

Basic Plotting with Python and Matplotlib

This guide assumes that you have already installed NumPy and Matplotlib for your Python distribution. You can check if it is installed by importing it:

```
import numpy as np
import matplotlib.pyplot as plt    # The code below assumes this convenient renaming
```

For those of you familiar with MATLAB, the basic Matplotlib syntax is very similar.

1 Line plots

The basic syntax for creating line plots is `plt.plot(x,y)`, where `x` and `y` are arrays of the same length that specify the (x,y) pairs that form the line. For example, let's plot the cosine function from -2 to 1 . To do so, we need to provide a discretization (grid) of the values along the x -axis, and evaluate the function on each x value. This can typically be done with `numpy.arange` or `numpy.linspace`.

```
xvals = np.arange(-2, 1, 0.01)    # Grid of 0.01 spacing from -2 to 10
yvals = np.cos(xvals)             # Evaluate function on xvals
plt.plot(xvals, yvals)            # Create line plot with yvals against xvals
plt.show()                         # Show the figure
```

You should put the `plt.show` command last after you have made all relevant changes to the plot. You can create multiple figures by creating new figure windows with `plt.figure()`. To output all these figures at once, you should only have one `plt.show` command at the very end. Also, unless you turned the interactive mode on, the code will be paused until you close the figure window.

Suppose we want to add another plot, the quadratic approximation to the cosine function. We do so below using a different color and line type. We also add a title and axis labels, which is *highly recommended* in your own work. Also note that we moved the `plt.show` command to the end so that it shows both plots.

```
newyvals = 1 - 0.5 * xvals**2      # Evaluate quadratic approximation on xvals
plt.plot(xvals, newyvals, 'r--')   # Create line plot with red dashed line
plt.title('Example plots')
plt.xlabel('Input')
plt.ylabel('Function values')
plt.show()                         # Show the figure (remove the previous instance)
```

The third parameter supplied to `plt.plot` above is an optional format string. The particular one specified above gives a red dashed line. See the extensive Matplotlib documentation online for other formatting commands, as well as many other plotting properties that were not covered here:

http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot

2 Contour plots

The basic syntax for creating contour plots is `plt.contour(X,Y,Z,levels)`. To trace a contour, `plt.contour` requires a 2-D array `Z` that specifies function values on a grid. The underlying grid is given by `X` and `Y`, either both as 2-D arrays with the same shape as `Z`, or both as 1-D arrays where `len(X)` is the number of columns in `Z` and `len(Y)` is the number of rows in `Z`.

In most situations it is more convenient to work with the underlying grid (i.e., the former representation). The `meshgrid` function is useful for constructing 2-D grids from two 1-D arrays. It returns two 2-D arrays `X,Y` of the same shape, where each element-wise pair specifies an underlying (x,y) point on the grid. Function values on the grid `Z` can then be calculated using these `X,Y` element-wise pairs.

```
plt.figure()                # Create a new figure window
xlist = np.linspace(-2.0, 1.0, 100)  # Create 1-D arrays for x,y dimensions
ylist = np.linspace(-1.0, 2.0, 100)
X,Y = np.meshgrid(xlist, ylist)      # Create 2-D grid xlist,ylist values
Z = np.sqrt(X**2 + Y**2)             # Compute function values on the grid
```

We also need to specify the contour levels (of `Z`) to plot. You can either specify a positive integer for the number of automatically- decided contours to plot, or you can give a list of contour (function) values in the `levels` argument. For example, we plot several contours below:

```
plt.contour(X, Y, Z, [0.5, 1.0, 1.2, 1.5], colors = 'k', linestyles = 'solid')
plt.show()
```

Note that we also specified the contour colors and linestyles. By default, negative contours are given by dashed lines, hence we specified `solid`. Again, many properties are described in the Matplotlib specification: http://matplotlib.sourceforge.net/api/plotting_api.html#matplotlib.pyplot.contour

3 More plotting properties

The function considered above should actually have circular contours. Unfortunately, due to the different scales of the axes, the figure likely turned out to be flattened and the contours appear like ellipses. This is undesirable, for example, if we wanted to visualize 2-D Gaussian covariance contours. We can force the aspect ratio to be equal with the following command (placed before `plt.show`):

```
plt.axes().set_aspect('equal')      # Scale the plot size to get same aspect ratio
```

Finally, suppose we want to zoom in on a particular region of the plot. We can do this by changing the axis limits (again before `plt.show`). The input list to `plt.axis` has form `[xmin, xmax, ymin, ymax]`.

```
plt.axis([-1.0, 1.0, -0.5, 0.5])    # Set axis limits
```

Notice that the aspect ratio is still equal after changing the axis limits. Also, the commands above only change the properties of the current axis. If you have multiple figures you will generally have to set them for each figure before calling `plt.figure` to create the next figure window.

You can find out how to set many other axis properties at:

http://matplotlib.sourceforge.net/api/plotting_api.html#matplotlib.pyplot.axis

http://matplotlib.sourceforge.net/api/axes_api.html#matplotlib.axes

The final link covers many things, but most functions for changing axis properties begin with “`set_`”.

4 Figures

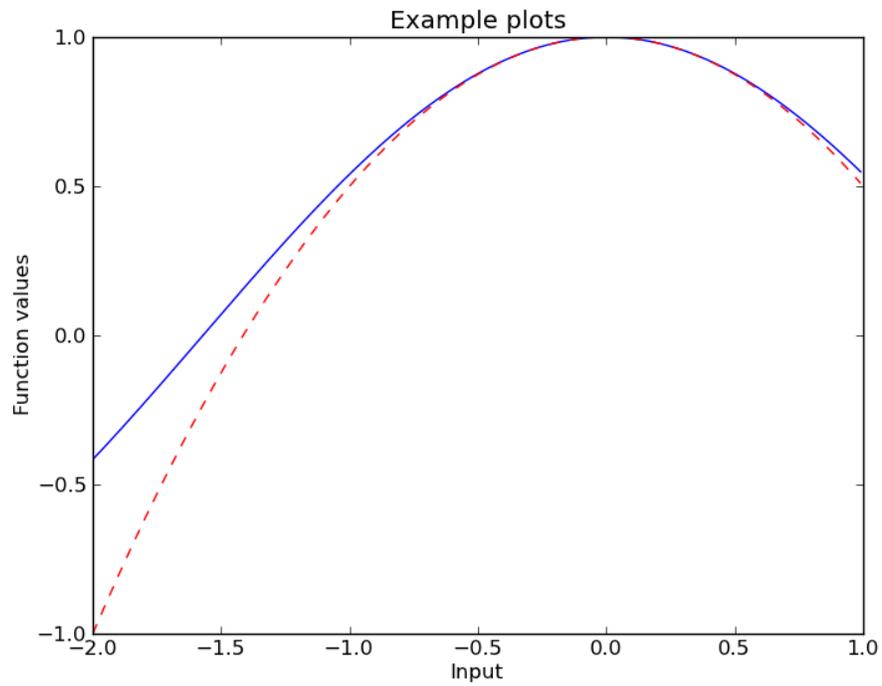


Figure 1: Example from section on line plots.

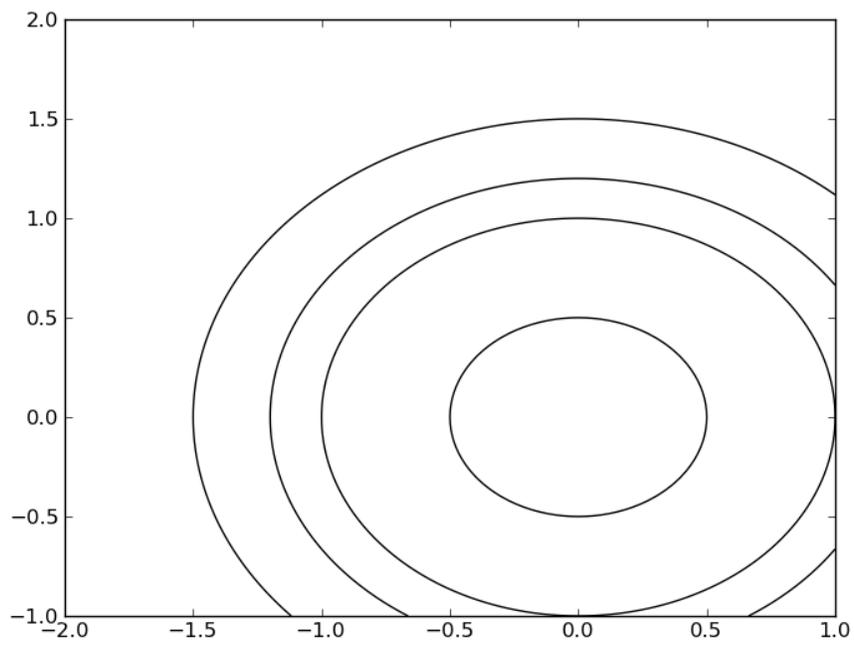


Figure 2: Example from section on contour plots.

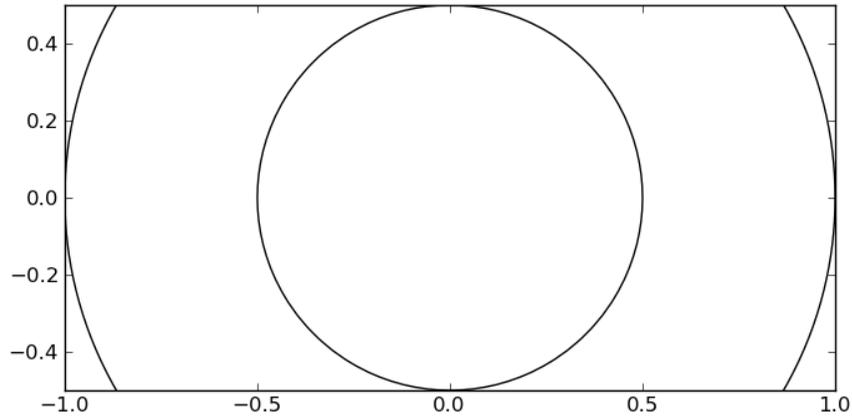


Figure 3: Setting the aspect ratio to be equal and zooming in on the contour plot.

5 Code

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

xvals = np.arange(-2, 1, 0.01) # Grid of 0.01 spacing from -2 to 10
yvals = np.cos(xvals) # Evaluate function on xvals
plt.plot(xvals, yvals) # Create line plot with yvals against xvals
# plt.show() # Show the figure

newyvals = 1 - 0.5 * xvals**2 # Evaluate quadratic approximation on xvals
plt.plot(xvals, newyvals, 'r--') # Create line plot with red dashed line
plt.title('Example plots')
plt.xlabel('Input')
plt.ylabel('Function values')
# plt.show() # Show the figure

plt.figure() # Create a new figure window
xlist = np.linspace(-2.0, 1.0, 100) # Create 1-D arrays for x,y dimensions
ylist = np.linspace(-1.0, 2.0, 100)
X,Y = np.meshgrid(xlist, ylist) # Create 2-D grid xlist,ylist values
Z = np.sqrt(X**2 + Y**2) # Compute function values on the grid

plt.contour(X, Y, Z, [0.5, 1.0, 1.2, 1.5], colors = 'k', linestyles = 'solid')
plt.axes().set_aspect('equal') # Scale the plot size to get same aspect ratio
plt.axis([-1.0, 1.0, -0.5, 0.5]) # Change axis limits
plt.show()
```