PROFESSOR: OK. Here we are. We've got our data. We've managed to plot it. Now let's go about constructing a matrix which will enable us to fit a polynomial to this data. First of all, I need a matrix. Here is a way to construct a matrix. We put s equal to-- The function zeros is a function which produces a matrix with all zeros in it, and I want this matrix to have dimensions endpoints by endpoints. So this command should produce such a matrix, and lo and behold, it does.

Let me just clean up this display a little bit. At the moment, it's a little bit annoying that I'm plotting out absolutely everything that I have. This won't be too much trouble when I only have six points, but it might be in a lot of trouble if I had a lot more points.

Here's how to stop our MATLAB or Octave from printing out the results of a command. All you do is you put a semicolon at the end of the line, like that. Or, like that. If I make that change, so you hit and run again, it no longer prints out x and y. It's still doing the plotting, and it's printing out s as being zeros of endpoints endpoints.

That's not terribly interesting either, so I'm going to put a code on at the end of that line and prevent s from being printed out. Now, zeros is not what I want. What I want is a matrix which has the appropriate values to enable me to fit polynomials. And those values I'm going to generate by using a loop.

In general, it's not a great idea within MATLAB or Octave to use loops, but it's perfectly possible to do so. Loops are generated in the following way. Supposing I choose i to be the index of my loop, I can let i range from one to endpoints by writing 4i equals 1 Coulomb endpoints.

The end of that for loop is indicated by writing end. Actually, in Octave, it's endfor, but in MATLAB it's end, and so I'm just going to put end and that works in Octave as well. So that would loop over one index, but actually I want two indices. So I'm going to put another loop inside of the first loop that I had and I'm going to make its index j and I'm going to let it also run from one to endpoints.

And I need to end that loop, too. Let's do my indentation neatly so I can understand this loop when I come back and read it later, and now that's a loop which will loop over all of the indices of the matrix s. To refer to a particular element of the matrix s, I write s of i and j, where i and j are indices.

So I'm going to set the ij-th element of s equal to an appropriate value to use for polynomial fitting, and that value is x-- actually x I have to refer to by its index, so that's xi-- and I'm going to raise it to the power j minus 1.

So if j is 1, which is the lowest element, this would raise it to the 0 power, which would just give me the result of being x to the 0, which is 1, and for higher powers, it would go on appropriately. I don't want to print this out at every time that I calculate it, so I'm going to end the line with a semicolon. And then at the end, perhaps I do want to see s, so let me just write s, and that will cause s to be printed out.

So here I am. I run it again, and now s consists of this 6 by 6 matrix whose values, if I look at the top line, you can see that-- Well, first of all, let's look at the first column. The first column is all the values of j equals 1, and they're all x to the 0. And then this next column is x to the power 1, so you can see that it ranges from a 6 to 1.

The next column is x to the power 2, and so it runs from a smaller value up to 1. So of course all the values along the bottom row are 1, and the values along the top row are 1/6 to the power 0, 1, 2, 3, and so forth. So that's my s and that's what I'm going to use as my matrix.

Now, I can take the inverse of s by writing some new variable name. I'm going to call it sinv to remind me that it's the inverse of s, and put it equal to a function which is called inv, which stands for inverse, called s. So this command finds me the inverse of the matrix s. And when I run that, I find that s inverse is given by this matrix here, whose values will be very hard to guess.

Now, the way to find the coefficients of a polynomial fit to my original data are to put those coefficients equal to the matrix product of the inverse of s, which is sinv times the y values, which we learned about in the lecture. So that would give me the coefficients. When I run that, I find out what the coefficients, or the different powers, of x are.

So in other words, these coefficients-- the first line is the coefficient of the power x to the power 0, the next one is x to the power 1, and so on. So I've actually found the coefficients of the fit. For the moment, it's not obvious whether I've done it correctly. I could certainly plot out the values of this fit at various places, and that's what I'm going to do.

I'm going to plot it actually with more points than I started with. So let me define a new parameter, np2. I might make it equal to, let's say, 40. I could then have a second variable, x2, equal to something running from 1 to np2 transposed, divided by np2. So that will be an x array running from a small value to 1 with, in this case, 40 steps.

I'll set, for now, y2 equal to x2. That's just created an array that is of the same length as x, and is actually equal to x, but I'm actually going to end up setting y to be something else in a minute. And let me, for now, put fj, a new variable, equal to the c that I've just calculated.

This won't do anything interesting, because I've commented those out, and so when I run it, I just get the same result that I had before. So I haven't actually done anything. Now I'm going to have another couple of nested for loops. I'm going to say for j equals 1,np2, and for i equals 1:np2.

Let's f set the value of fj, which is a matrix of the vector, strictly speaking, column vector of the same length as c. Let's set that the i-th element of that equal to the x value, which is x2, because this is the second array that I'm creating, evaluated at parameter j to the power j minus 1. So that's actually evaluating x2 to the appropriate power.

So it shouldn't have been j minus 1. It should have been i minus 1. So this is x2 j to the i minus 1. Again, I don't want to print all of those out, so I'm going to put a semicolon at the end. So that's calculated by fj value, and now I'm going to put y2 of j equal to-- Well, what I basically want to do is form the dot product between c, the coefficients, and the value of fj that I've just calculated. So I want c1 times fj1 plus c2 times fj2 and so forth.

One way of doing that-- there are various different ways of doing it-- is to take the transpose of c. C remember, is a column vector. Let's transpose it. That's going to give me a row vector. I can then take the product of that row vector with the column vector fj. And so taking to the matrix product of a column vector times a row vector is equivalent to taking the dot product of two vectors.

So that has produced a value y2 of j, which is equal to the sum of the coefficients times the values of x to the appropriate power. So that's what I want to form in that matrix. Again, I'm--

It seems as though I've made a mistake. And I'm not quite sure what I did wrong, but I need to find out. So let's have a look. It's saying that I made my mistake at line 29 of column 8. 29 is this, and it's saying that I have a 1 by 6, and op2 is 40 by 1. I don't know why that is 40 by 1. I'm a bit surprised by it, because-- Ah. I see what I did wrong. This should have been endpoints.

So what I did was, this inner for loop shouldn't have been over all of the j values. The j values refer to the length of the new vectors that I produced. It should've been over the shorter version. So let's see. Let's save that and hope that's corrected my error. Yes. Lo and behold, it has. I haven't actually done anything to show what the result is, but at least now I've got the result.

Now what I want to do is, I want to plot these two new matrices, x2 and y2, and actually I'd like to overplot them on my old plot. If I just say plot like this, save this and plot it again, that isn't what happens. Instead, what happens is just the curve that I've just calculated is plotted, and my previous plot is wiped out.

What I can do to prevent that happening is to say, hold on. Only MATLAB slash Octave would have something like hold on as a command. But anyway, hold on basically says, retain the data that you've already got in this plot and add some more data on. So let's try this.

So now what we see is the data that I've plotted out in my first plot, which was up here, is held and the second plot is plotted. Notice that this fit does indeed go through the six points that I originally retained, and yet, that line actually does some funny things. Particularly at the end here, it goes negative.

There's another trick that one can do in plotting, and one can say things like axis manual, and I think that, if I'm lucky, that will fix this plot. No, it won't. Never mind. Anyway, there's a way-- and I'll show it later-- to prevent the plot rescaling and the required command is axis manual.

But at the moment, I think, before I do it, I've got to say hold off. And then if I re-run it, I will find that lo and behold, I've got a plot of the six points and the line that I fitted, which is a polynomial fitted to those lines. And lo and behold, it fits.