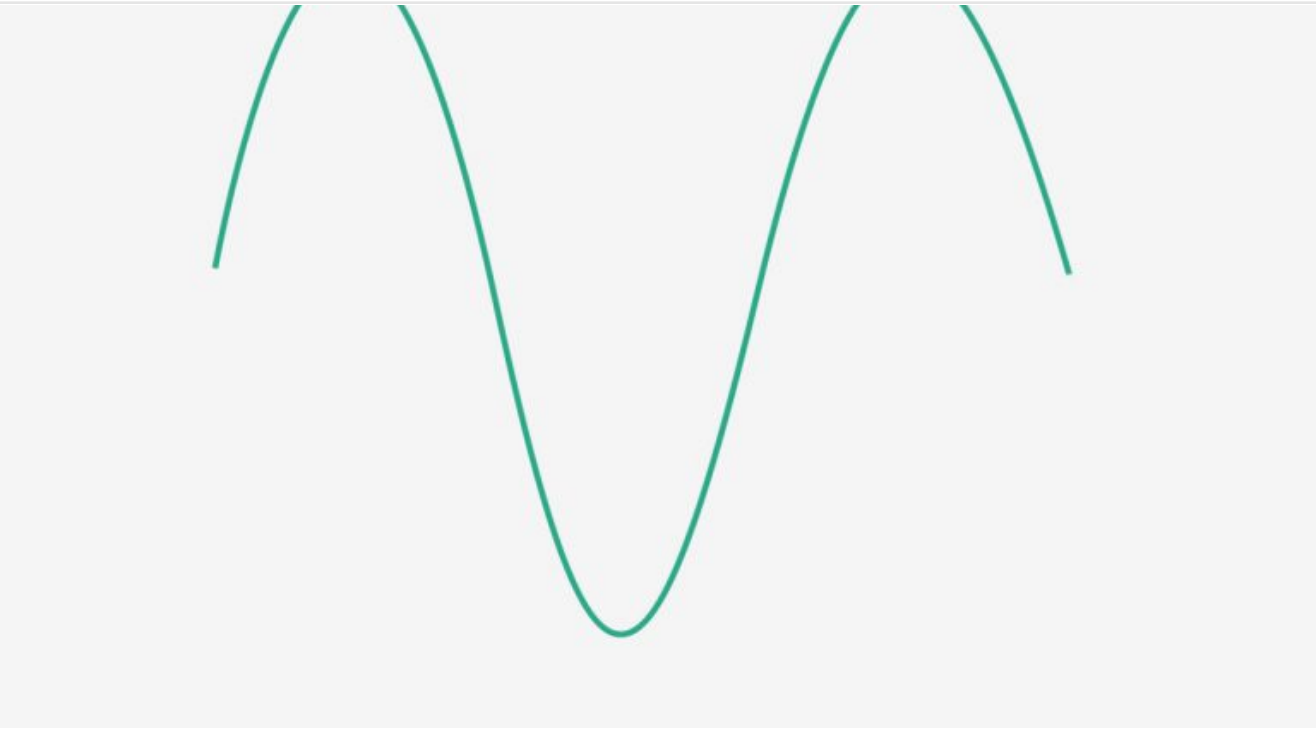
Simple Line Plots with Matplotlib

This recipe covers the basics of setting up a matplotlib plot, and how to create simple line plots.

By Jake VanderPlas

March 3, 2015



Simple line plot

This recipe will teach you how to make interactive plots, like this:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from IPython.html.widgets import interact

def plot_sine(frequency=1.0, amplitude=1.0):
    plt.ylim(-1.0, 1.0);
    x = np.linspace(0, 10, 1000)
    plt.plot(x, amplitude*np.sin(x*frequency));

interact(plot_sine, frequency=(0.5, 10.0), amplitude=(0.0, 1.0));
```

Perhaps the simplest of all plots is the visualization of a single function y = f(x). Here we will take a first look at creating a simple plot of this type. For all

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
print("Dependencies loaded...")
```

In their simplest form, a figure and axes can be created as follows:

```
fig = plt.figure()
ax = plt.axes()
```

In matplotlib, the *figure* (an instance of the class `plt.Figure`) can be thought of as a single container which contains all the objects representing axes, graphics, text, labels, etc. The *axes* (an instance of the class `plt.Axes`) is what we see above: a bounding box with ticks and labels, which will eventually contain other plot elements. Through this book, we'll commonly use the variable name `fig` to refer to a figure instance, and `ax` to refer to an axes instance or set of axes instances.

Once we have created an axes, we can use the `ax.plot` function to plot some data. Let's start with a simple sine wave:

```
fig = plt.figure()
ax = plt.axes()

x = np.linspace(0, 10, 1000)
ax.plot(x, np.cos(x));
```

Alternatively, we can use the pylab interface and let the figure and axes be

```
plt.plot(x, np.sin(x));
```

If we want to create a single figure with multiple lines, we can simply call the `plot` function multiple times:

```
plt.plot(x, np.sin(x*2));
plt.plot(x, np.sin(x));
```

That's all there is to plotting simple functions in matplotlib! Below we'll dive into some more details about how to control the appearance of the axes and lines.

Adjusting the Plot: Line Colors and Styles

The first adjustment you might wish to make to a plot is to control the line colors and styles. The `plt.plot()` function takes additional arguments which can be used to specify these. To adjust the color, you can use the `color` keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways, which we'll show below:

```
plt.plot(x, np.sin(x - 0), color='blue')        # specify color by name
plt.plot(x, np.sin(x - 1), color='g')           # short color code (works fc
plt.plot(x, np.sin(x - 2), color='0.75')        # Greyscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex color code (RRGGBB frc
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, between 0 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse')  # all html color names are s
```

Similarly, the line style can be adjusted using the `linestyle` keyword:

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':')  # dotted;
```

If you would like to be extremely terse, these linestyle codes and color codes can be combined into a single non-keyword argument to the `plt.plot()` function:

```
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r')  # dotted red;
```

These single-character color codes reflect the standard abbreviations in the RGB (Red/Green/Blue) and CMYK (Cyan/Magenta/Yellow/blacK) color systems, commonly used for digital color graphics.

There are many other keyword arguments that can be used to fine-tune the appearance of the plot: for more details, refer to matplotib's online documentation, or the docstring of the `plt.plot()` function.

but sometimes it's nice to have finer control. Here we'll briefly see how to change the limits of the x and y axes. The most basic way to do this is to use the `plt.xlim()` and `plt.ylim()` methods to set the numerical limits of the x and y axes:

```
plt.plot(x, np.sin(x))

plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```

If for some reason you'd like either axis to be displayed in reverse, you can simply reverse the order of the arguments:

```
plt.plot(x, np.sin(x))

plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```

A useful related method is `plt.axis()`: note here the potential confusion between *axes* with an *e*, and *axis* with an *i*. This method allows you to set the x and y limits with a single call, by passing a list which specifies `[xmin, xmax, ymin, ymax]`:

```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```

The `plt.axis()` method goes even beyond this, allowing you to do things like automatically tighten the bounds around the current plot:

```
plt.axis('tight');
```

It allows even higher-level specifications, such as ensuring an equal aspect ratio so that one unit in x is equal to one unit in y:

```
plt.plot(x, np.sin(x))
plt.axis('equal');
```

For more information on axis limits and the other capabilities of the `plt.axis` method, refer to matplotlib's online documentation.

Labeling Plots

As the last piece of this recipe, we'll briefly look at the labeling of plots: titles, axis labels, and simple legends.

Titles and axis labels are the simplest of these: there are methods which can be used to quickly set these:

```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```

The position, size, and style of these labels can be adjusted using optional arguments to the function. For more information, see the matplotlib documentation and the docstrings of each of these functions.

When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type. Again, matplotlib has a built-in

plot function:

```
plt.plot(x, np.sin(x), '-r', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend();
```

As you can see, the `plt.legend()` function keeps track of the line style and color, and matches these with the correct label. More information on specifying and formatting plot legends can be found in the `plt.legend` doc string; additionally, we will cover some more advanced legend options in recipe X.X.

Sidebar: Gotchas

While most `plt` functions translate directly to `ax` methods (such as `plt.plot()` & `ax.plot()`), this is not always the case. In particular, functions to set limits, labels, and titles are slightly modified. For transitioning between matlab-style functions and object-oriented methods, make the following changes:

- `plt.xlabel()` & `plt.ylabel()` become `ax.set_xlabel()` & `ax.set_ylabel()`
- `plt.xlim()` & `plt.ylim()` become `ax.set_xlim()` & `ax.set_ylim()`
- `plt.title()` becomes `ax.set_title()`

Post topics: Data science