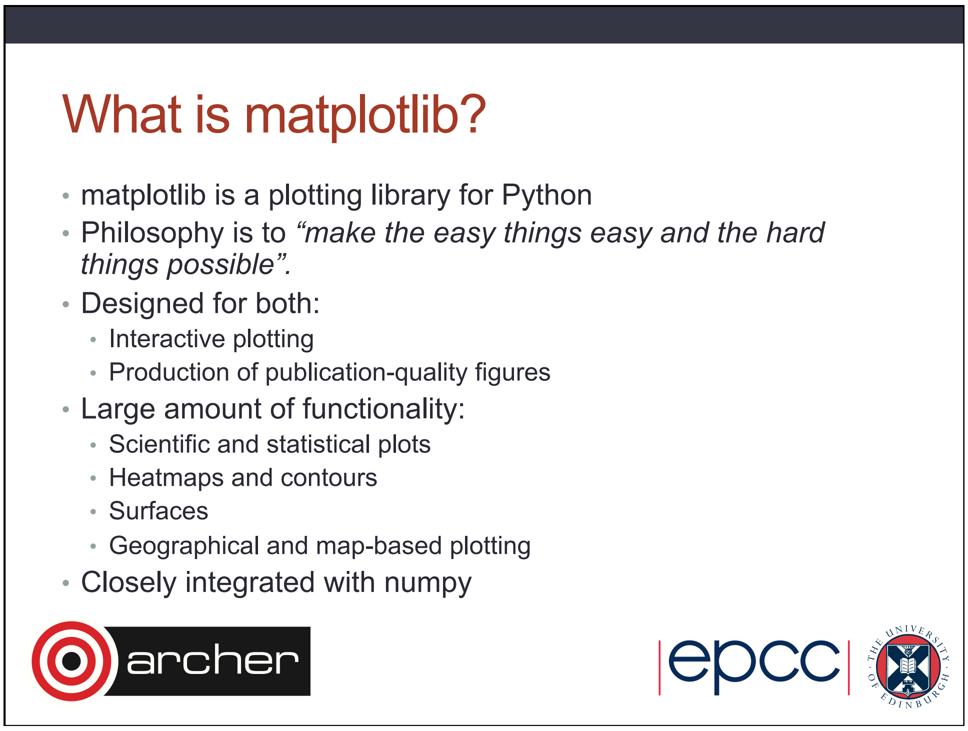
Reusing this material

http://www.archer.ac.uk
support@archer.ac.uk



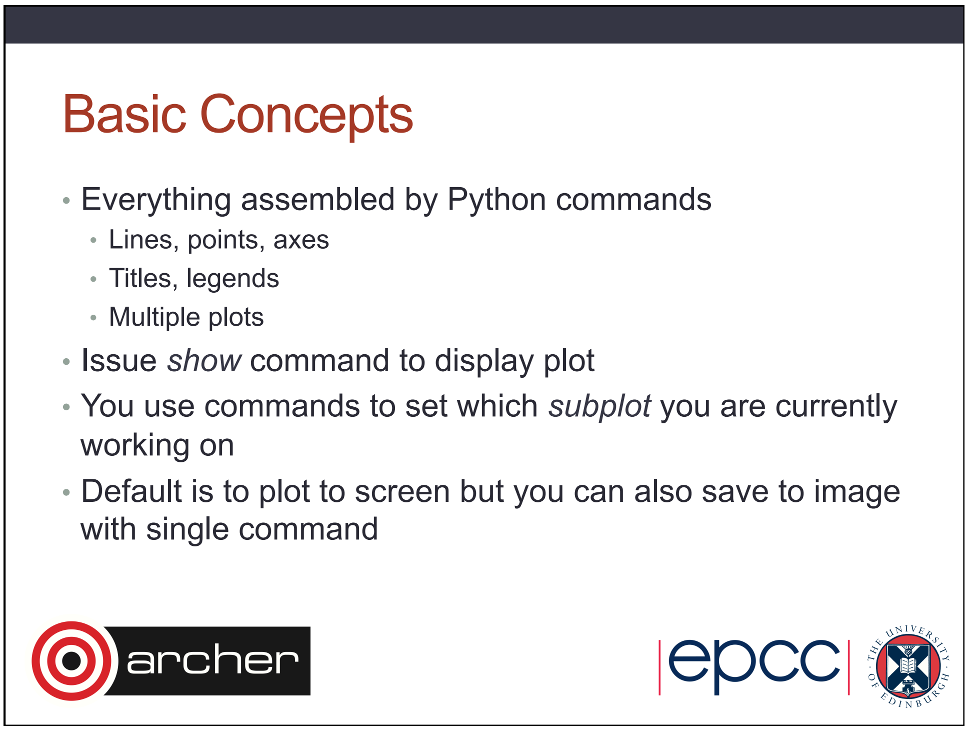
Synopsis

- What is matplotlib?
- Basic concepts
  - Figures and subplots
- Simple plots from plain text files
  - Replacing gnuplot in your workflow
- More complex plots
  - Different types of plots
  - Preparation for publication



What is matplotlib?

- matplotlib is a plotting library for Python
- Philosophy is to *"make the easy things easy and the hard things possible".*
- Designed for both:
  - Interactive plotting
  - Production of publication-quality figures
- Large amount of functionality:
  - Scientific and statistical plots
  - Heatmaps and contours
  - Surfaces
  - Geographical and map-based plotting
- Closely integrated with numpy



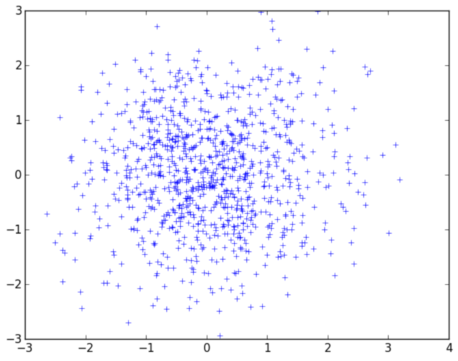
Basic Concepts

- Everything assembled by Python commands
  - Lines, points, axes
  - Titles, legends
  - Multiple plots
- Issue *show* command to display plot
- You use commands to set which *subplot* you are currently working on
- Default is to plot to screen but you can also save to image with single command

Example: random scatterplot

- Assuming we are using ipython –pylab:

```
x = randn(1000)
y = randn(1000)
plot(x, y, '+')
```



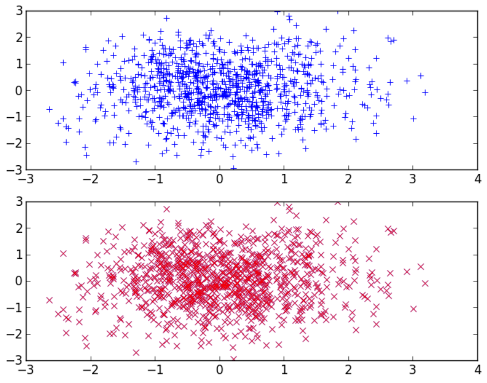
Figures and subplots

- The whole plotting area is known as the *figure*
- Within the figure there can be *subplots*
  - subplots are placed on a regular grid within the figure
  - (If you need more control over placement you can use *axes*)
  - For simple plots there is usually only one subplot (1, 1, 1)
- You use the `subplot` command to specify which subplot you are currently working on
  - `subplot(nrows, ncols, plot number)`

Example: random scatterplots

- Assuming we are using ipython –pylab:

```
x = randn(1000)
y = randn(1000)
fig = figure()
subplot(2, 1, 1)
plot(x, y, ‘b+’)
subplot(2, 1, 2)
plot(x, y, ‘rx’)
fig.show()
```



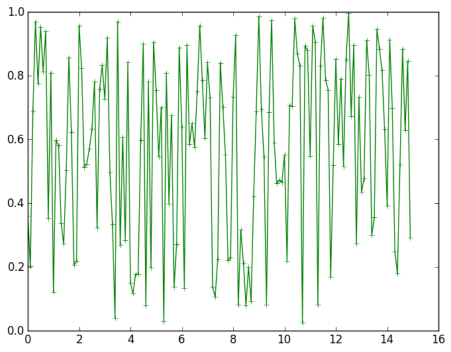
Simple plots from plain text

- 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 the data interactively and replot
  - Can save the session to keep record of what you did if required
- Use numpy functions for reading data
  - Simple interface to complex reading if required
- As data is in numpy, matplotlib can plot it easily

Example: read and plot x, y data

- Assuming we are using ipython –pylab:

```
data = genfromtxt(‘random1.dat’)
fig = figure()
subplot(1, 1, 1)
plot(data[:,0], data[:1], ‘g+-’)
fig.show()
```



Setting axis labels, titles and legends

- Axis labels: use *xlabel* and *ylabel* (they act on the currently selected subplot)*:*
  - `xlabel(“Job Size”)`
- Title: use fig.suptitle:
  - `fig.suptitle(“Job Size Distribution on ARCHER”)`
- Legend: use legend (acts on the currently selected subplot):
  - Requires that label is set for plot:

```
plot(jobs[:,0], jobs[:,1], ‘r–’, label=“2014”)
legend()
```

Save to image file

- Saving to image file is simple using *fig.savefig*
  - File format is determined from the extension
  - e.g. to save to a PNG image:

```
fig.savefig(“archer_jobs.png”)
```

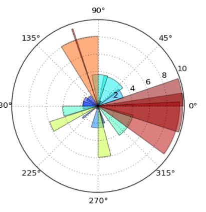
- Resolution set using *dpi* option:
  - e.g:

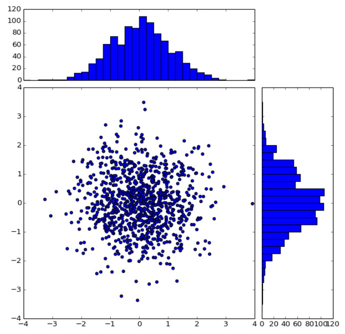
```
fig.savefig(“archer_jobs.png”, dpi=300)
```

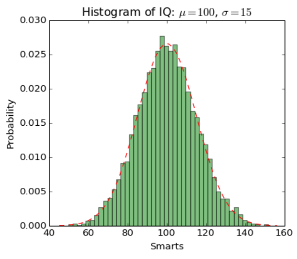
- Commonly supports: png, jpg, pdf, ps

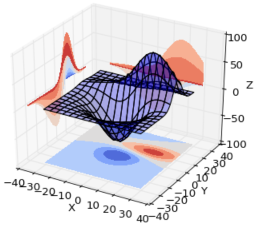
Other types of plots

- http://matplotlib.org/gallery.html









Preparing publication images

- You will probably want different settings for each journal
- matplotlib uses a settings file: *matplotlibrc*, to setup font sizes, types and plotting defaults
  - Useful to keep a different *matplotlibrc* file for each journal
- Import a particular settings file with:

```
from matplotlib import rc_file
rc_file('/path/to/my/matplotlibrc')
```

- From Damon McDougall: http://bit.ly/1jIuuU0

Useful matplotlibrc font settings

```
# Font sizes and types
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

# Use TeX to format all text
text.usetex : True
```

Setting 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 occupy
widthpt  = WIDTH * FACTOR

inperpt = 1.0 / 72.27
golden_ratio  = (np.sqrt(5) - 1.0) / 2.0  # because it looks good

widthin  = widthpt * inperpt
heightin = widthin * golden_ratio
figdims    = [widthin, heightin] # Dimensions as list

fig = plt.figure(figsize=figdims)
```

Setting a nice figure ratio (cont.)

- When you include in the LaTeX source you should specify the scale factor as the width:

```
\begin{figure}
\includegraphics[width=0.45\textwidth]{figure.pdf}
\end{figure}
```

Eliminate unnecessary whitespace

- Eliminate the whitespace with:

```
fig.tight_layout(pad=0.1)
```

- Finally, save your figure in a useful format:

```
fig.savefig('plot.pdf', dpi=600)
```

Summary

- Simple ,interactive plotting:
  - 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
- Producing publication-ready images is relatively simple
  - Easily customised for different scenarios
- The more you use matplotlib, the more you get out of it!