Suppose I augment this information with reported uncertainties: the current literature suggests a value of around 71 ± 2.5 (km/s)/Mpc, and my method has measured a value of 74 ± 5 (km/s)/Mpc. Now are the values consistent? That is a question that can be quantitatively answered.

In visualization of data and results, showing these errors effectively can make a plot convey much more complete information.

Basic Errorbars

A basic errorbar can be created with a single Matplotlib function call (Figure 4-27):

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

In[2]: x = np.linspace(0, 10, 50)
       dy = 0.8
       y = np.sin(x) + dy * np.random.randn(50)

       plt.errorbar(x, y, yerr=dy, fmt='.k');
```

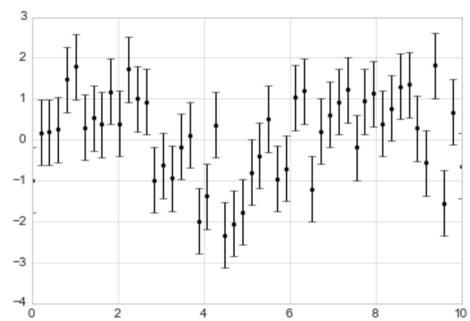

Figure 4-27. An errorbar example

Here the `fmt` is a format code controlling the appearance of lines and points, and has the same syntax as the shorthand used in `plt.plot`, outlined in "Simple Line Plots" on page 224 and "Simple Scatter Plots" on page 233.

In addition to these basic options, the `errorbar` function has many options to fine-tune the outputs. Using these additional options you can easily customize the aesthetics of your errorbar plot. I often find it helpful, especially in crowded plots, to make the errorbars lighter than the points themselves (Figure 4-28):

```
In[3]: plt.errorbar(x, y, yerr=dy, fmt='o', color='black',
                    ecolor='lightgray', elinewidth=3, capsize=0);
```

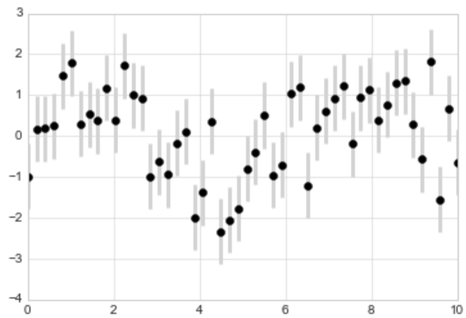

Figure 4-28. Customizing errorbars

In addition to these options, you can also specify horizontal errorbars (`xerr`), one-sided errorbars, and many other variants. For more information on the options available, refer to the docstring of `plt.errorbar`.

Continuous Errors

In some situations it is desirable to show errorbars on continuous quantities. Though Matplotlib does not have a built-in convenience routine for this type of application, it's relatively easy to combine primitives like `plt.plot` and `plt.fill_between` for a useful result.

Here we'll perform a simple *Gaussian process regression* (GPR), using the Scikit-Learn API (see "Introducing Scikit-Learn" on page 343 for details). This is a method of fitting a very flexible nonparametric function to data with a continuous measure of the uncertainty. We won't delve into the details of Gaussian process regression at this point, but will focus instead on how you might visualize such a continuous error measurement:

```
In[4]: from sklearn.gaussian_process import GaussianProcess

       # define the model and draw some data
       model = lambda x: x * np.sin(x)
       xdata = np.array([1, 3, 5, 6, 8])
       ydata = model(xdata)

       # Compute the Gaussian process fit
       gp = GaussianProcess(corr='cubic', theta0=1e-2, thetaL=1e-4, thetaU=1E-1,
                            random_start=100)
       gp.fit(xdata[:, np.newaxis], ydata)

       xfit = np.linspace(0, 10, 1000)
       yfit, MSE = gp.predict(xfit[:, np.newaxis], eval_MSE=True)
       dyfit = 2 * np.sqrt(MSE)  # 2*sigma ~ 95% confidence region
```