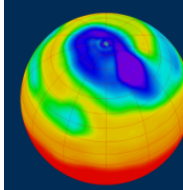




**National Centre for  
Atmospheric Science**  
NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental  
Data Analysis**  
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL  
NATURAL ENVIRONMENT RESEARCH COUNCIL

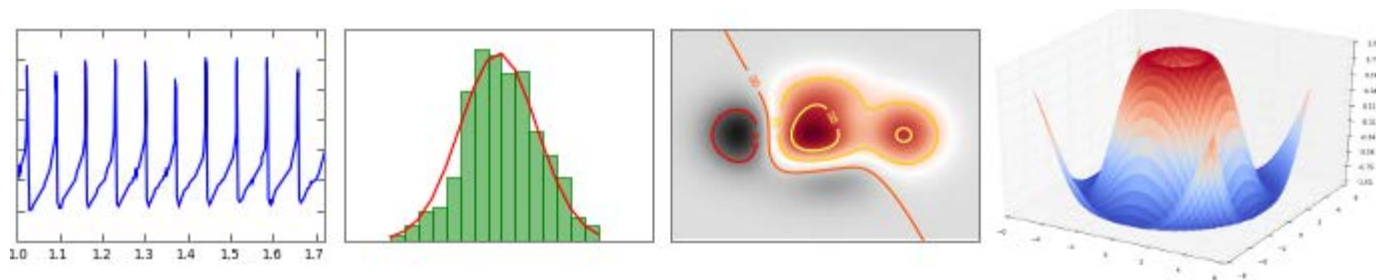
# Visualisation in python (with Matplotlib)

Thanks to all contributors:

Ag Stephens, Stephen Pascoe.

# Introducing Matplotlib

Matplotlib is a python **2D plotting library** which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in python scripts, the python shell, web application servers, and six graphical user interface toolkits.



# Introducing Matplotlib

Matplotlib enables you to generate **plots, histograms, power spectra, bar charts, error charts, scatterplots**, etc, with just a few lines of code.

For simple plotting the "pyplot" interface provides a **MATLAB-like interface**.

You also have full control of *line styles, font properties, axes properties*, etc, via an **object oriented interface** or via a set of functions familiar to MATLAB users.

# Recommending Matplotlib

As with all open source Python tools there are other options and approaches available.

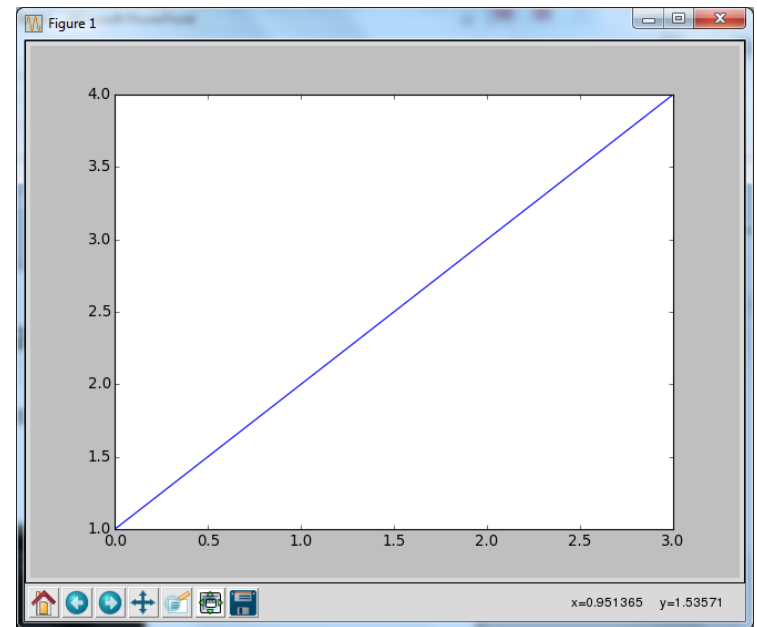
However, Matplotlib, like NumPy, has become the **clear leader** in its particular niche.

If you want to do (high quality) visualisation in Python – use Matplotlib!

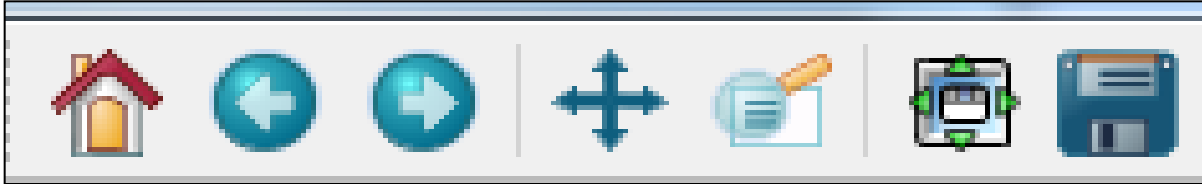
# Using Matplotlib Interactively

- Matplotlib has its own interactive plotting window:

```
1. /home/mobaxterm 2. astephen@jasmin-sci1 3. /home/mobaxterm
[astephen@jasmin-sci1 ~]$ python2.7
Python 2.7.3 (default, Feb 21 2014, 13:11:38)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more info
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3,4])
>>> plt.show()
[<matplotlib.lines.Line2D object at 0x2944710>]
>>> plt.show()
```



# Using Matplotlib Interactively



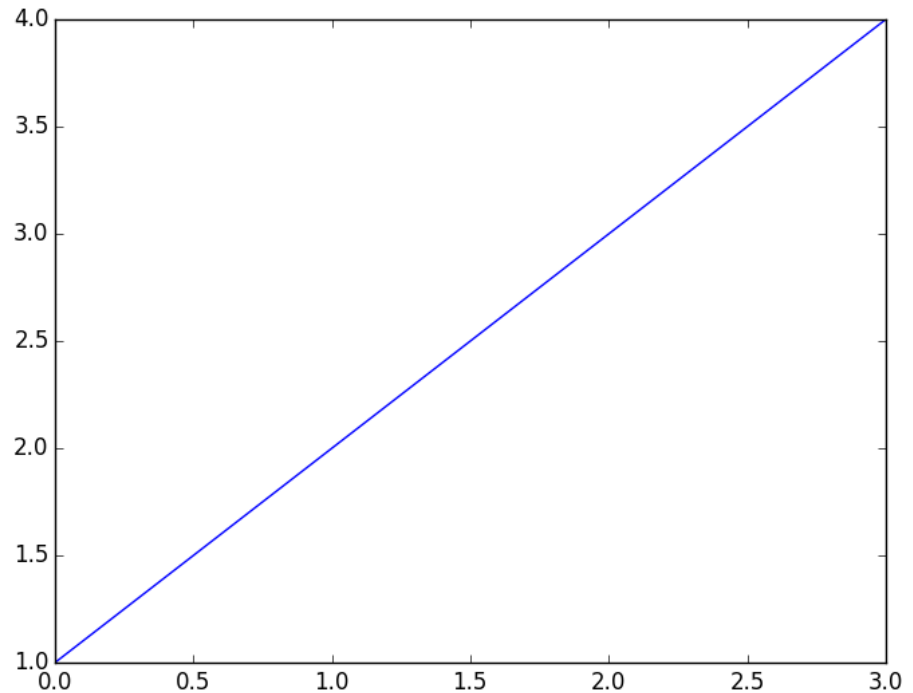
The buttons allow you to:

- Re-set the image
- Move between different plots in this session
- Scroll around the current plot
- Zoom in to specified region
- View whole plot
- Save the plot

# The first plot: A simple line graph

```
import matplotlib.pyplot as plt  
plt.plot([1,2,3,4])  
plt.show()
```

Defaults are used for things you do not specify (such as the x-axis values).



!!!This slide is important!!!

## Before we go any further...

In the environment you are working with it is useful to be able to work in Python without closing your interactive window. **Here's how you do it:**

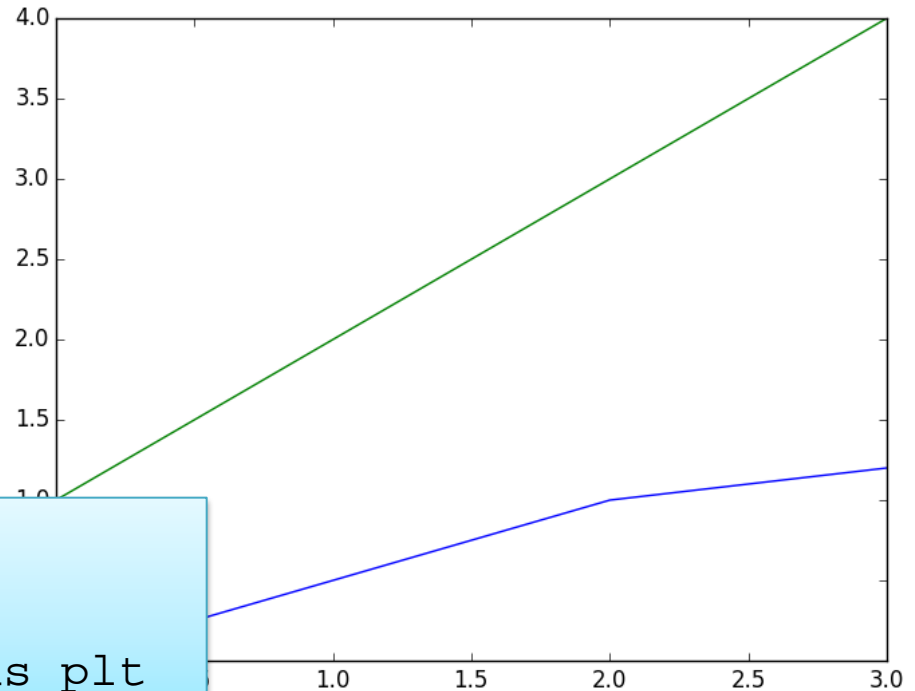
```
>>> import matplotlib.pyplot as plt # The standard import
>>> plt.plot(range(5))               # Generate a plot
>>> plt.pause(1)                     # Show plot but refocus
                                     # on python shell
# Now you can continue to modify the plot in python
>>> plt.clf()                        # Clears the figure
>>> plt.pause(1)                     # Updates (clears) canvas
>>> plt.plot([1, 10, 3, 4, 9])       # Plot something different
>>> plt.pause(1)                     # Show the plot in the
                                     # window.
```

!!!This slide is important...if a bit of a kludge!!!



# Two lines

```
plt.plot([0,0.5,1,1.2])  
plt.plot([1,2,3,4])  
plt.show()
```

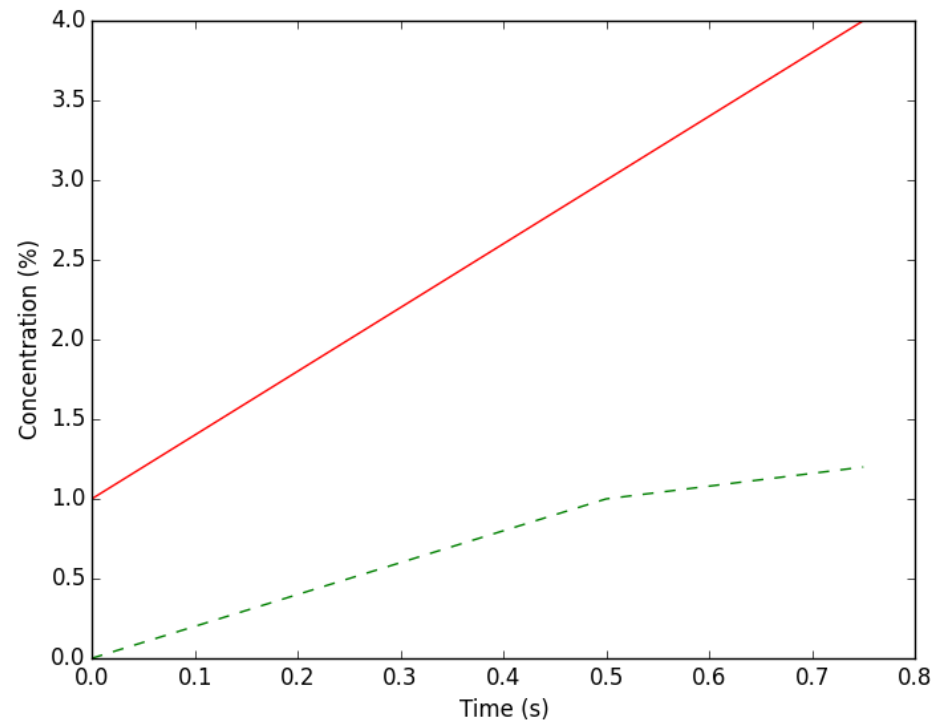


Assume we have always run:

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

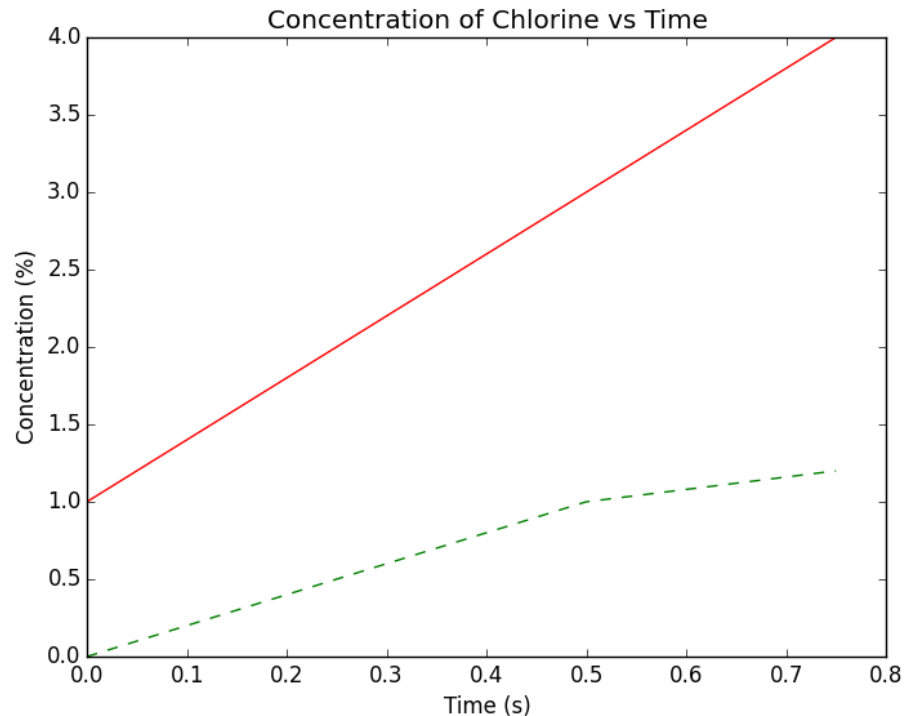
# Two lines formatted – with axes

```
times = [0, 0.25, 0.5, 0.75]
plt.plot(times, [0,0.5,1,1.2], 'g--',
         times, [1, 2, 3, 4], 'r')
plt.ylabel('Concentration (%)')
plt.xlabel('Time (s)')
plt.show()
```



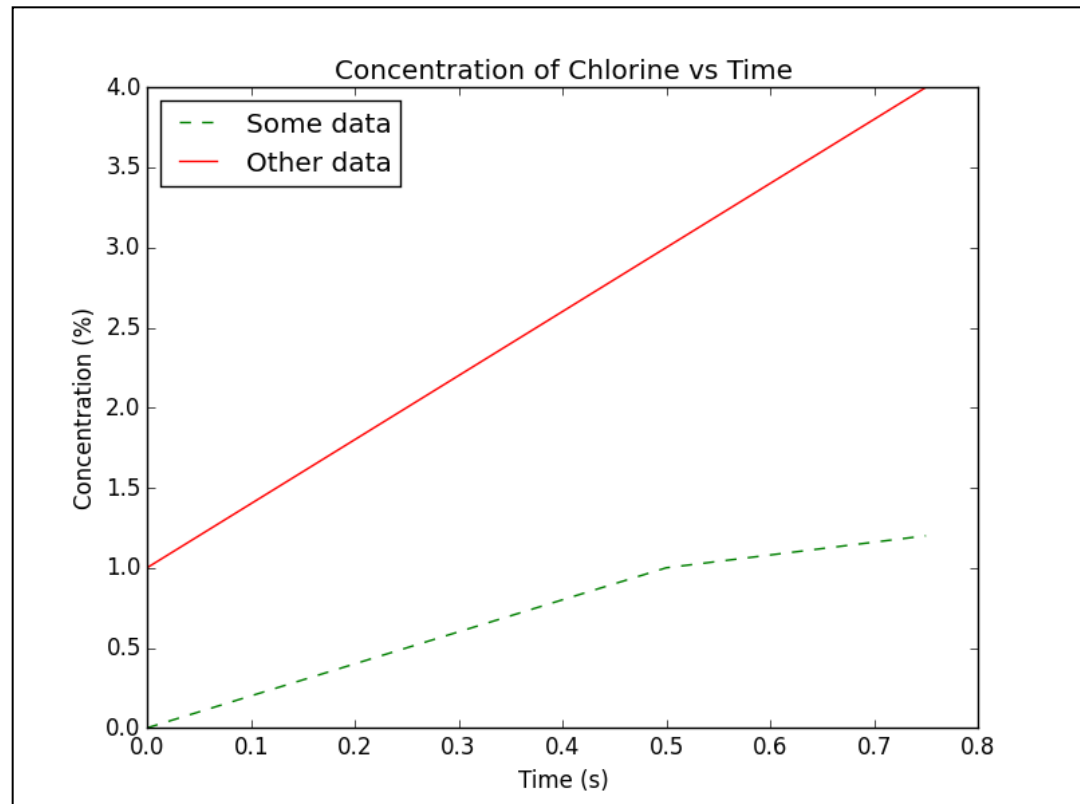
# Add a title

```
times = [0, 0.25, 0.5, 0.75]
plt.plot(times, [0,0.5,1,1.2], 'g--',
         times, [1, 2, 3, 4], 'r')
plt.title('Concentration of Chlorine vs Time')
plt.ylabel('Concentration (%)')
plt.xlabel('Time (s)')
plt.show()
```



# Add a legend

```
times = [0, 0.25, 0.5, 0.75]
plt.plot(times, [0,0.5,1,1.2], 'g--', label = "Some data")
plt.plot(times, [1, 2, 3, 4], 'r', label = "Other data")
plt.title('Concentration of Chlorine vs Time')
plt.ylabel('Concentration (%)')
plt.xlabel('Time (s)')
plt.legend()
plt.show()
```



# Saving an image: savefig

- To save an image use:

```
plt.savefig( "myplot.png" )
```

- Optional arguments include:
  - dpi: resolution
  - orientation: "portrait" or "landscape"
  - format: "png", "pdf", "ps", "eps" or "svg"
  - And more...

# plt.figure – To plot **multiple figures** and change **size**

- To draw multiple plots from the same session:

```
plt.figure()  
plt.plot(range(5))  
plt.figure(figsize = (10, 10)) # size in inches  
plt.plot(range(100))  
plt.show() # shows both figures
```

- plt.figure: returns a new figure so you can interact with them independently, e.g.:

```
f1 = plt.figure()  
f2 = plt.figure()
```



# Histogram – prepare the data

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

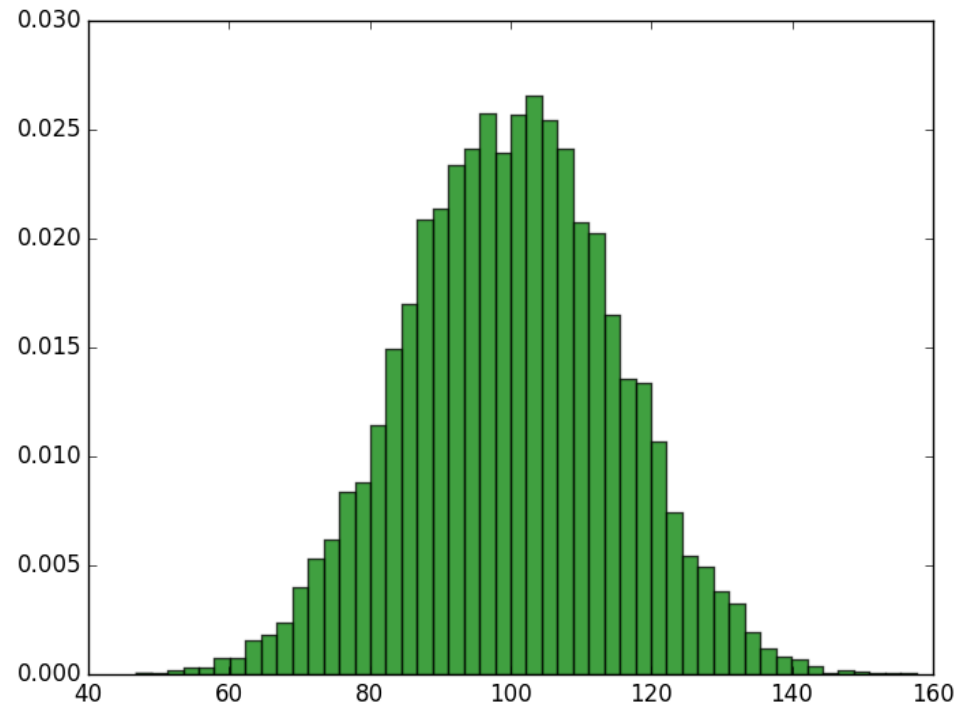
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
```

The above code will be assumed for all the following histogram examples.



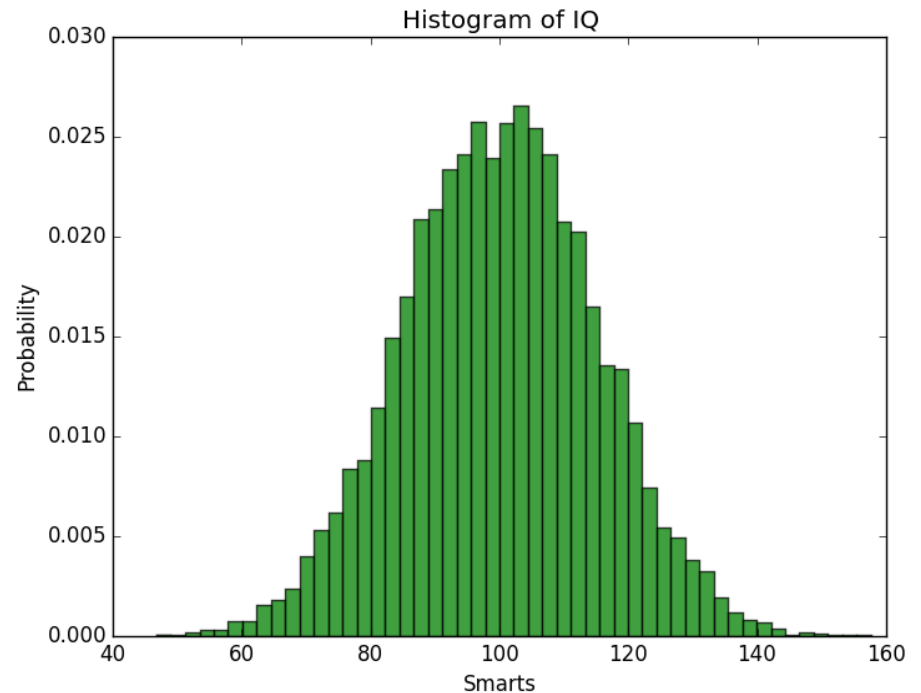
# Histogram - basic

```
n, bins, patches = plt.hist(x, 50, normed=1,  
                             facecolor='g', alpha=0.75)  
plt.axis([40, 160, 0, 0.03])  
plt.show()
```



# Histogram – annotated

```
n, bins, patches = plt.hist(x, 50, normed=1,  
                             facecolor='g', alpha=0.75)  
  
plt.xlabel('Smarts')  
plt.ylabel('Probability')  
plt.title('Histogram of IQ')  
plt.axis([40, 160, 0, 0.03])  
plt.show()
```



# Multiple plots – prepare data

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

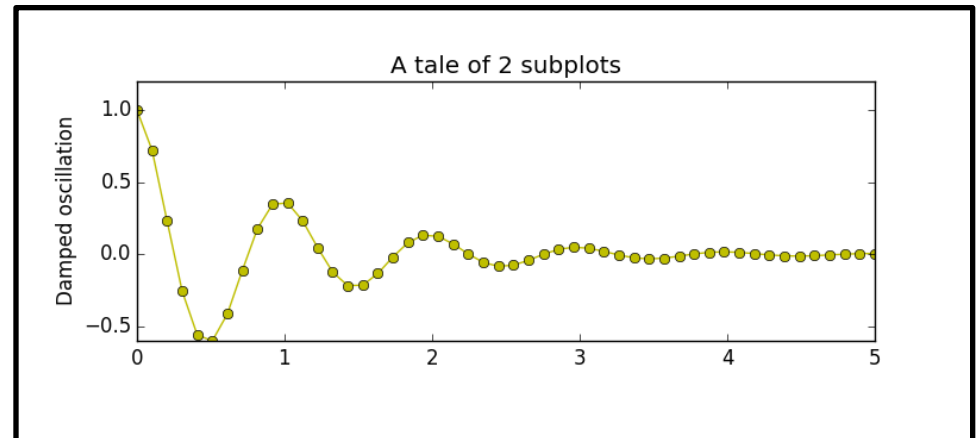
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
```

The above code will be assumed for all the following multiple plot examples.

# Multiple plots (using subplot)

```
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
```



The "subplot" function is defined as:

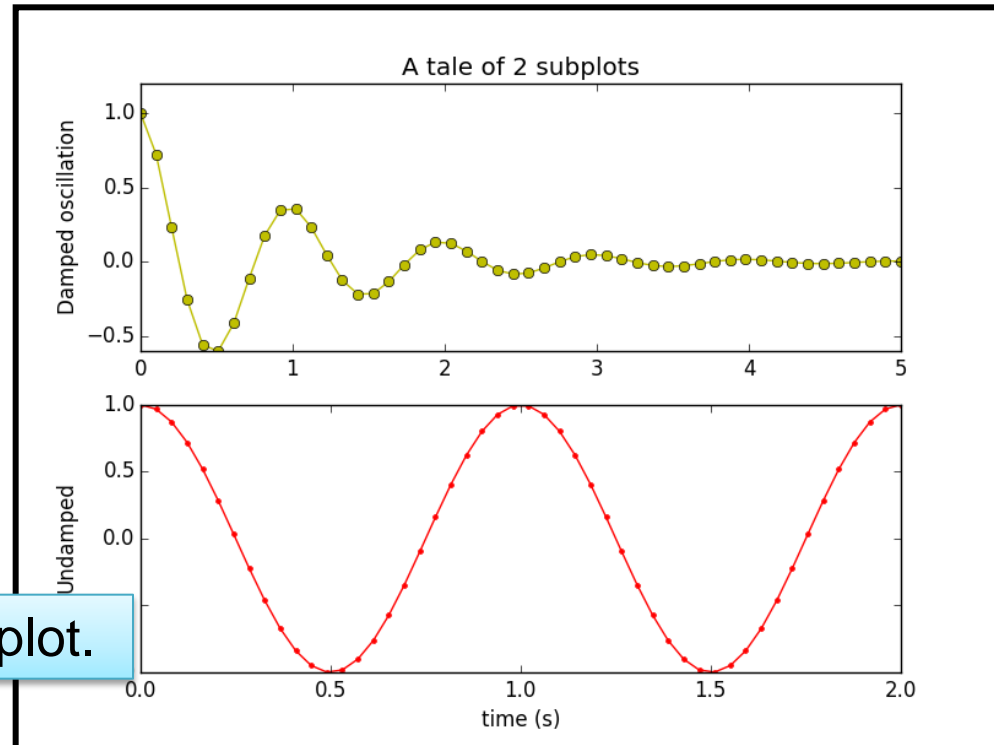
```
subplot(nrows, ncols, plot_number)
```

Here we define: 2 rows, 1 column and we add the first plot.

# Multiple plots (using subplot)

```
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
```

```
plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
plt.show()
```



Now we have added the second plot.

# Multiple axes on one plot (1)

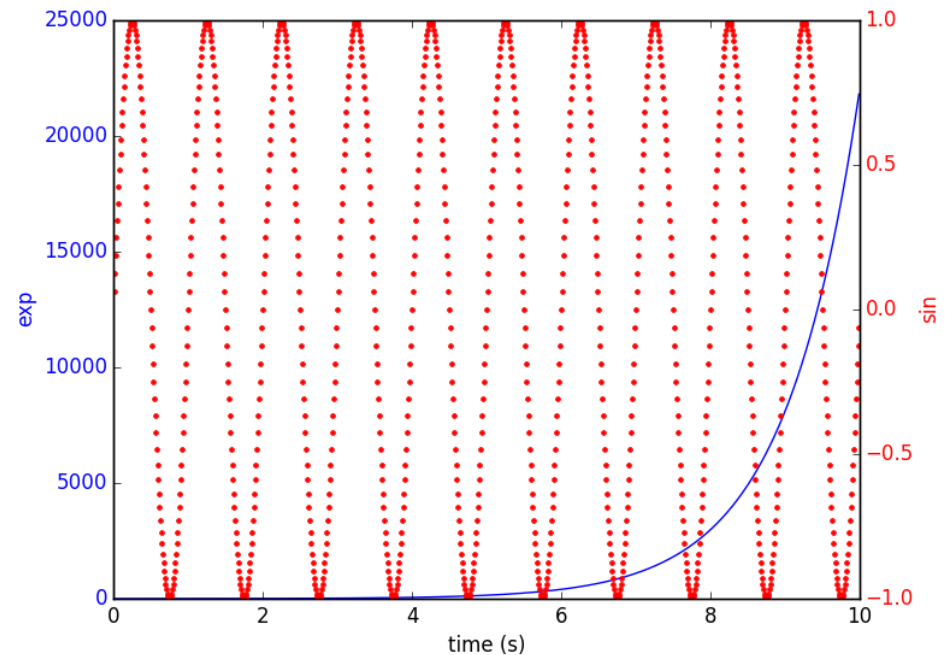
```
import numpy as np
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
t = np.arange(0.01, 10.0, 0.01)
s1 = np.exp(t)
ax1.plot(t, s1, 'b-')
ax1.set_xlabel('time (s)')

# Make the y-axis label and tick labels match the line color.
ax1.set_ylabel('exp', color='b')
for tl in ax1.get_yticklabels():
    tl.set_color('b')

...
```

# Multiple axes on one plot (2)

```
...  
ax2 = ax1.twinx()  
s2 = np.sin(2*np.pi*t)  
ax2.plot(t, s2, 'r.')  
ax2.set_ylabel('sin', color='r')  
for tl in ax2.get_yticklabels():  
    tl.set_color('r')  
  
plt.show()
```







# Contour plot – prepare data

```
import matplotlib
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
```

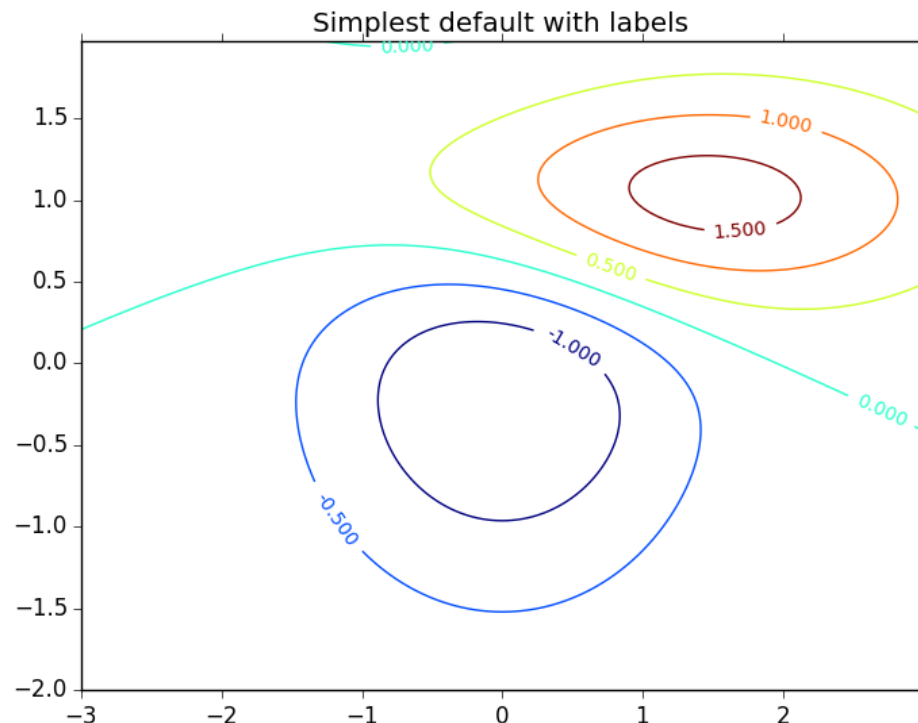
Assume this code applies to all contour examples.

```
matplotlib.rcParams['xtick.direction'] = 'out'
matplotlib.rcParams['ytick.direction'] = 'out'

delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
Z2 = mlab.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
# difference of Gaussians
Z = 10.0 * (Z2 - Z1)
```

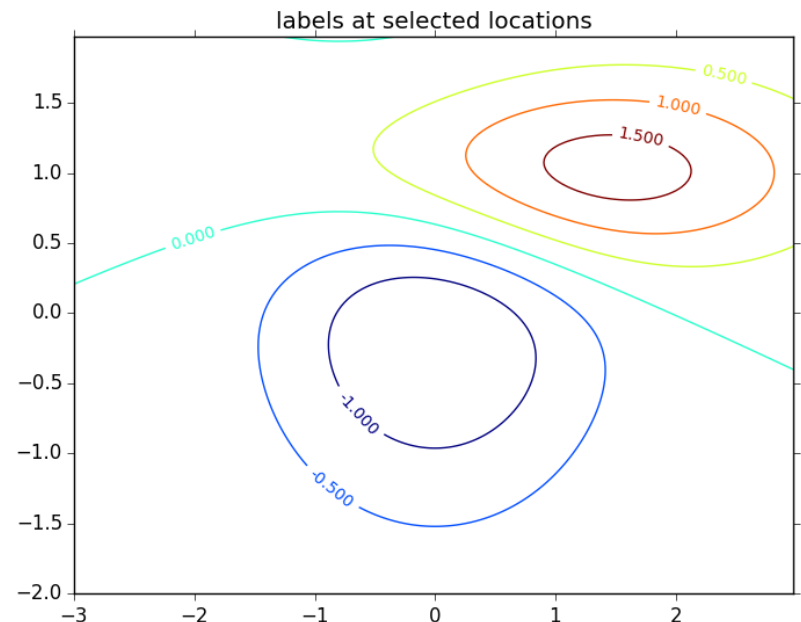
# Contour plot – default colours

```
CS = plt.contour(X, Y, Z)  
plt.clabel(CS, inline=1, fontsize=10)  
plt.title('Simplest default with labels')  
plt.show()
```



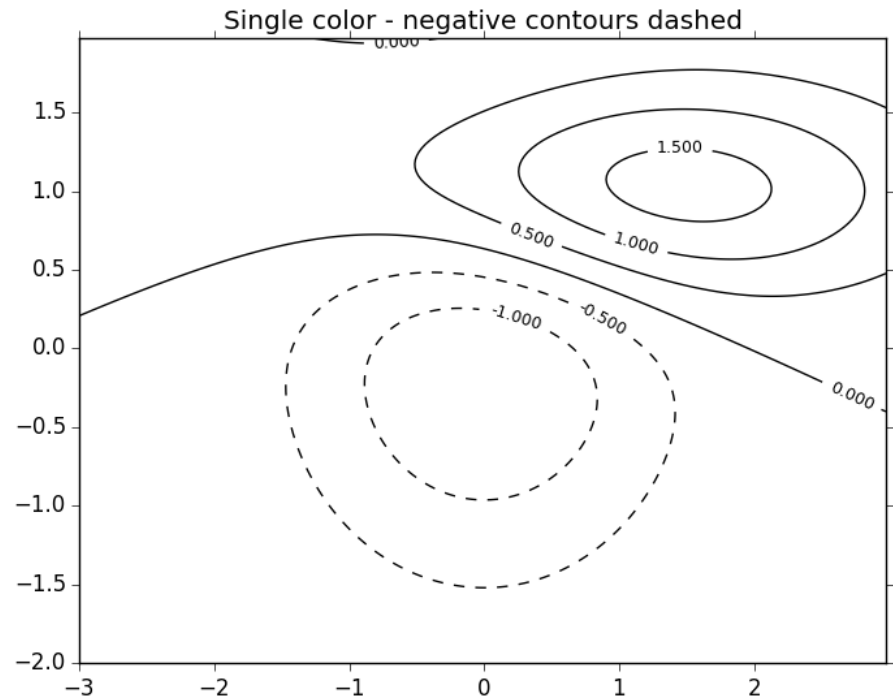
# Contour plot – control labels

```
CS = plt.contour(X, Y, Z)
manual_locations = [(-1, -1.4), (-0.62, -0.7), (-2, 0.5),
                    (1.7, 1.2), (2.0, 1.4), (2.4, 1.7)]
plt.clabel(CS, inline=1, fontsize=10,
           manual=manual_locations)
plt.title('labels at selected locations')
plt.show()
```



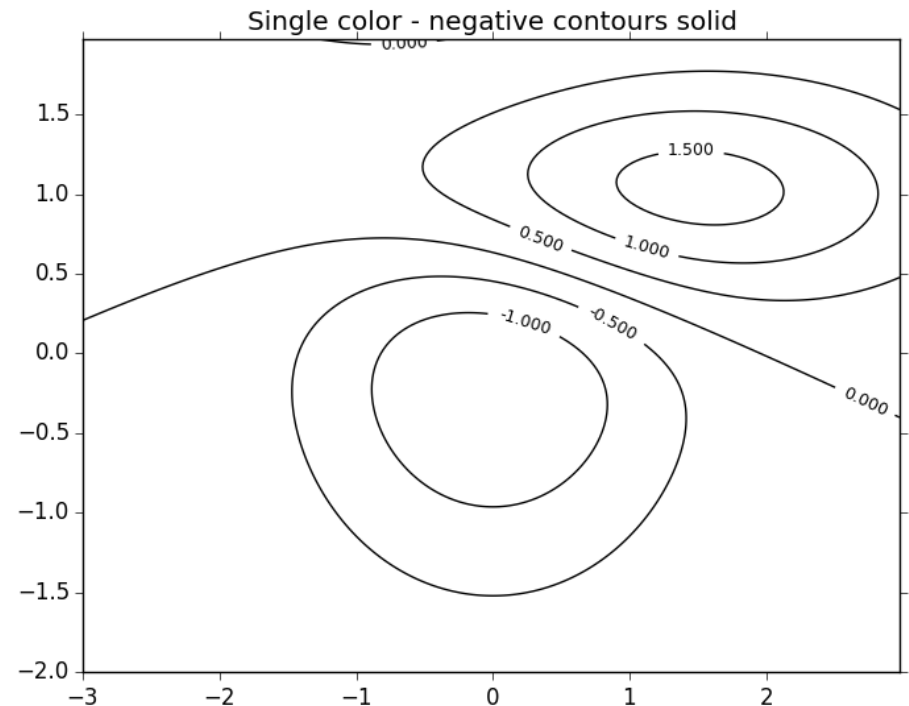
# Contour plot – with negative values

```
# This time negative contours will be dashed by default
CS = plt.contour(X, Y, Z, 6, colors='k')
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Single color - negative contours dashed')
plt.show()
```



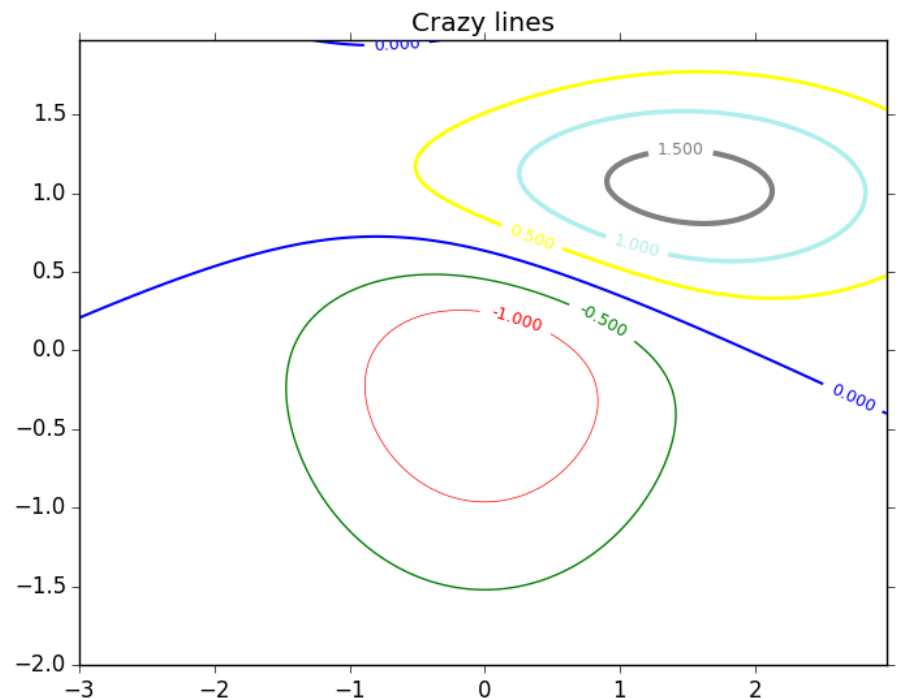
# Contour plot - set negative line style

```
# Override negative contours - use solid lines
matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
CS = plt.contour(X, Y, Z, 6, colors='k')
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Single color - negative contours solid')
plt.show()
```



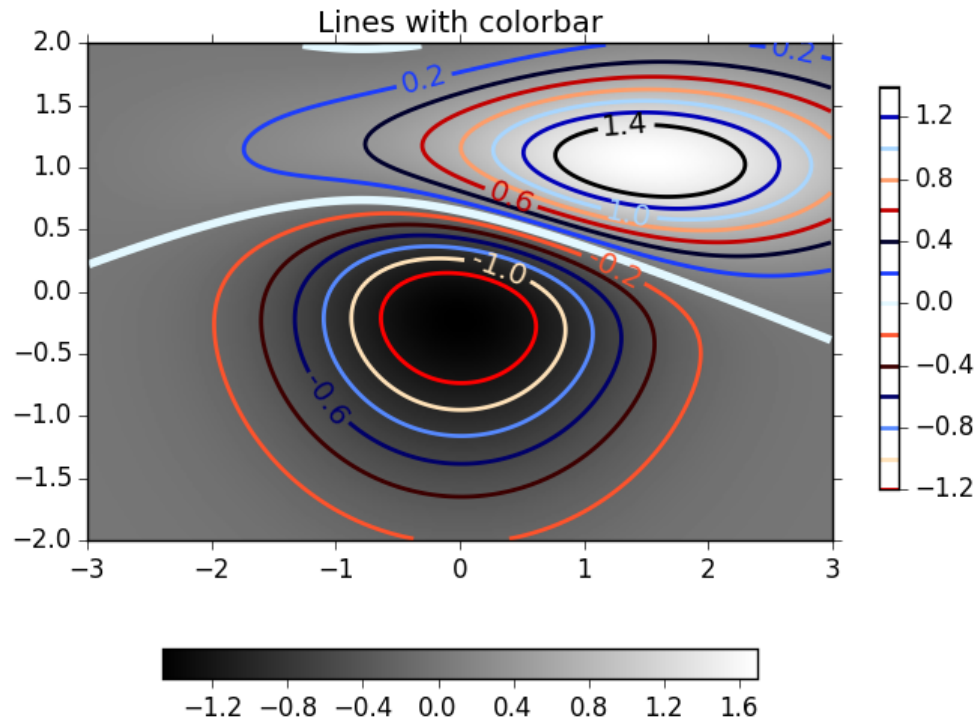
# Contour plot – specify colours

```
CS = plt.contour(X, Y, Z, 6, linewidths=np.arange(.5, 4, .5),  
                 colors=('r', 'green', 'blue', (1, 1, 0),  
                        '#afeeee', '0.5'))  
plt.clabel(CS, fontsize=9, inline=1)  
plt.title('Crazy lines')  
plt.show()
```



# Contour plot - smart

And you can keep going...



# Introducing Basemap

- The matplotlib `basemap` toolkit is a library for plotting 2D data on maps in Python.
- It is similar in functionality to the *MATLAB mapping toolbox*.
- **Basemap** does not do any plotting on its own, but provides the facilities to transform coordinates to one of 25 different map projections.
- **Matplotlib** is then used to plot contours, images, vectors, lines or points in the transformed coordinates. Shoreline, river and political boundary datasets are provided, along with methods for plotting them.



# Plotting maps – using Basemap

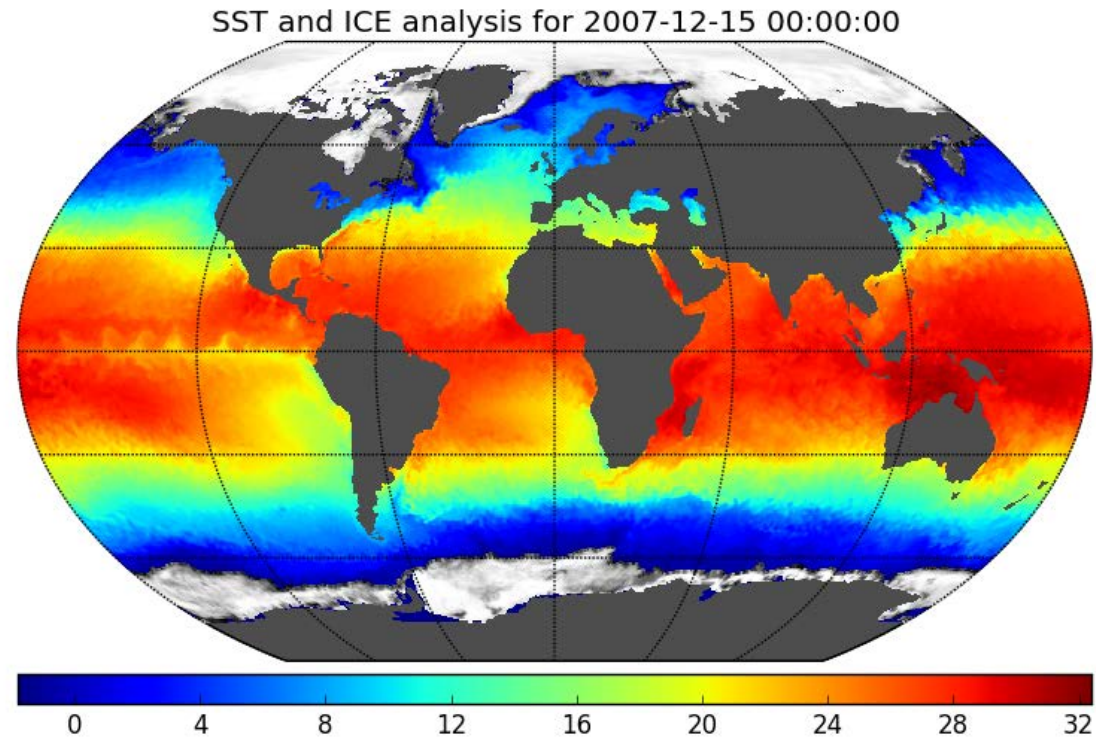
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
m = Basemap(width=12000000,height=9000000,projection='lcc',
            resolution=None,lat_1=45.,lat_2=55,lat_0=50,lon_0=-107.)
m.shadedrelief()
plt.text(0.2, 0.2, "What a lovely map!", color="white",
        weight="bold")
plt.show()
```



<http://matplotlib.org/basemap/users/geography.html>

# Plotting maps – and data

With about **30 lines of code** you can extract Sea Surface Temperature and Ice from a file and plot on a required projection:

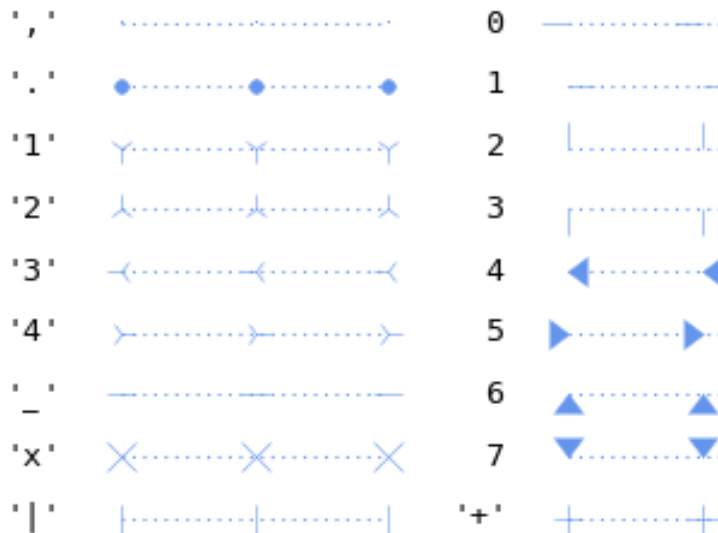


<http://matplotlib.org/basemap/users/examples.html>

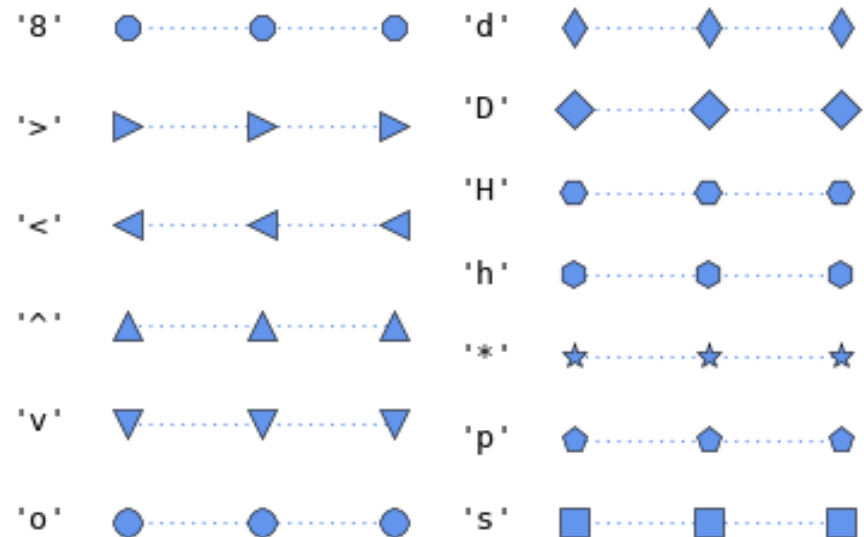
# Useful features: Markers

[http://matplotlib.org/examples/lines\\_bars\\_and\\_markers/marker\\_reference.html](http://matplotlib.org/examples/lines_bars_and_markers/marker_reference.html)

un-filled markers

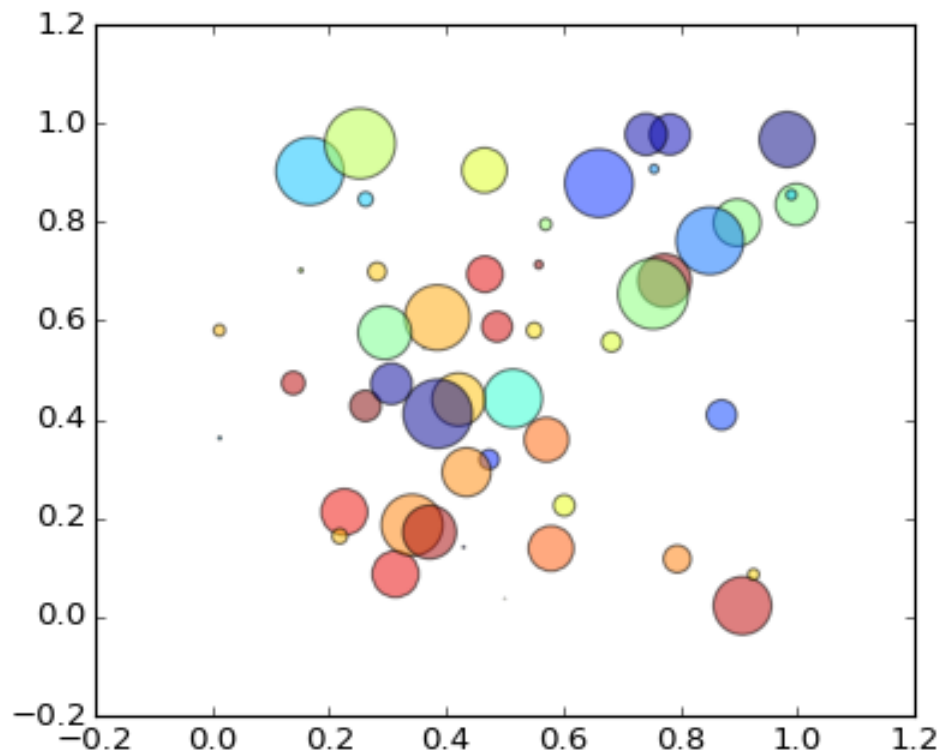


filled markers



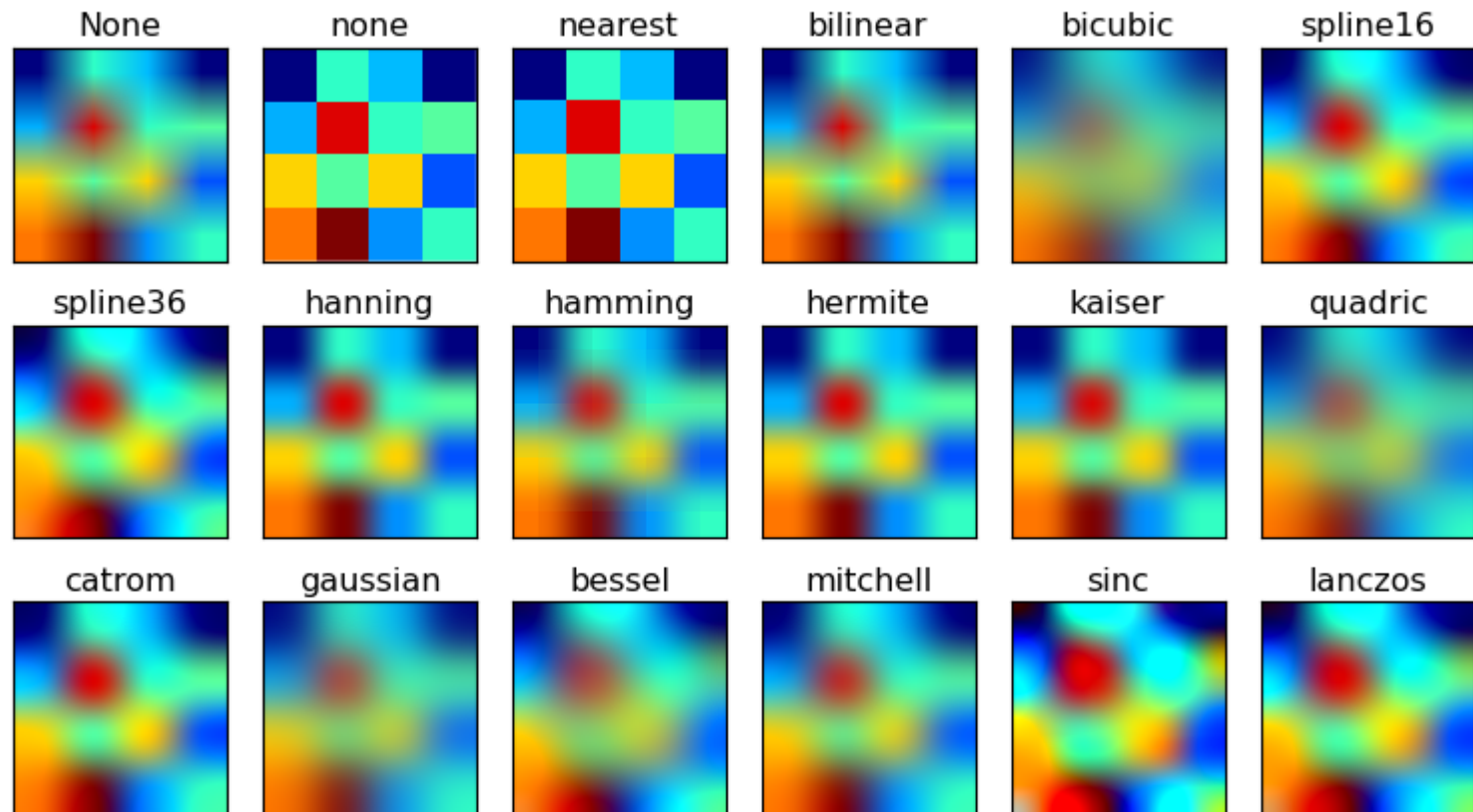
# Useful features: Scatter plots

[http://matplotlib.org/examples/shapes\\_and\\_collections/scatter\\_demo.html](http://matplotlib.org/examples/shapes_and_collections/scatter_demo.html)



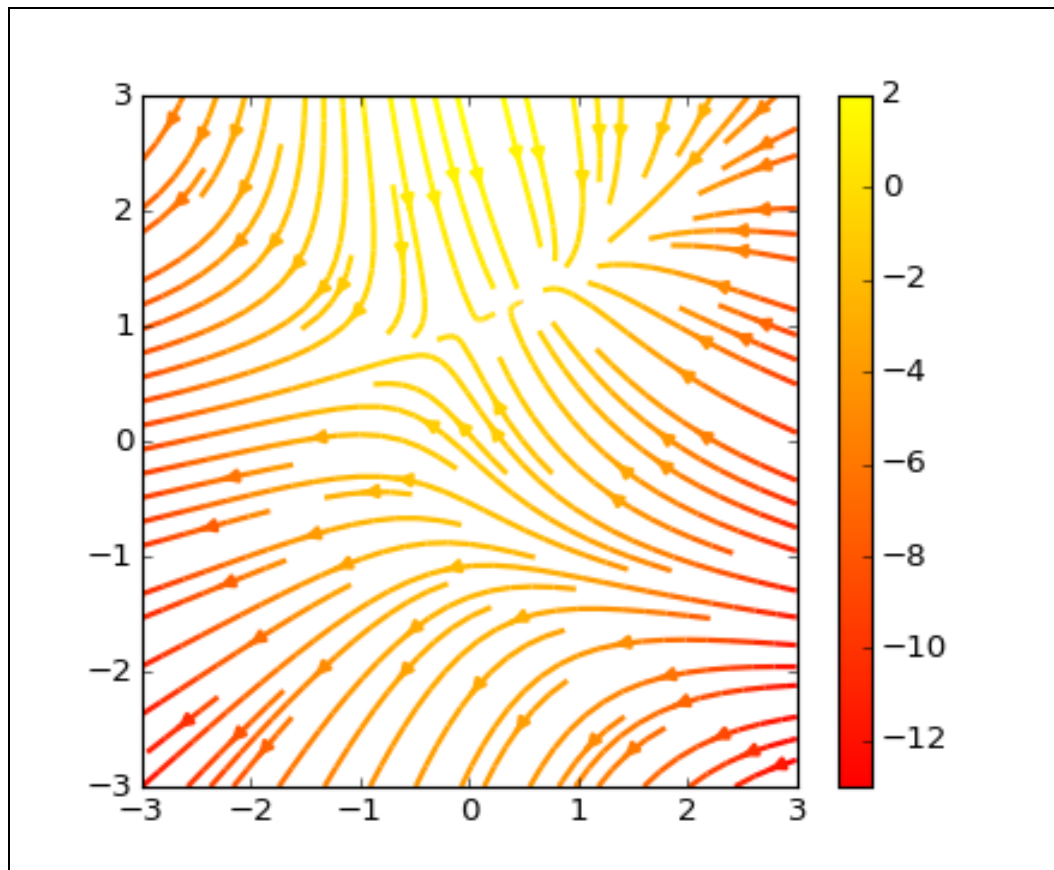
# Useful features: Interpolation

[http://matplotlib.org/examples/images\\_contours\\_and\\_fields/interpolation\\_methods.html](http://matplotlib.org/examples/images_contours_and_fields/interpolation_methods.html)



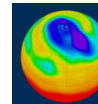
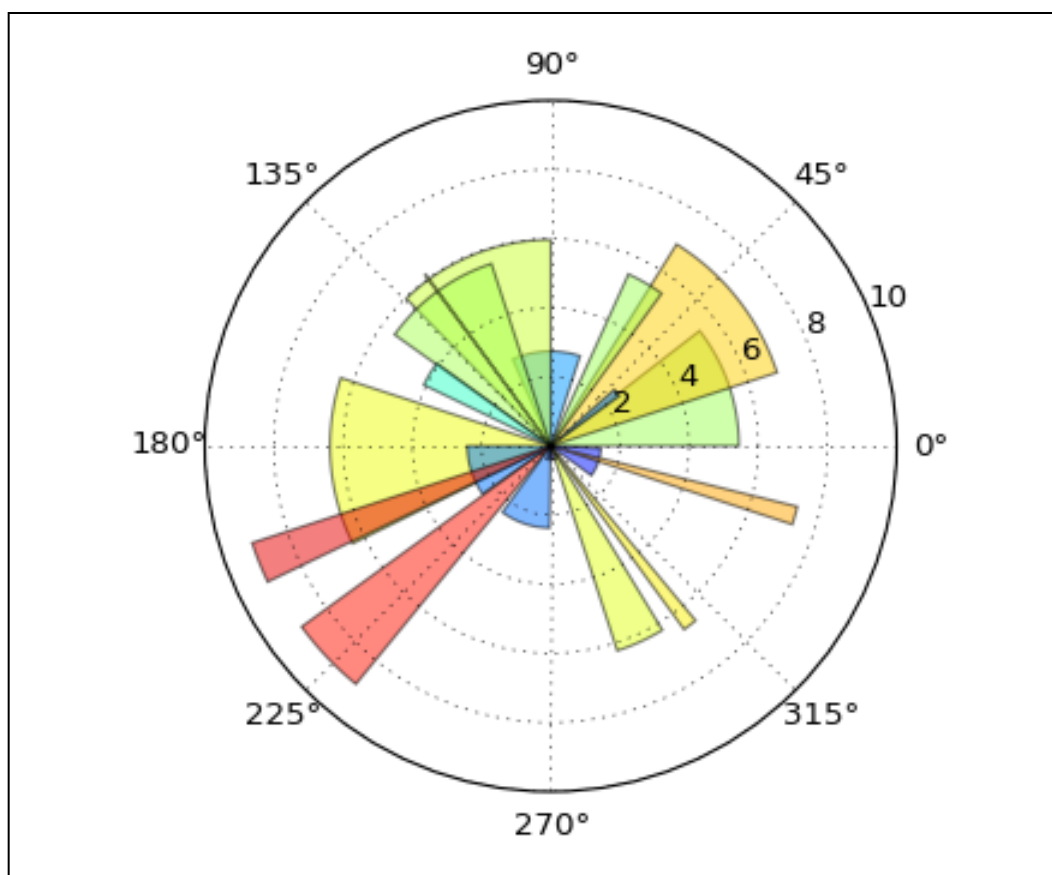
# Useful features: Stream plot

[http://matplotlib.org/examples/images\\_contours\\_and\\_fields/streamplot\\_demo\\_features.html](http://matplotlib.org/examples/images_contours_and_fields/streamplot_demo_features.html)



# Useful features: Polar bar

[http://matplotlib.org/examples/pie\\_and\\_polar\\_charts/polar\\_bar\\_demo.html](http://matplotlib.org/examples/pie_and_polar_charts/polar_bar_demo.html)



# One last word: the OOP interface

We have demonstrated Matplotlib using the "pylab" interface (which aims to mimic that of MATLAB).

You can interact with Matplotlib using its OOP interface (known as the *Matplotlib API*). This is a different interface to the same functionality.

**Over time you may wish to use the OOP interface for complex plotting applications.**



# More info

- Matplotlib:
  - <http://matplotlib.org>
- Matplotlib gallery:
  - <http://matplotlib.org/gallery.html>
- Pyplot reference:
  - [http://matplotlib.org/api/pyplot\\_summary.html](http://matplotlib.org/api/pyplot_summary.html)
- Basemap toolkit (for map plotting):
  - <http://matplotlib.org/basemap/>
- Books, videos and tutorials:
  - <http://matplotlib.org/1.4.3/resources/>