Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

# Visualisation in Python

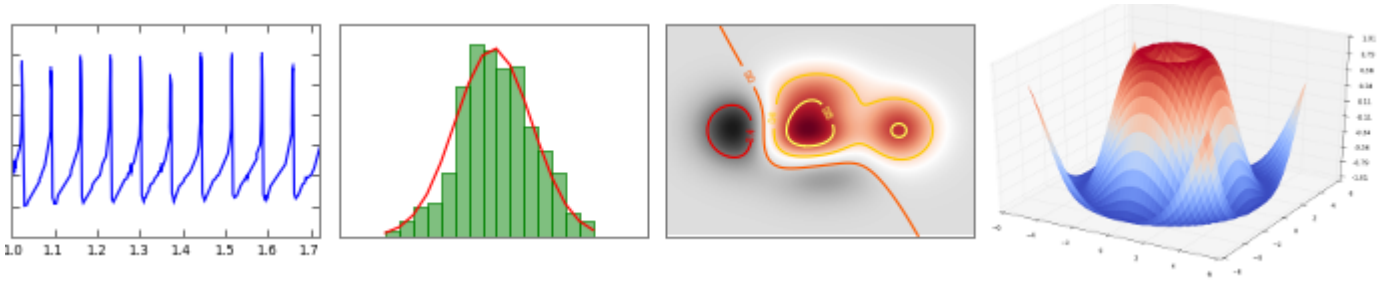
## Matplotlib

Thanks to all contributors:

Ag Stephens, Stephen Pascoe, Tommy Godfrey and Andy Heaps.

# 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 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

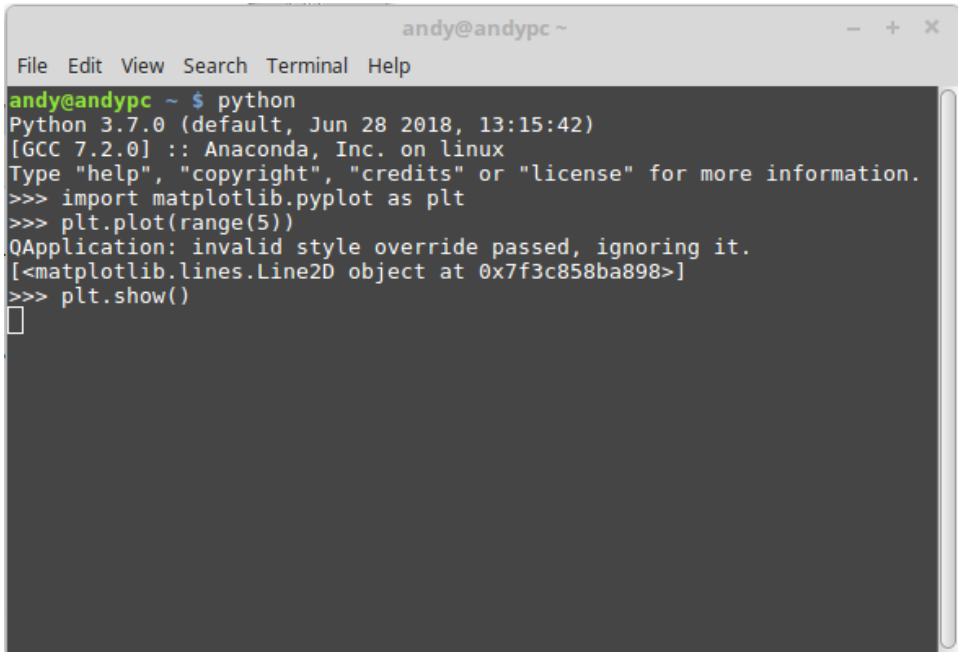
As with all open source Python tools there are other options and approaches available.

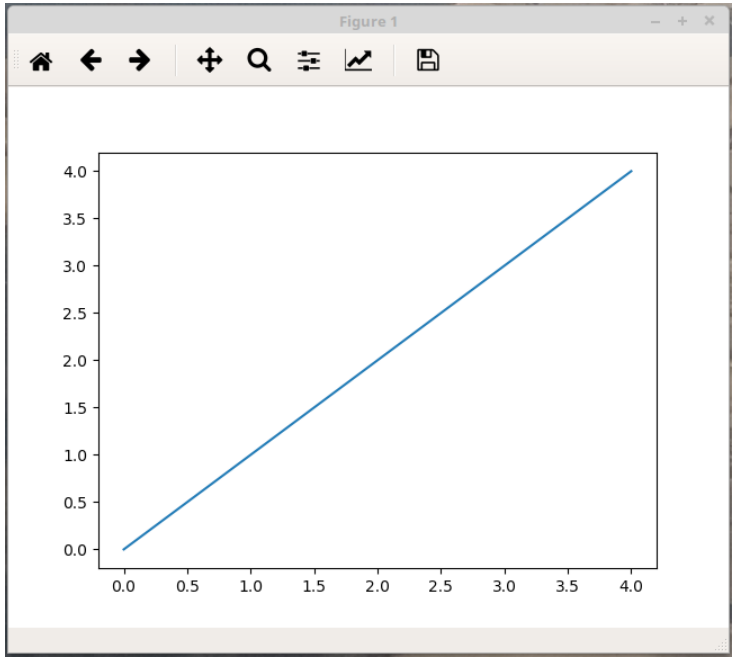
However, Matplotlib, like NumPy, has become the **clear leader** in its particular niche.

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

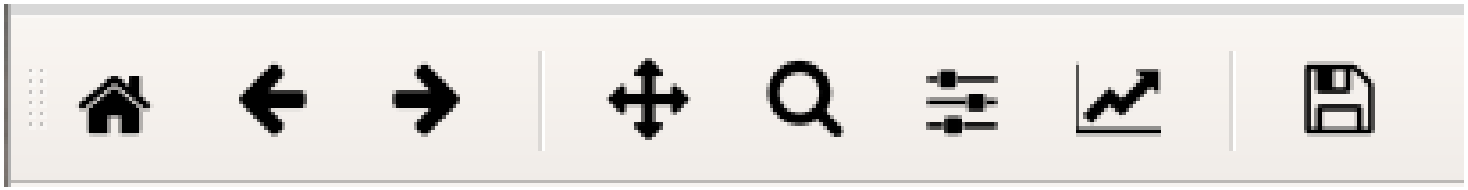
# Using Matplotlib Interactively

- Matplotlib has its own interactive plotting window:



# Using Matplotlib Interactively

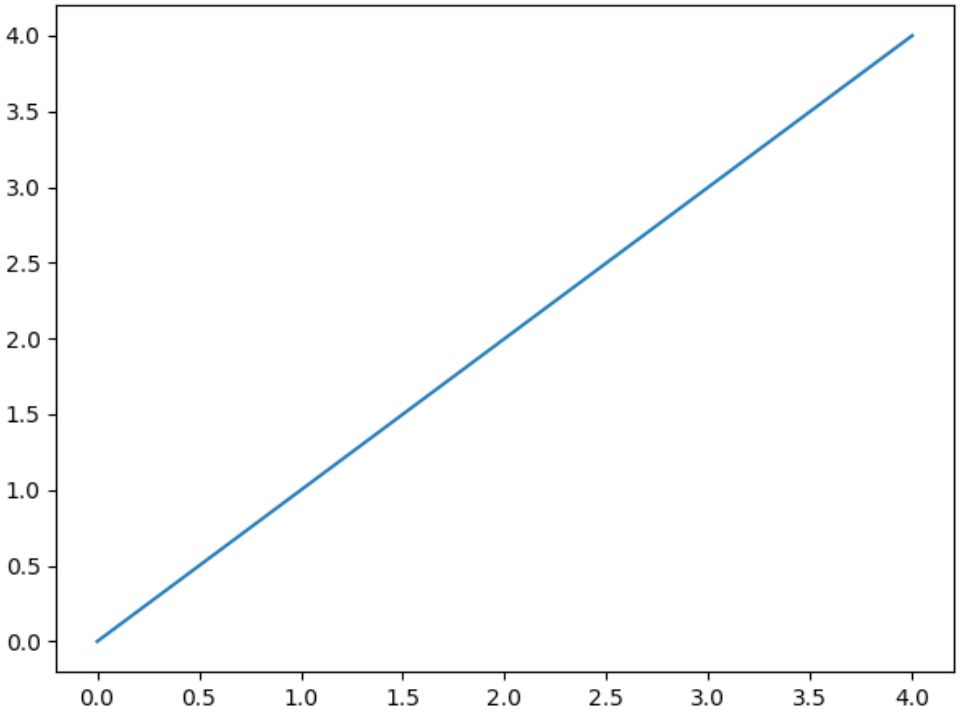


The buttons allow you to:

- 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

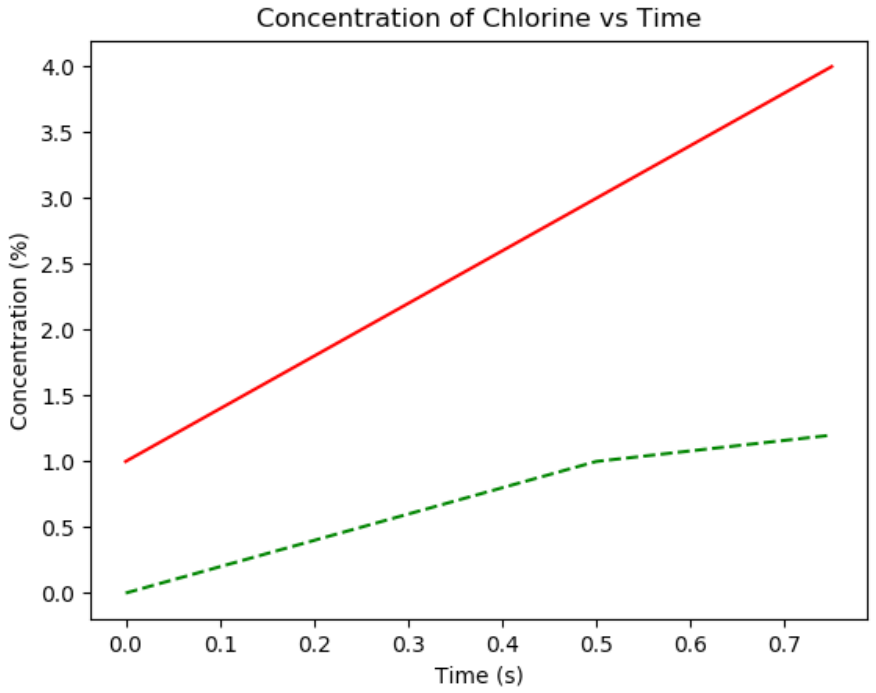
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.show()
```




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

# Two lines with axes and a title

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

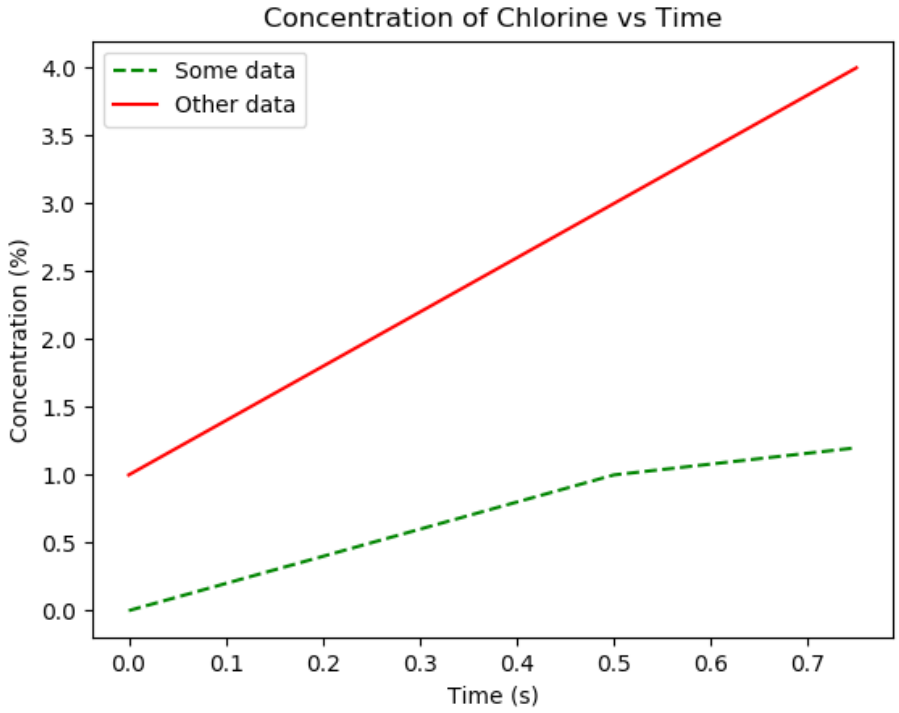



Assume we have always run:

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

# 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"

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

- To draw multiple plots from the same session:

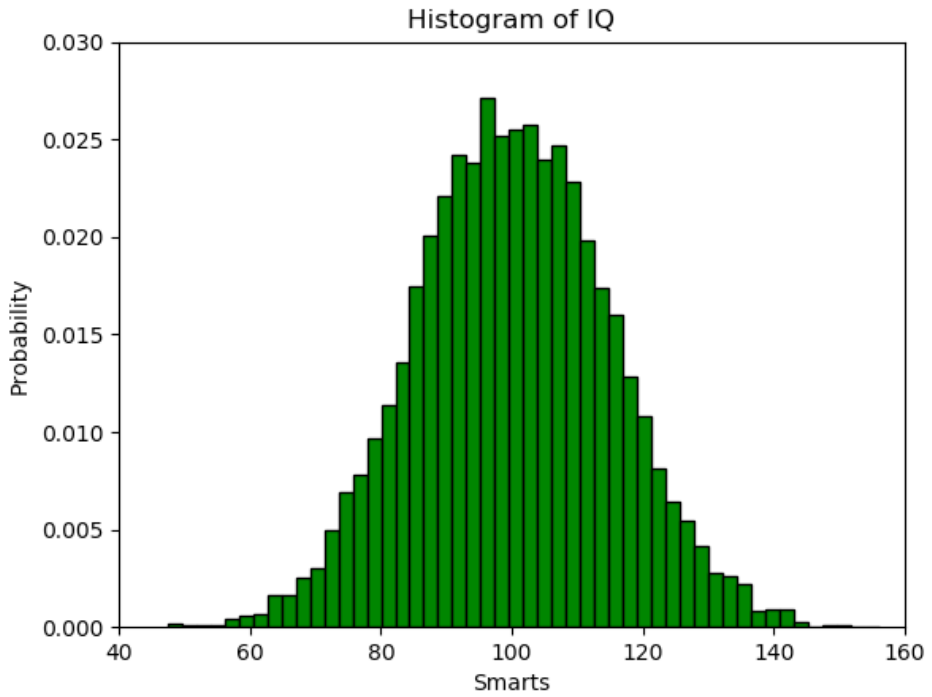
```
plt.figure()
plt.plot(range(5))
plt.figure(figsize = (10, 10)) # size in inches
plt.plot(range(100))
plt.show() # shows both figures
```

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

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

# Histogram plot

```
import numpy as np
import matplotlib.pyplot as plt
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
n, bins, patches = plt.hist(x,50,density=1,color='g',
edgecolor='k')
plt.axis([40, 160, 0, 0.03])
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.show()
```



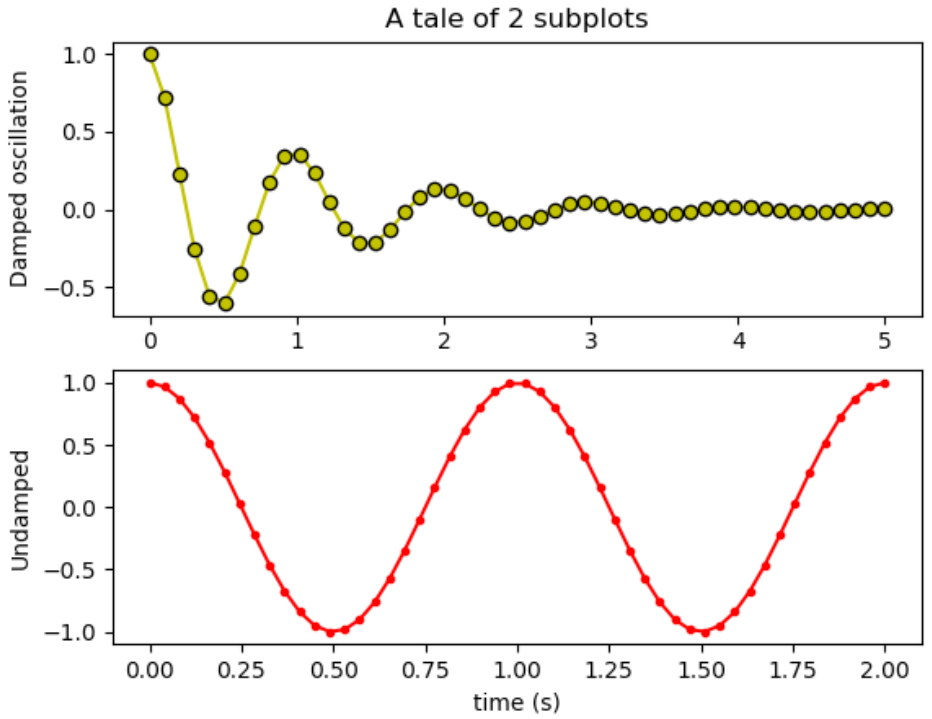
# Multiple plots – data and subplot function (1)

```
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 "subplot" function is defined as:
`subplot(nrows, ncols, plot_number)`
Here we define: 2 rows, 1 column.

# Multiple plots using subplot (2)

```
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-', markeredgecolor='k')
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()
```

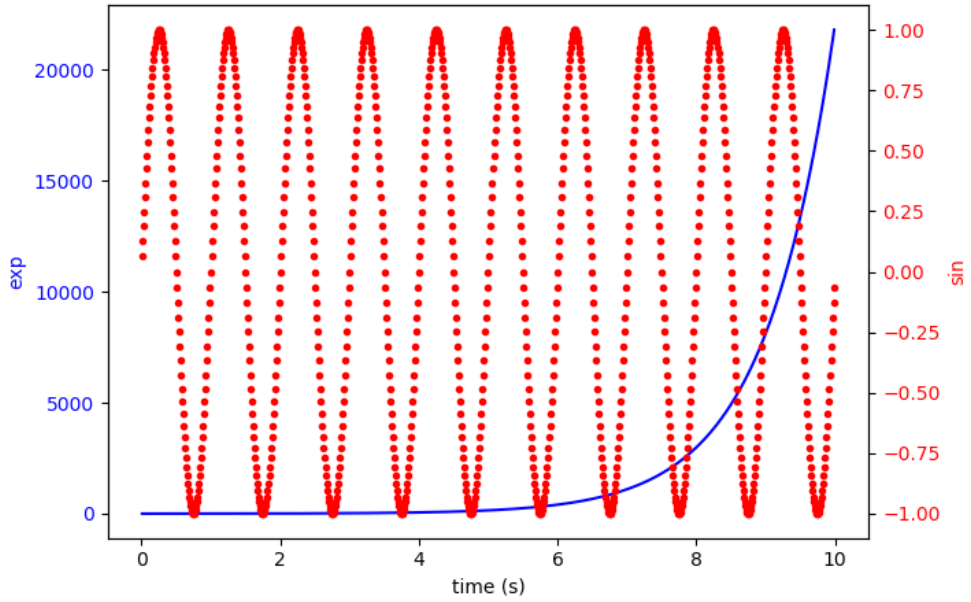


# Multiple axes on one plot (1)

```
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()
```



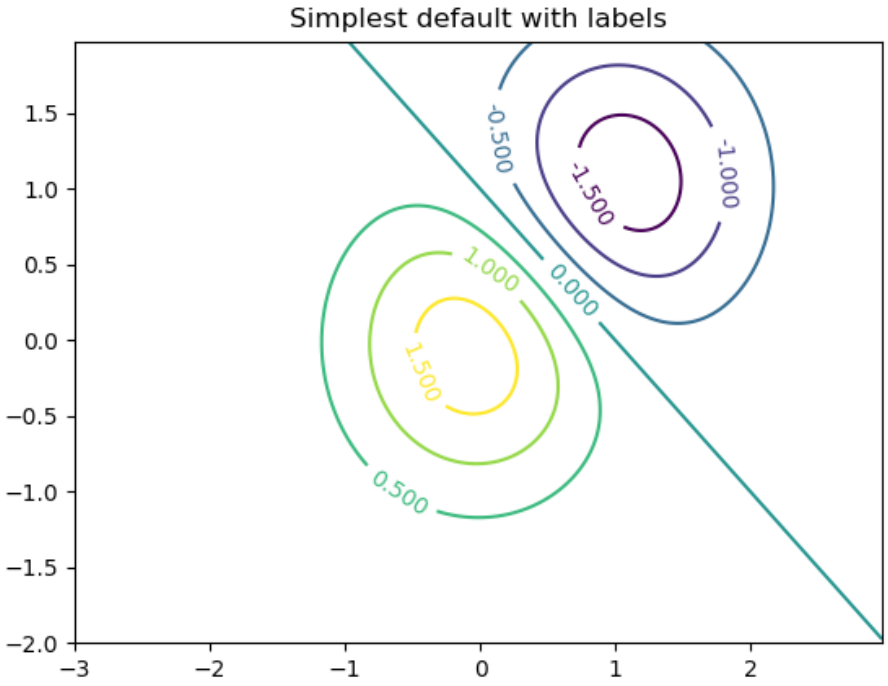
# Contour plot – prepare data

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

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)

# Make field to contour
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2
```

Assume this code applies to all contour examples.

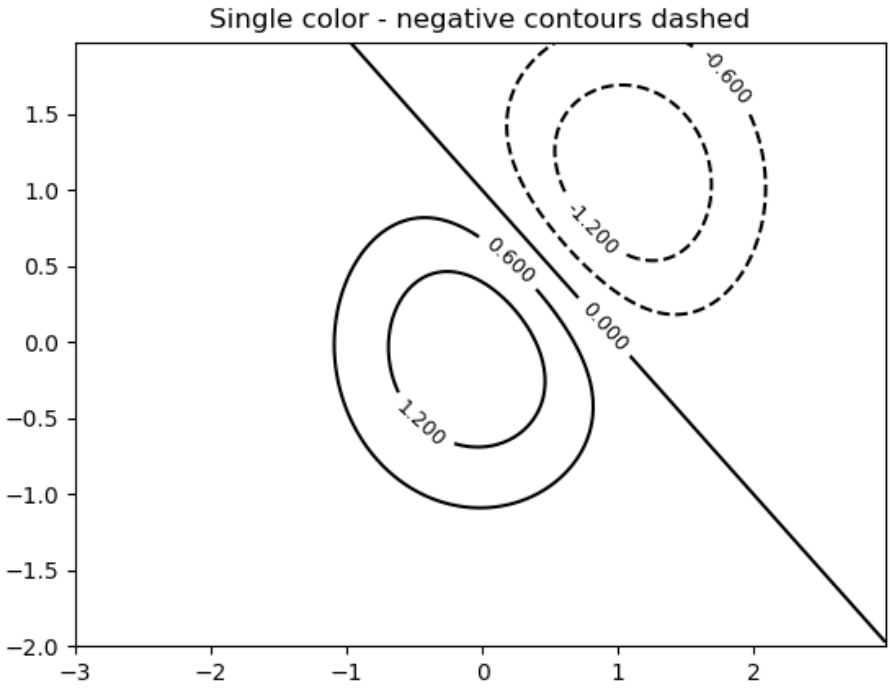
# Contour plot – default colours

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



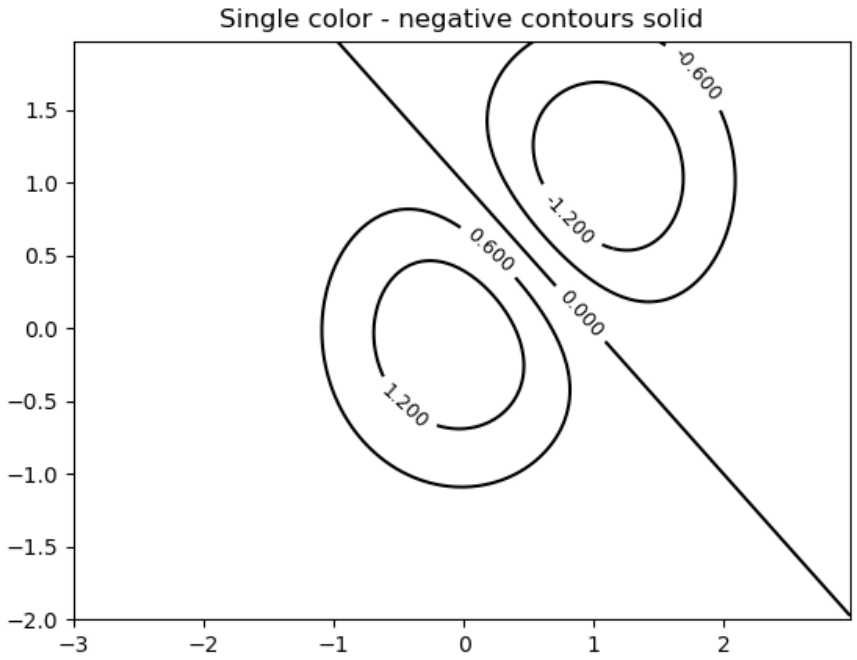
# Contour plot – with negative values

```
# 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()
```



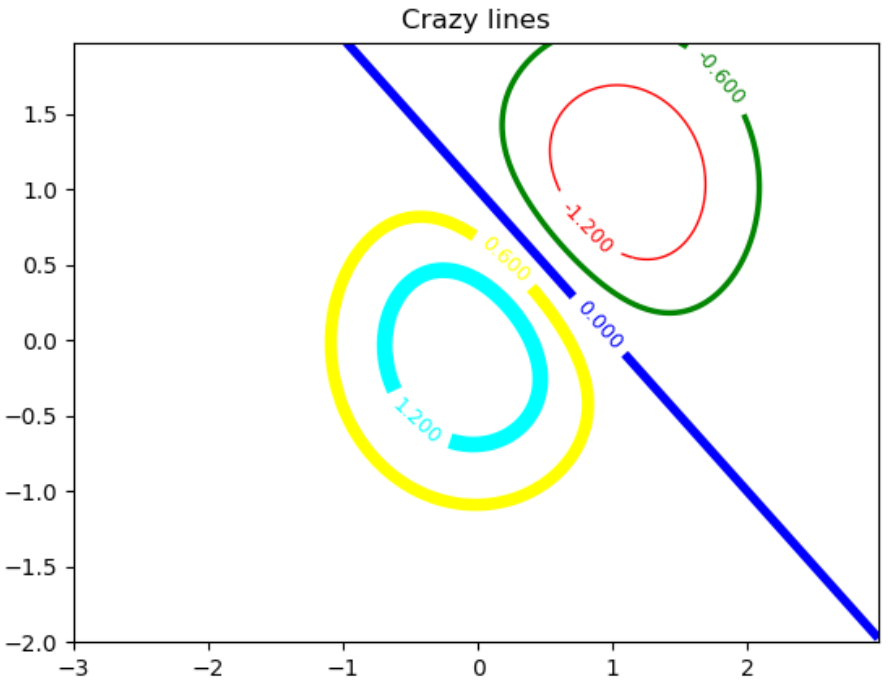
# Contour plot - set negative line style

```
# 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(1, 8, 1.5),
colors=('r', 'green', 'blue', 'yellow', 'cyan', 'wheat'))
plt.clabel(CS, fontsize=9, inline=1)
plt.title('Crazy lines')
plt.show()
```

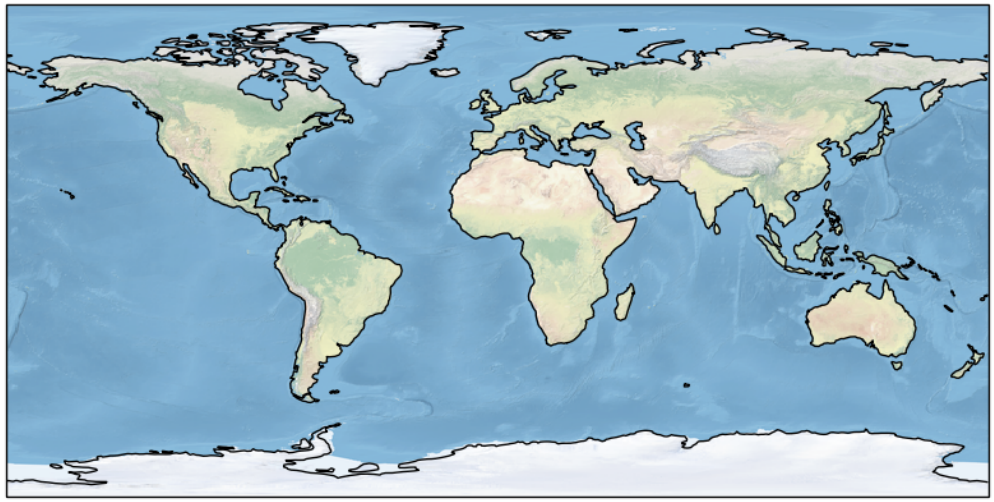


# Introducing **cartopy**

- **cartopy** is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses.
- **cartopy** does not do any plotting on its own, but provides the facilities to transform coordinates to one of 33 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 within **cartopy**, along with methods for plotting them.

# Plotting maps using **cartopy**

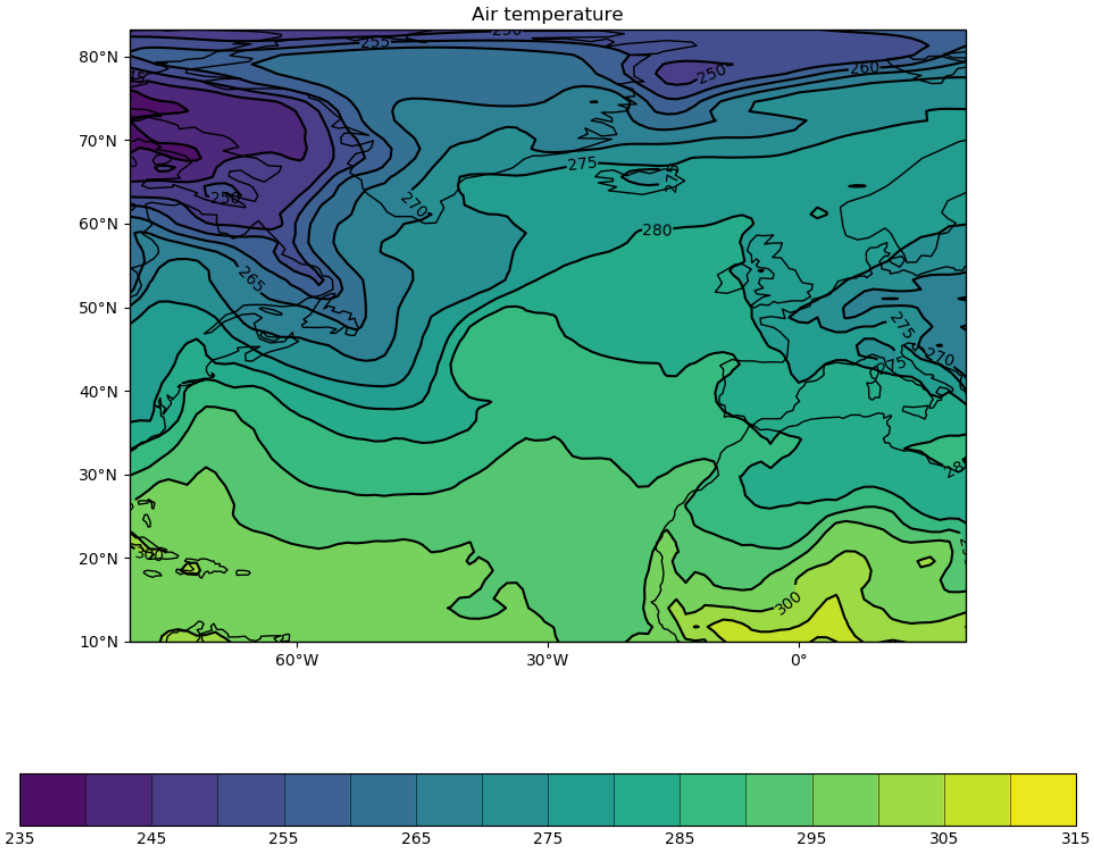
```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
ax.set_global()
ax.stock_img()
ax.coastlines()
fig.show()
```



https://scitools.org.uk/cartopy/docs/latest/gallery/index.html

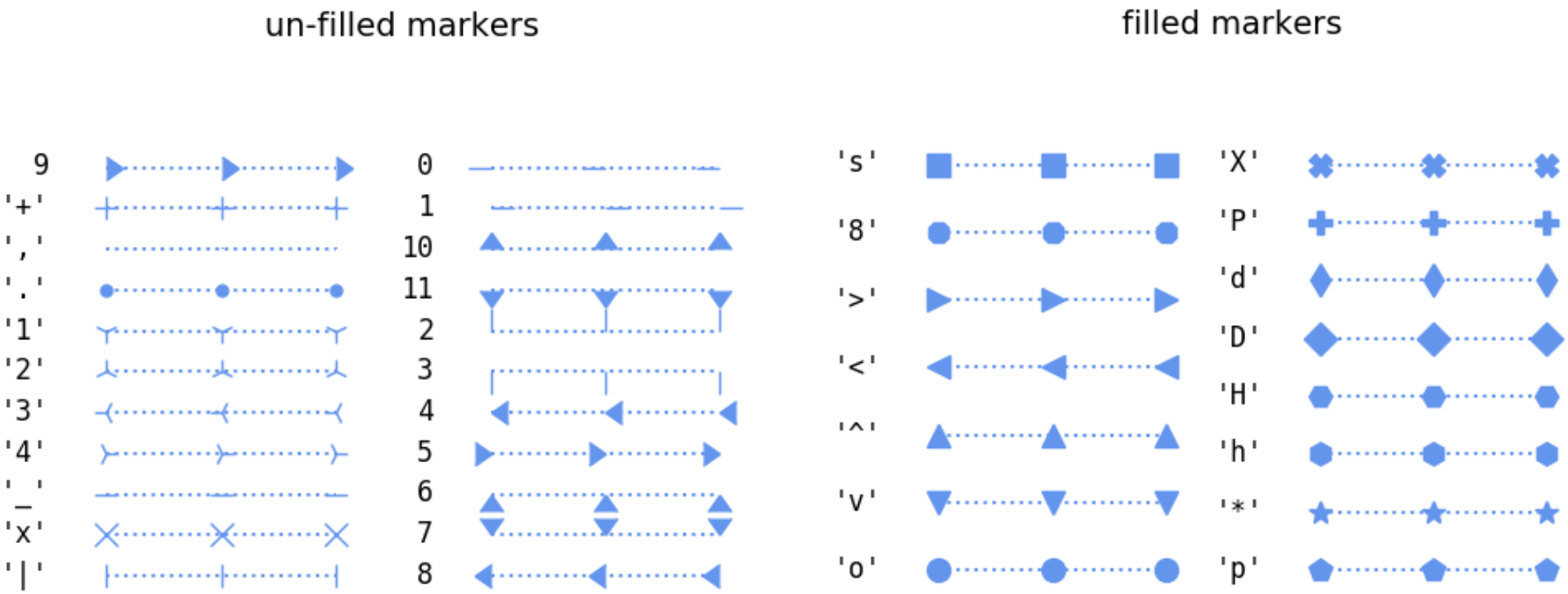
# Plotting maps – and data

With **24 lines of code** you can extract the Air Temperature from a netCDF file and plot on a required projection:



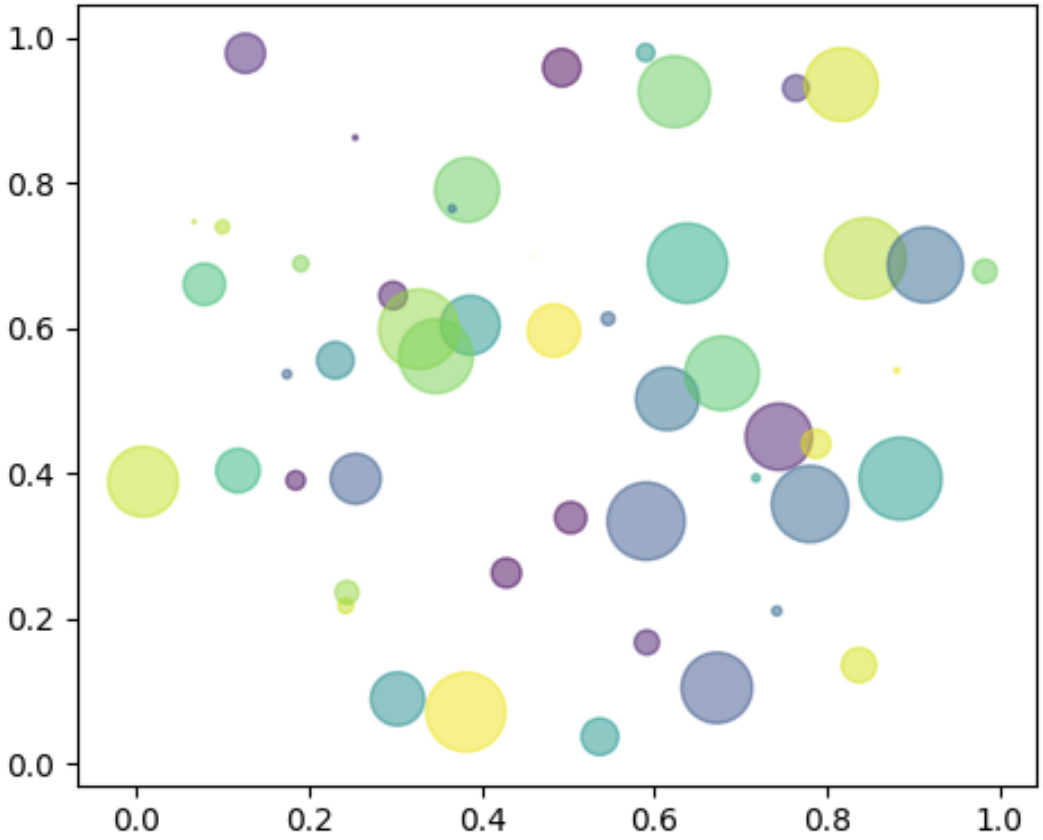
# Useful features: Markers

https://matplotlib.org/examples/lines_bars_and_markers/marker_reference.html



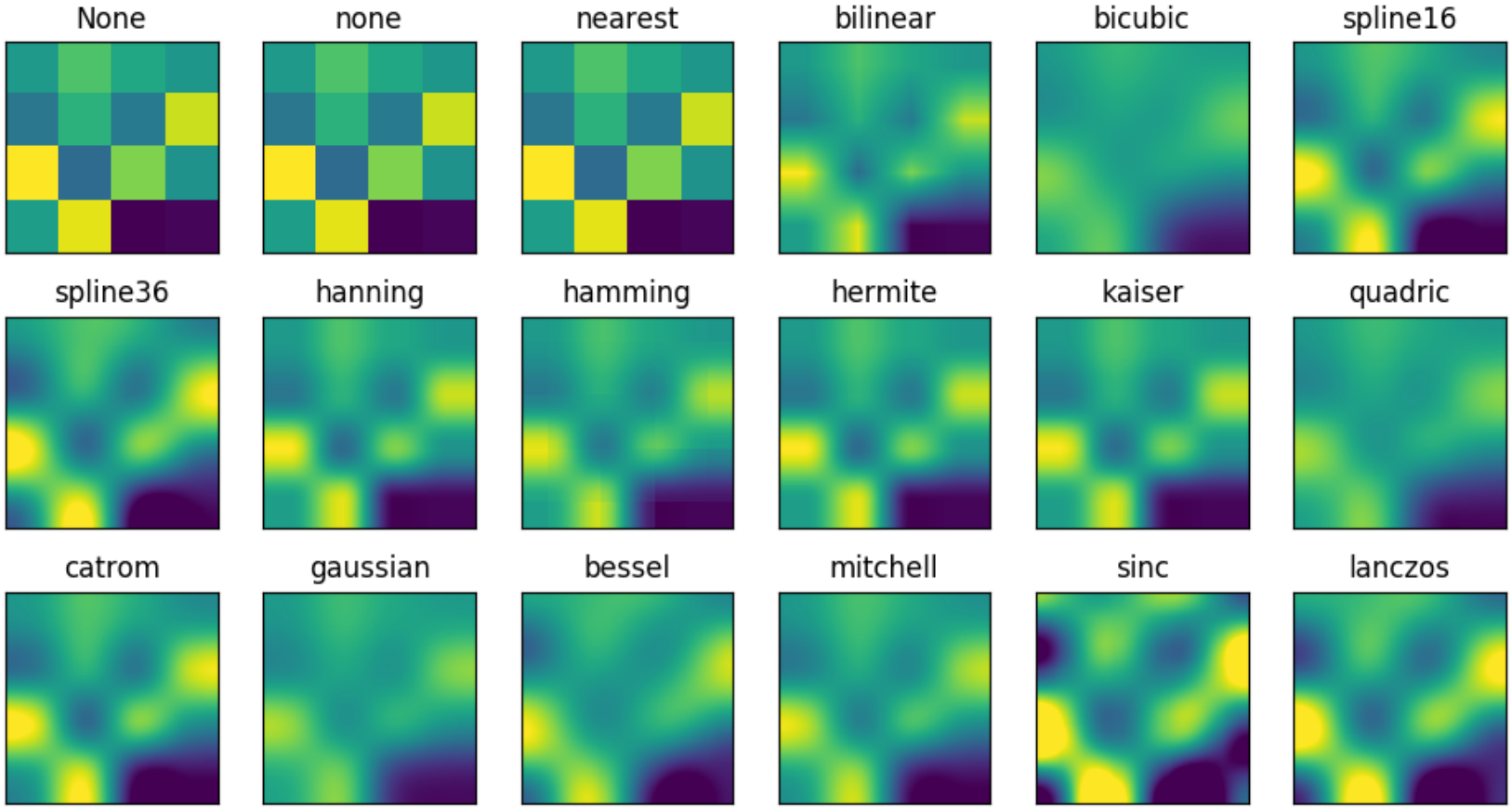
# Useful features: Scatter plots

https://matplotlib.org/examples/shapes_and_collections/scatter_demo.html



# Useful features: Interpolation

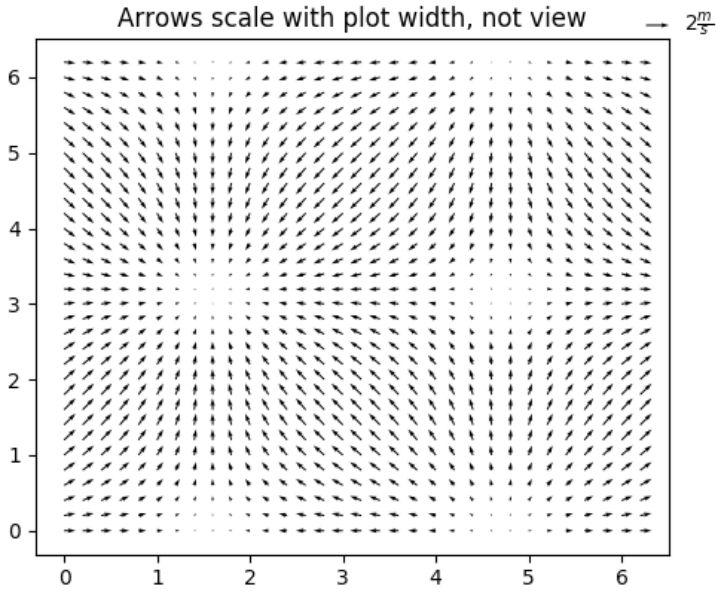
https://matplotlib.org/examples/images_contours_and_fields/interpolation_methods.html

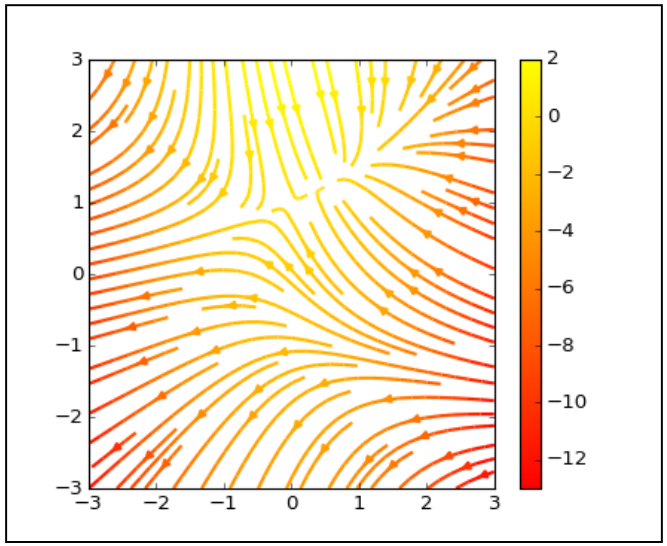


# Useful features: quiver and stream plot

https://matplotlib.org/examples/pylab_examples/quiver_demo.html

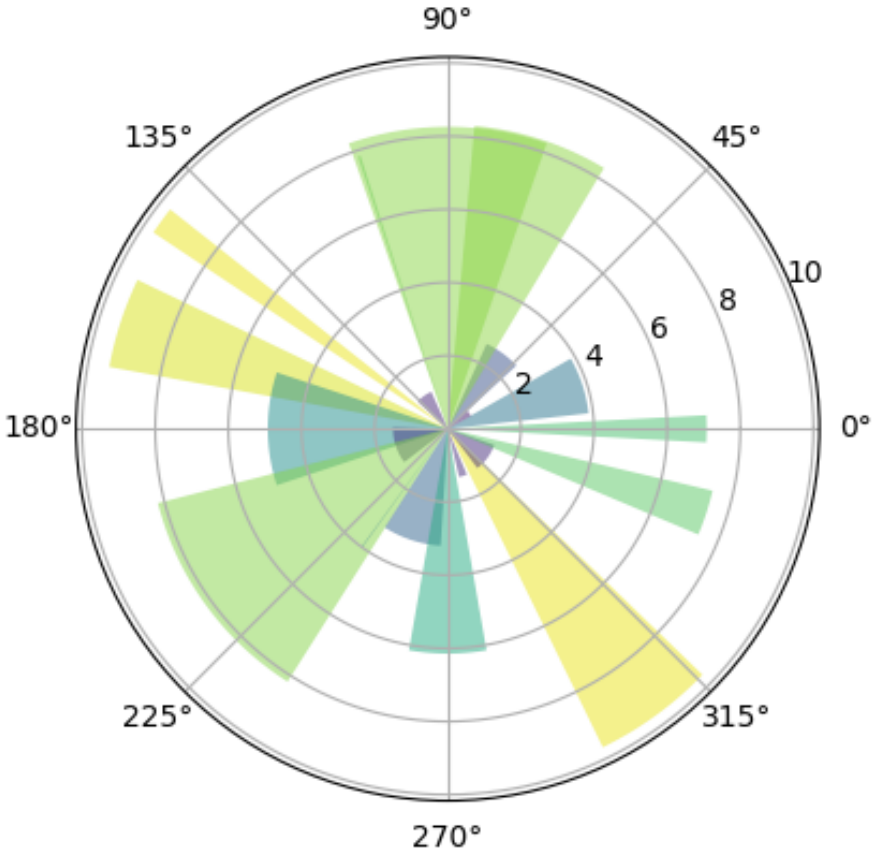
https://matplotlib.org/examples/images_contours_and_fields/streamplot_demo_features.html



# Useful features: Polar bar

https://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:

- https://matplotlib.org
- Matplotlib gallery:
- https://matplotlib.org/gallery
- Pyplot reference:
  https://matplotlib.org/api/pyplot_summary
- Cartopy (for map plotting):
- https://scitools.org.uk/cartopy/docs/latest/index.html
- Books, videos and tutorials:
  https://matplotlib.org/3.0.0/resources/