Usually it is not possible to create dense representations of sparse data (as they would not fit into memory), so we need to create sparse representations directly. Here is a way to create the same sparse matrix as before, using the COO format:

In[5]:

```
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("COO representation:\n{}".format(eye_coo))
```

Out[5]:

```
COO representation:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
```

More details on SciPy sparse matrices can be found in the SciPy Lecture Notes.

matplotlib

`matplotlib` is the primary scientific plotting library in Python. It provides functions for making publication-quality visualizations such as line charts, histograms, scatter plots, and so on. Visualizing your data and different aspects of your analysis can give you important insights, and we will be using `matplotlib` for all our visualizations. When working inside the Jupyter Notebook, you can show figures directly in the browser by using the `%matplotlib notebook` and `%matplotlib inline` commands. We recommend using `%matplotlib notebook`, which provides an interactive environment (though we are using `%matplotlib inline` to produce this book). For example, this code produces the plot in Figure 1-1:

In[6]:

```
%matplotlib inline
import matplotlib.pyplot as plt

# Generate a sequence of numbers from -10 to 10 with 100 steps in between
x = np.linspace(-10, 10, 100)
# Create a second array using sine
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```

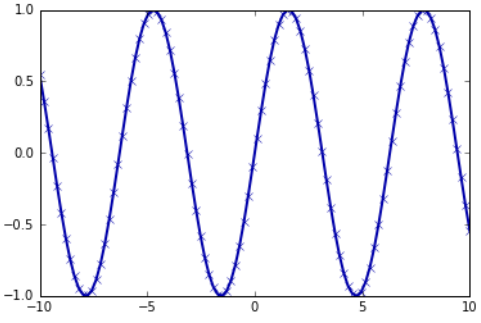

Figure 1-1. Simple line plot of the sine function using matplotlib

pandas

`pandas` is a Python library for data wrangling and analysis. It is built around a data structure called the `DataFrame` that is modeled after the R `DataFrame`. Simply put, a `pandas` `DataFrame` is a table, similar to an Excel spreadsheet. `pandas` provides a great range of methods to modify and operate on this table; in particular, it allows SQL-like queries and joins of tables. In contrast to NumPy, which requires that all entries in an array be of the same type, `pandas` allows each column to have a separate type (for example, integers, dates, floating-point numbers, and strings). Another valuable tool provided by `pandas` is its ability to ingest from a great variety of file formats and databases, like SQL, Excel files, and comma-separated values (CSV) files. Going into detail about the functionality of `pandas` is out of the scope of this book. However, *Python for Data Analysis* by Wes McKinney (O'Reilly, 2012) provides a great guide. Here is a small example of creating a `DataFrame` using a dictionary:

`In[7]:`

```
import pandas as pd

# create a simple dataset of people
data = {'Name': ["John", "Anna", "Peter", "Linda"],
        'Location' : ["New York", "Paris", "Berlin", "London"],
        'Age' : [24, 13, 53, 33]
       }

data_pandas = pd.DataFrame(data)
# IPython.display allows "pretty printing" of dataframes
# in the Jupyter notebook
display(data_pandas)
```