**Video #13: 3D Visualization**

In this video we'll explore MATLAB functions for visualizing three-dimensional curves, points, and surfaces, and along the way, you'll also learn more about colormaps. We'll begin by looking at 3D versions of plot and scatter, called plot3 and scatter3. These functions have three inputs at the beginning that are vectors of x, y, and z coordinates. The remaining inputs are the same as the 2D versions of these functions. The first example here displays a spiral curve - we begin by creating evenly spaced angles from 0 to 6-pi that will allow us to draw three loops for the spiral. The x and y coordinates are computed from the cosine and sine of the angles, which gives us points around a circle, and we'll use the angles directly for the z coordinates, so z will range from 0 to 6-pi which is about 20. We can add a linestyle and linewidth, like our 2D plots, and label the axes x, y, z. The second example creates a three-dimensional scatter plot, and here we generate random values for the coordinates. The expression rand(1,100) creates a row vector of 100 random numbers, each number is a floating point number in the range from 0 to 1. Let's run this section and view the results. By default, MATLAB draws the x axis along the right and automatically sets the range of values on the axis to fit the data (here from -1 to +1), the y axis is shown on the left (again from -1 to +1), and the z axis increases upwards, and as we figured, z values range from 0 to about 20. The next example is the 3D cloud of randomly positioned dots, with x,y,z coordinates all in the range from 0 to 1.

Suppose we want to graph a 3D function - mathematically we might write something like this, $z = x^2 + y^2$, and let's say we want to graph this function for x,y coordinates in the range from -4 to +4. In MATLAB, we'll do this in three steps: first, we construct matrices of the x and y coordinates for the range of values that we want, from -4 to +4 (meshgrid does this for us and in a moment, I'll describe what meshgrid does in more detail). Once we have the x,y coordinates, we can create the z coordinates with a statement that's a literal translation of our mathematical formula, x-squared plus y-squared, and the .^ means we're squaring each x and y coordinate. Finally we'll display the function as a wireframe mesh - let's see what this looks like. Here's our wireframe mesh, and MATLAB shows it in color using the default color map. The x,y coords range from -4 to +4 and z ranges from 0 to thirty-something.

Let's look at meshgrid more closely. Here's a picture of an x,y coordinate frame with the origin in the middle. The numbers in black are the x,y coordinates for a grid of locations shown with the black dots. Note, for the leftmost points, the x coordinates are all -4, for the next column of points, x is -2, and so on, to the rightmost points where x = 4. If we look along horizontal lines, along the top, the points all have a y coordinate of 4, then y = 2 in the next row, so on until the bottom row of points where y = -4. The numbers in red are the z values, sum of the squares of the x and y coordinates, e.g. in the corners, z is the sum of 4-squared + 4-squared, which is 32. What meshgrid does, is given a set of specific values for the x coordinates and a set of values for the y coordinates, it constructs the x,y coordinates for a grid of locations like this, and stores all the x coordinates in one matrix, and all the y coordinates in another matrix. For this example, the matrices would be 5 x 5 corresponding to this 5 x 5 grid of locations, and

we'd expect the matrix of x coordinates to have a column of -4s then a column of -2s, then 0s, 2s and 4s. For the matrix of y coordinates, we'd expect a row of 4s, then 2s, 0s, -2s, and -4s. Let's look at an example. The input is one vector, -4:2:4, which is the vector of values [-4 -2 0 2 4]. If we only provide one set of values, the same values are used for both x and y. I repeat the creation of z values. Let's run this and look at matrices of x,y,z coordinates. x has the columns of constant values that we expected, and y has the rows of constant values (the printout has the -4 at the top and +4 at the bottom). To compute the matrix of z values, each x coordinate is squared and added to the square of the corresponding y coordinate and the result is placed in the z matrix (e.g. $(-4)^2 + (-4)^2 = 32$). In the last statement here, two different vectors of values are provided - the first, which corresponds to four values [1 4 7 10], is used for the x coordinates and the second, which has 3 values [-2 0 2] is used for the y coordinates, and we can see this in the matrices returned.

There are some built-in functions, like peaks, that return matrices of x,y,z coordinates that can be used directly to display a 3D surface. The input specifies the number of distinct values to use for the coordinates. (A larger number will give us a smoother looking surface.) The mesh function has lots of properties you can specify, like EdgeColor, set here to 'k' that means black. And we can display a filled-in surface with the function surf. colorbar will display the colormap used to color the surface. Let's run this section. Here's the black wireframe mesh for the peaks function, and then the peaks function shown as a filled-in surface with the default colors shown in the colorbar on the right.

Let's now explore colormaps a bit more - you had some taste of colormaps in the animation you created of rising temperatures and Dow Jones Industrial averages. MATLAB has a number of different named colormaps that you can see in the documentation, and the colormap function allows us to change the colormap, for example, to spring colors or winter colors - let's see what they look like. You can see the full range of colors in the colorbars.

How are the colors actually represented and stored? You know that colors can be specified as combinations of red, green, and blue. Each of the named colormaps is actually a function that can take an input that specifies how many distinct colors you want to use. Here we'll use the jet colormap and sample 10 distinct colors over the range of colors in that colormap. The colormap function actually returns a matrix that stores the red, green, and blue values for each of the 10 colors in this case. Let's take a closer look at this matrix. When we draw a colorbar, we can also specify where to put it - southoutside means outside and below the graph. Let's run the section. Here are the 10 colors and their RGB values - the first column shows the amount of red in each color, second column is the amount of green, and third column is blue. The first two colors are pure blue, so red and green = 0 and the second blue is brighter. Then we start adding in more green to the color until we reach green = blue = 1, which is cyan. Then blue drops off while red is added until we reach red = green = 1, which is yellow. Finally, the green drops off so the color gets more orange.

Given that a colormap is just a set of RGB values stored in a 3-column matrix, we ought to be able to construct our own colormap, which we can do. Let's suppose we want to create a colormap with 10 shades of purple. Any shade of purple has equal amounts of red and blue. So we create a matrix of 10 rows x 3 columns, and in the red and blue columns, we can just put 10 evenly spaced values from 0 to 1, so they start out dark and get increasingly bright. We can call colormap with our own matrix of colors. Let's run this section and see our purples colormap and beautiful purple peaks.

A few more things about visualizing 3D surfaces. First, we can change the viewpoint for the scene. All the pictures we looked at so far were shown from a standard viewpoint relative to the x,y,z axes. We can change the viewpoint with the view function. The viewpoint is defined by two angles: the azimuth angle is the angular position around the z axis, and the elevation angle is the angular position up from the x,y plane. The default view has an azimuth angle of 37.5 degrees and elevation of 30 deg. If our peaks function is viewed from an azimuth angle of -75 deg and elevation of 45 deg, it would appear like this picture on the right. To get an idea of how the appearance changes for different viewpoints, we can display a 3D surface and set rotate3d on - let's see what that does. A little icon appears at the location of my mouse that I can drag to change the viewpoint. If I let go for a moment, you can see on the right, the view angles corresponding to the current viewpoint <move around>, and when you find a view that you like, you can click on the update code button to record the view in the code.

Before we go to the last topic, I'm going to Clear all Output. We can also show 3D surfaces with smooth shading - this gets us more deeply into the realm of computer graphics. I'll start with a new built-in shape, a sphere - this function returns three matrices with x,y,z coordinates for points on the surface of a sphere, and the input specifies how many distinct values of the coordinates we want, here I use 20 samples that gives us a pretty smooth surface. Axis equal makes our object really look like a sphere and not oval, I use the gray colormap, then specify shading. There are three examples here of the spherical surface with three kinds of shading - faceted, flat and interp (short for interpolated). Let's run the section to see the difference. First is faceted - it superimposes a mesh on the surface and within each facet of the mesh, the color is uniform. In the case of shading flat, the surface is also shown with uniformly colored facets, but the mesh is removed. Finally, with interpolated shading, the color varies smoothly across the surface. This takes longer to compute, but gives us a nicer looking surface.

In the final example, we'll go back to our peaks surface and use the copper colormap. axis tight just makes the range of x,y,z values hug the actual range of values more tightly. We'll use smooth shading, but add two more things, a light source in the direction of the vector I provide here and we'll make the surface shiny. Here's the result.

That's a taste of how to create 3D visualizations in MATLAB, and how to specify different colormaps and even create your own colormap. If you want to create 3D visualizations of your own, you can learn more about these topics in the MATLAB documentation.