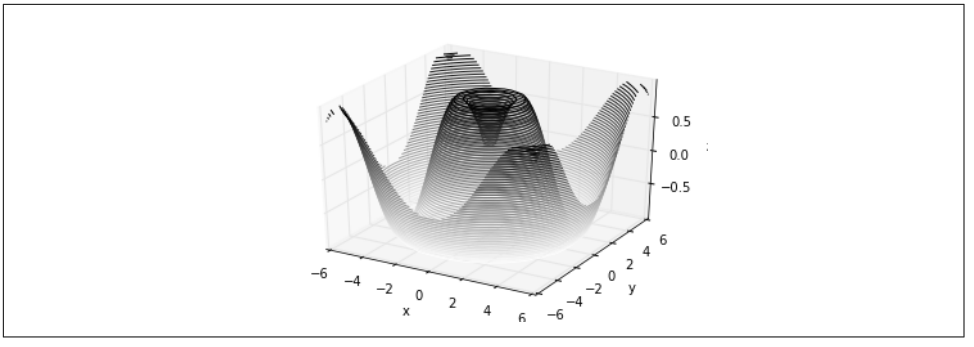

Figure 4-94. A three-dimensional contour plot

Sometimes the default viewing angle is not optimal, in which case we can use the `view_init` method to set the elevation and azimuthal angles. In this example (the result of which is shown in Figure 4-95), we'll use an elevation of 60 degrees (that is, 60 degrees above the *x*-*y* plane) and an azimuth of 35 degrees (that is, rotated 35 degrees counter-clockwise about the *z*-axis):

```
In[7]: ax.view_init(60, 35)
       fig
```

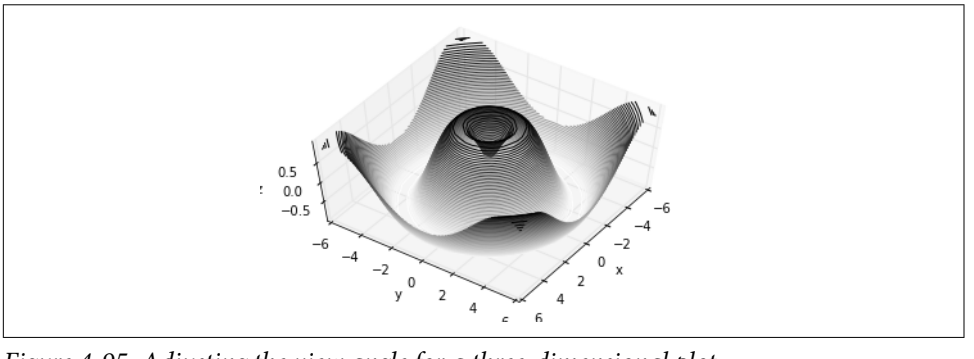

Figure 4-95. Adjusting the view angle for a three-dimensional plot

Again, note that we can accomplish this type of rotation interactively by clicking and dragging when using one of Matplotlib's interactive backends.

Wireframes and Surface Plots

Two other types of three-dimensional plots that work on gridded data are wireframes and surface plots. These take a grid of values and project it onto the specified three-dimensional surface, and can make the resulting three-dimensional forms quite easy to visualize. Here's an example using a wireframe (Figure 4-96):

```
In[8]: fig = plt.figure()
       ax = plt.axes(projection='3d')
       ax.plot_wireframe(X, Y, Z, color='black')
       ax.set_title('wireframe');
```

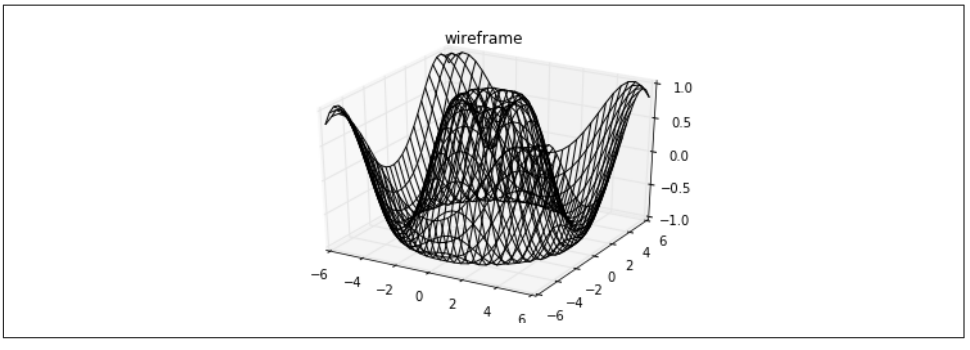

Figure 4-96. A wireframe plot

A surface plot is like a wireframe plot, but each face of the wireframe is a filled polygon. Adding a colormap to the filled polygons can aid perception of the topology of the surface being visualized (Figure 4-97):

```
In[9]: ax = plt.axes(projection='3d')
       ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                       cmap='viridis', edgecolor='none')
       ax.set_title('surface');
```

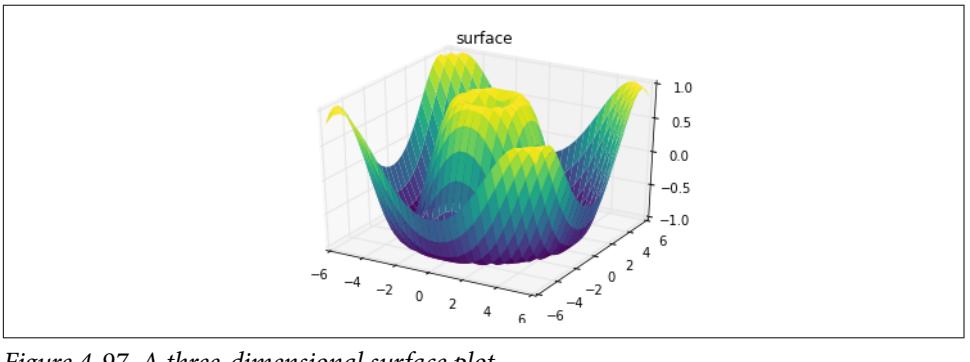

Figure 4-97. A three-dimensional surface plot

Note that though the grid of values for a surface plot needs to be two-dimensional, it need not be rectilinear. Here is an example of creating a partial polar grid, which when used with the `surface3D` plot can give us a slice into the function we're visualizing (Figure 4-98):

```
In[10]: r = np.linspace(0, 6, 20)
        theta = np.linspace(-0.9 * np.pi, 0.8 * np.pi, 40)
        r, theta = np.meshgrid(r, theta)

        X = r * np.sin(theta)
        Y = r * np.cos(theta)
        Z = f(X, Y)

        ax = plt.axes(projection='3d')
        ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                        cmap='viridis', edgecolor='none');
```

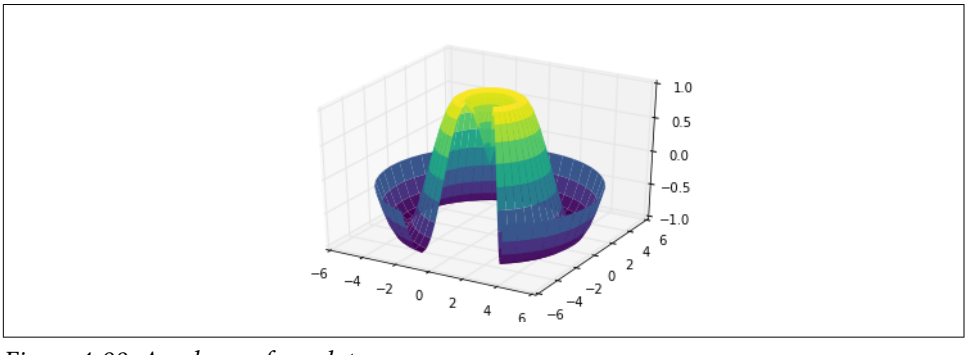

Figure 4-98. A polar surface plot

Surface Triangulations

For some applications, the evenly sampled grids required by the preceding routines are overly restrictive and inconvenient. In these situations, the triangulation-based plots can be very useful. What if rather than an even draw from a Cartesian or a polar grid, we instead have a set of random draws?

```
In[11]: theta = 2 * np.pi * np.random.random(1000)
        r = 6 * np.random.random(1000)
        x = np.ravel(r * np.sin(theta))
        y = np.ravel(r * np.cos(theta))
        z = f(x, y)
```

We could create a scatter plot of the points to get an idea of the surface we're sampling from (Figure 4-99):

```
In[12]: ax = plt.axes(projection='3d')
        ax.scatter(x, y, z, c=z, cmap='viridis', linewidth=0.5);
```