Simple Line Plots

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. As with all the following sections, we'll start by setting up the notebook for plotting and importing the functions we will use:

```
In[1]: %matplotlib inline
       import matplotlib.pyplot as plt
       plt.style.use('seaborn-whitegrid')
       import numpy as np
```

For all Matplotlib plots, we start by creating a figure and an axes. In their simplest form, a figure and axes can be created as follows (Figure 4-5):

```
In[2]: fig = plt.figure()
       ax = plt.axes()
```

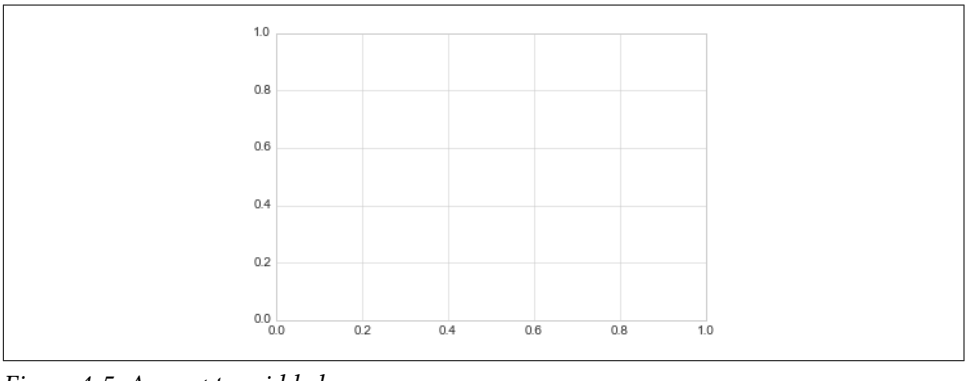

Figure 4-5. An empty gridded axes

In Matplotlib, the *figure* (an instance of the class `plt.Figure`) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels. 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 the plot elements that make up our visualization. Throughout 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 group 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 sinusoid (Figure 4-6):

```
In[3]: fig = plt.figure()
       ax = plt.axes()

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

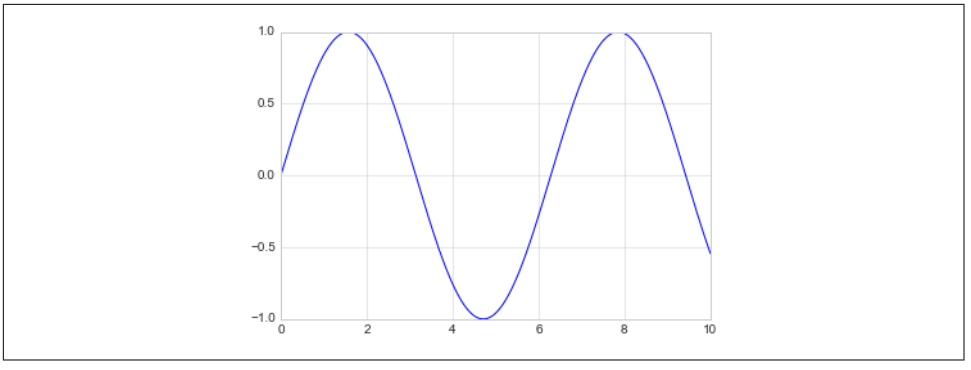

Figure 4-6. A simple sinusoid

Alternatively, we can use the pylab interface and let the figure and axes be created for us in the background (Figure 4-7; see "Two Interfaces for the Price of One" on page 222 for a discussion of these two interfaces):

```
In[4]: plt.plot(x, np.sin(x));
```

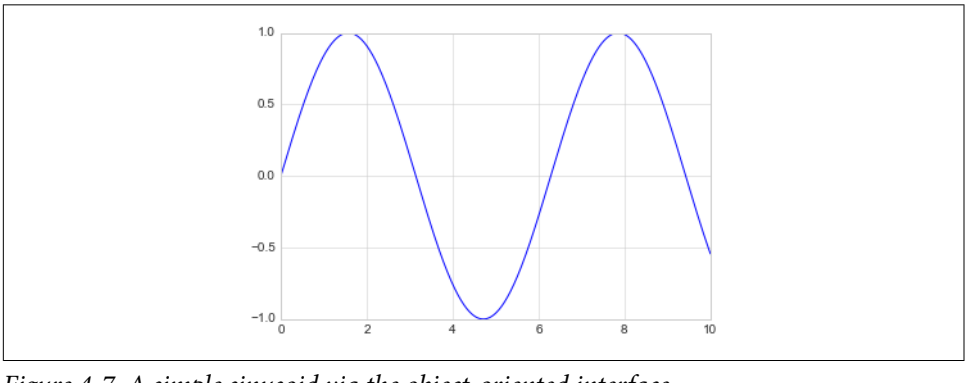

Figure 4-7. A simple sinusoid via the object-oriented interface

If we want to create a single figure with multiple lines, we can simply call the `plot` function multiple times (Figure 4-8):

```
In[5]: plt.plot(x, np.sin(x))
       plt.plot(x, np.cos(x));
```

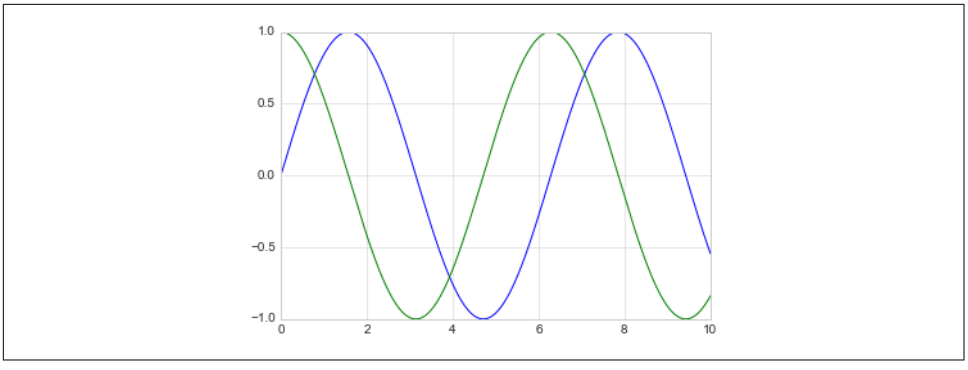

Figure 4-8. Over-plotting multiple lines

That's all there is to plotting simple functions in Matplotlib! We'll now 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 that 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 (Figure 4-9):

```
In[6]:
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 (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')        # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

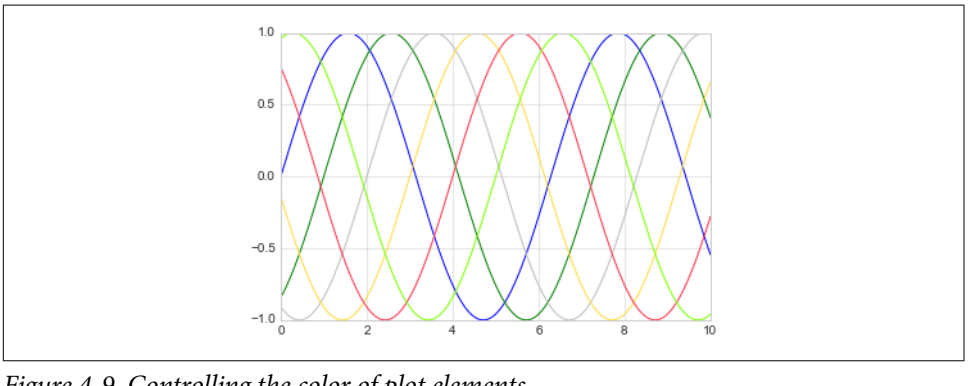

Figure 4-9. Controlling the color of plot elements