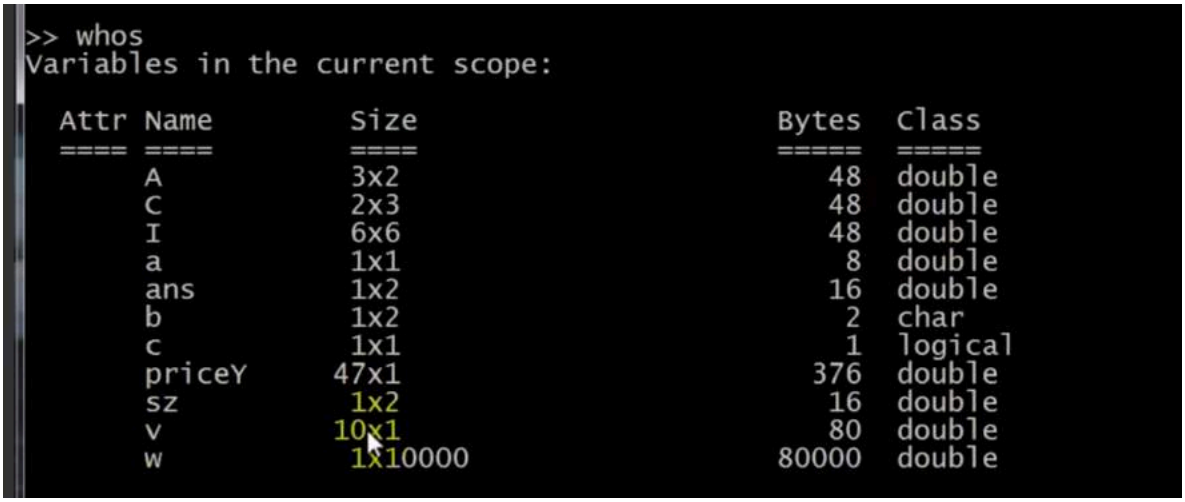
MACHINE LEARNING-3

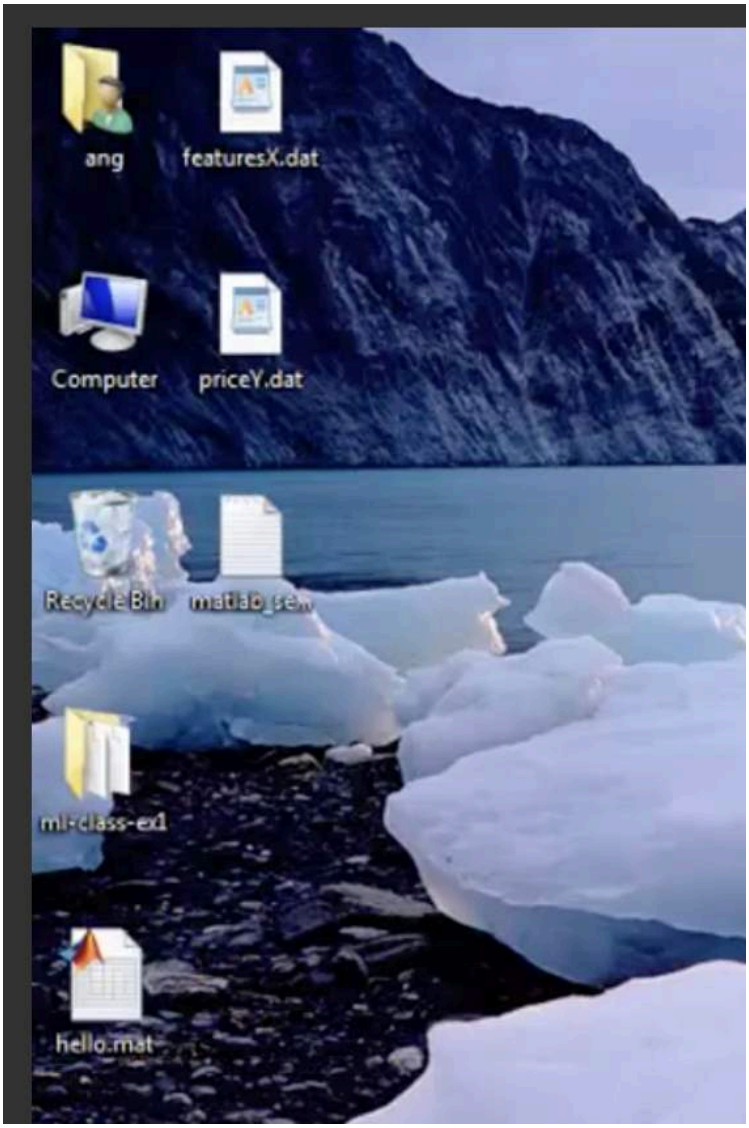
And how do we save data? Let's see. Let's take the variable V and say that it's a price Y 1 colon 10. This sets V to be the first 10 elements of vector Y. So let's type who or whose. Whereas Y was a 47 by 1 vector. V is now 10 by 1. B equals price Y, one column ten that sets it to the just the first ten elements of Y.



Let's say I wanna save this to date to disc the command save, hello.mat V. This will save the variable V into a file called hello.mat. So let's do that. And now a file has appeared on my Desktop, you know, called Hello.mat. I happen to have MATLAB installed in this window, which is why, you know, this icon looks like this because Windows is recognized as it's a MATLAB file, but don't worry

about it if this file looks like it has a different icon on your machine and let's say I clear all my variables. So, if you type clear without anything then this actually deletes all of the variables in your workspace. So there's now nothing left in the workspace. And if I load hello.mat, I can now load back my variable v, which is the data that I previously saved into the hello.mat file.

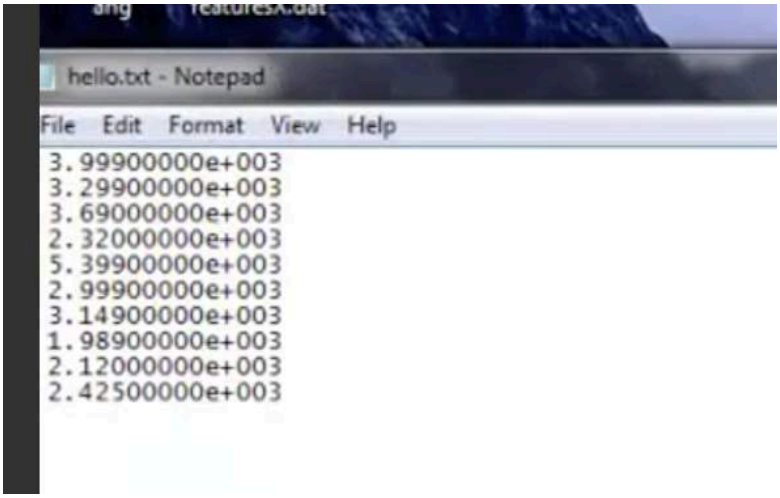
```
>>
>> save hello.mat v;
>> clear
>> whos
>> who
>> load hello.mat
>> whos
Variables in the current scope:

  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
       v          10x1                        80  double

Total is 10 elements using 80 bytes
```



```
>> save hello.mat v;
>> clear
>> whos
>> who
>> load hello.mat
>> whos
Variables in the current scope:

  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
       v           10x1                        80  double

Total is 10 elements using 80 bytes
```

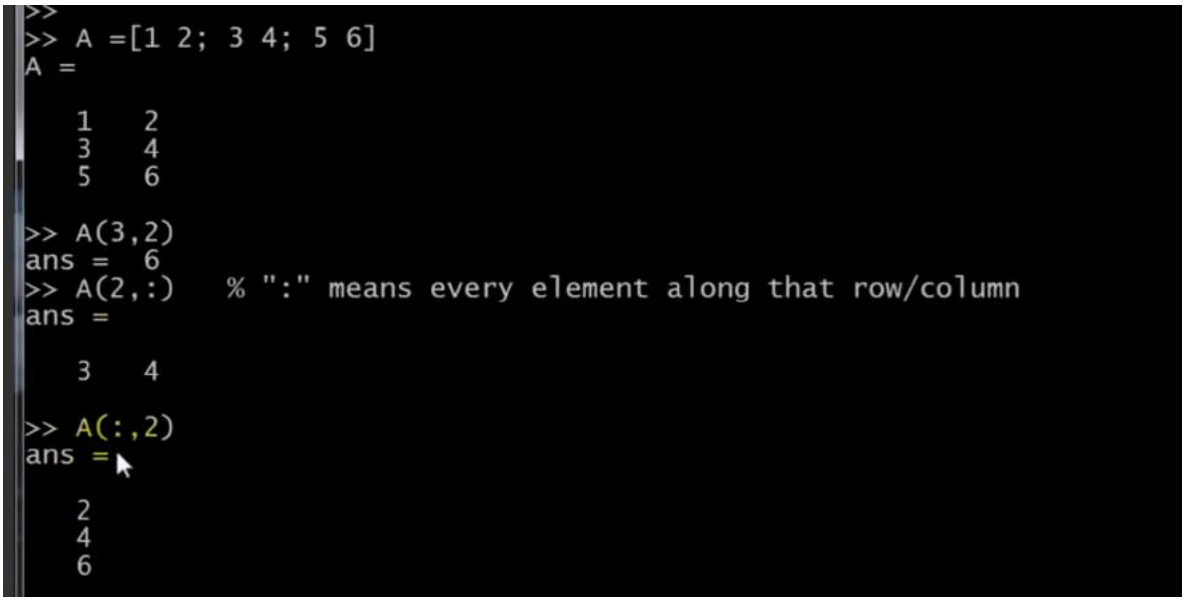
So, hello.mat, what we did just now to save hello.mat to view, this save the data in a binary format, a somewhat more compressed binary format. So if v is a lot of data, this, you know, will be somewhat more compressing. Will take off less the space. If you want to save your data in a human readable format then you type save hello.text the variable v and then -ascii. So, this will save it as a text or as ascii format of text. And now, once I've done that, I have this file. Hello.text has just appeared on my desktop, and if I open this up, we see that this is a text file with my data saved away.

```
>> save hello.txt v -ascii   % save as text (ASCII)
>>
```



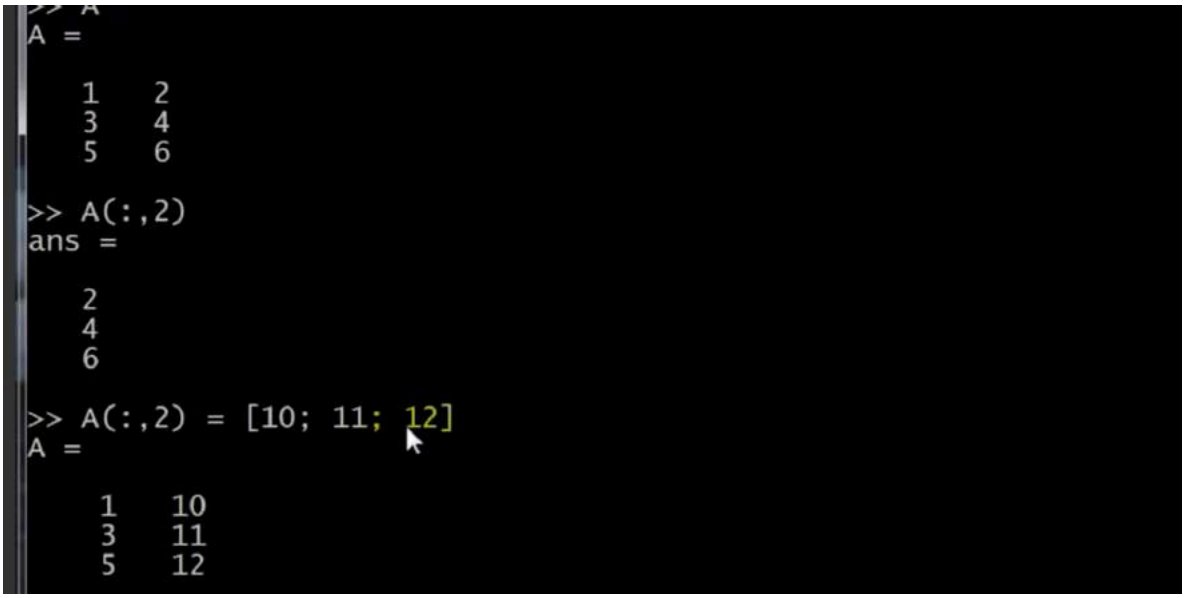
So that's how you load and save data. Now let's talk a bit about how to manipulate data. Let's set a equals to that matrix again so is my three by two matrix. So as indexing. So type A 3, 2. This indexes into the 3, 2 elements of the matrix A. So, this is what, you know, in normally, we will write this as a subscript 3, 2 or A subscript, you know, 3, 2 and so that's the element and third row and second column of A which is the element of six. I can also type A to

comma colon to fetch everything in the second row. So, the colon means every element along that row or column. So, a of 2 comma colon is this second row of a. Right. And similarly, if I do a colon comma 2 then this means get everything in the second column of A. So, this gives me 2 4 6. Right this means of A. everything, second column. So, this is my second column A, which is 2 4 6.



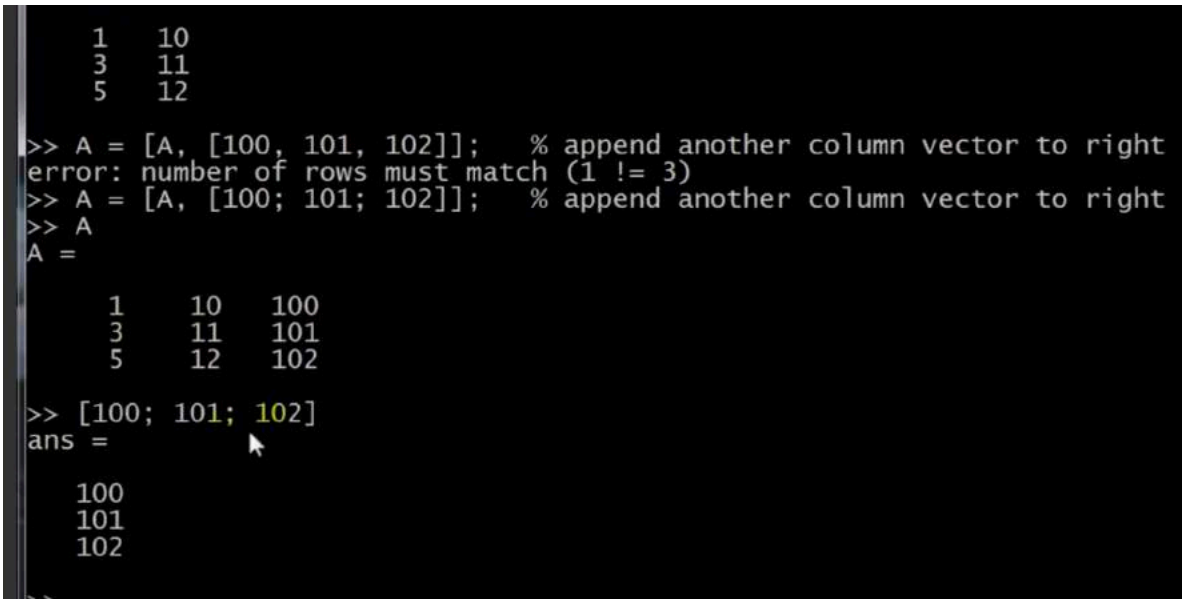
Now, you can also use somewhat most of the sophisticated index in the operations. So So, we just click each of an example. You do this maybe less often, but let me do this A 1 3 comma colon. This means get all of the elements of A who's first indexes one or three. This means I get everything from the first and third rows of A and from all columns. So, this was the matrix A and so A 1 3 comma colon means get everything from the first row and from the second row and from the third row and the colon means, you know, one both of first and the second columns and so this gives me this 1 2 5 6. Although, you use the source of more subscript index operations maybe somewhat less often.

To show you what else we can do. Here's the A matrix and this source A colon, to give me the second column. You can also use this to do assignments. So I can take the second column of A and assign that to 10, 11, 12, and if I do that I'm now, you know, taking the second column of a and I'm assigning this column vector 10, 11, 12 to it. So, now a is this matrix that's 1, 3, 5. And the second column has been replaced by 10, 11, 12.

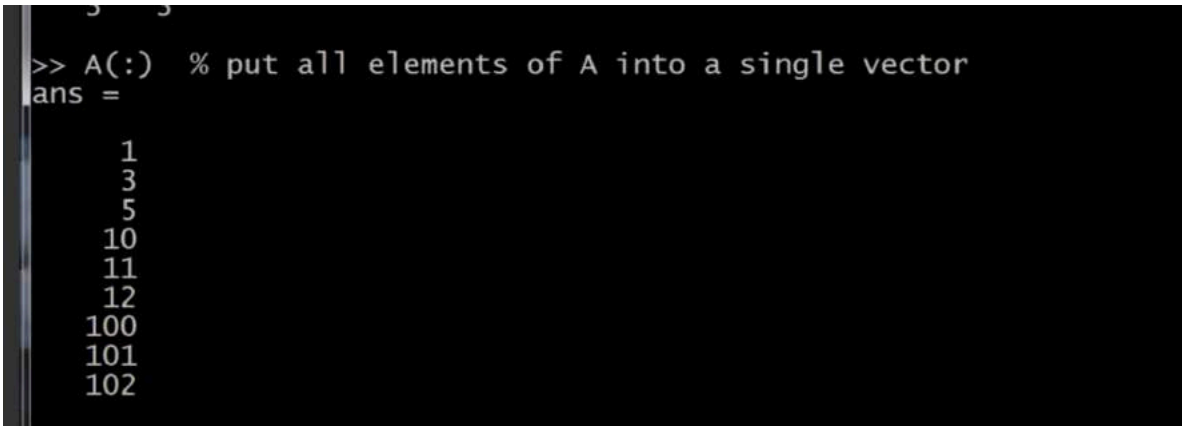


And here's another operation. Let's set A to be equal to A comma 100, 101, 102

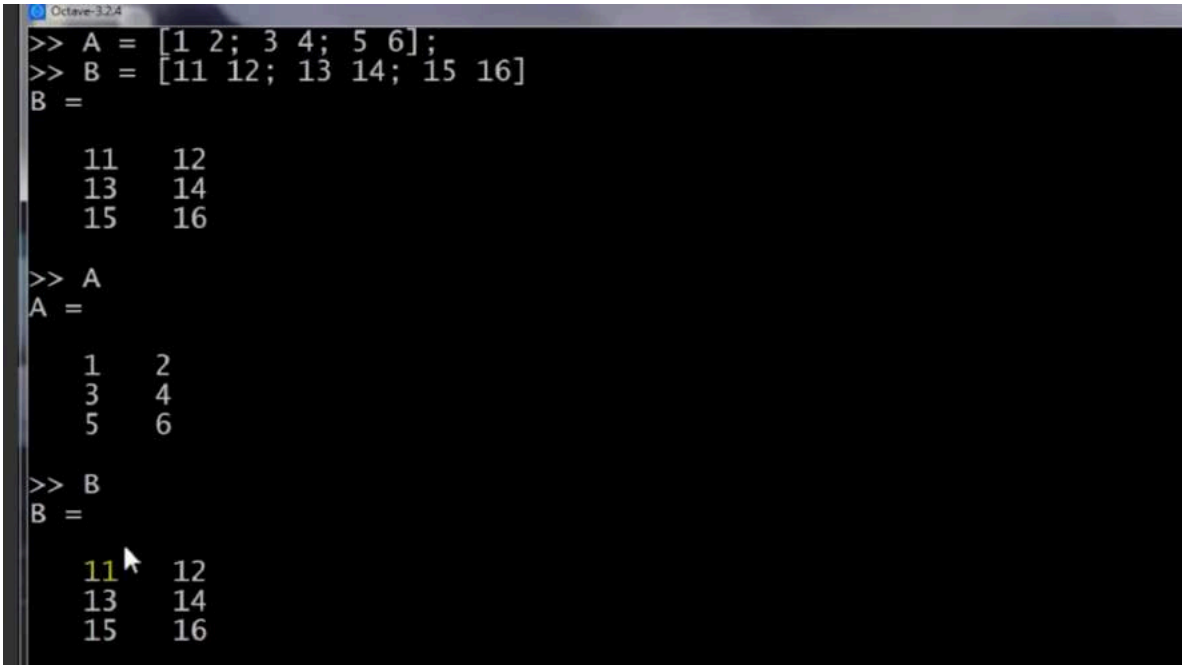
like so and what this will do is depend another column vector to the right. So, now, oops. I think I made a little mistake. Should have put semicolons there and now A is equals to this. Okay? I hope that makes sense. So this 100, 101, 102. This is a column vector and what we did was we set A, take A and set it to the original definition. And then we put that column vector to the right and so, we ended up taking the matrix A and--which was these six elements on the left. So we took matrix A and we appended another column vector to the right; which is now why A is a three by three matrix that looks like that.

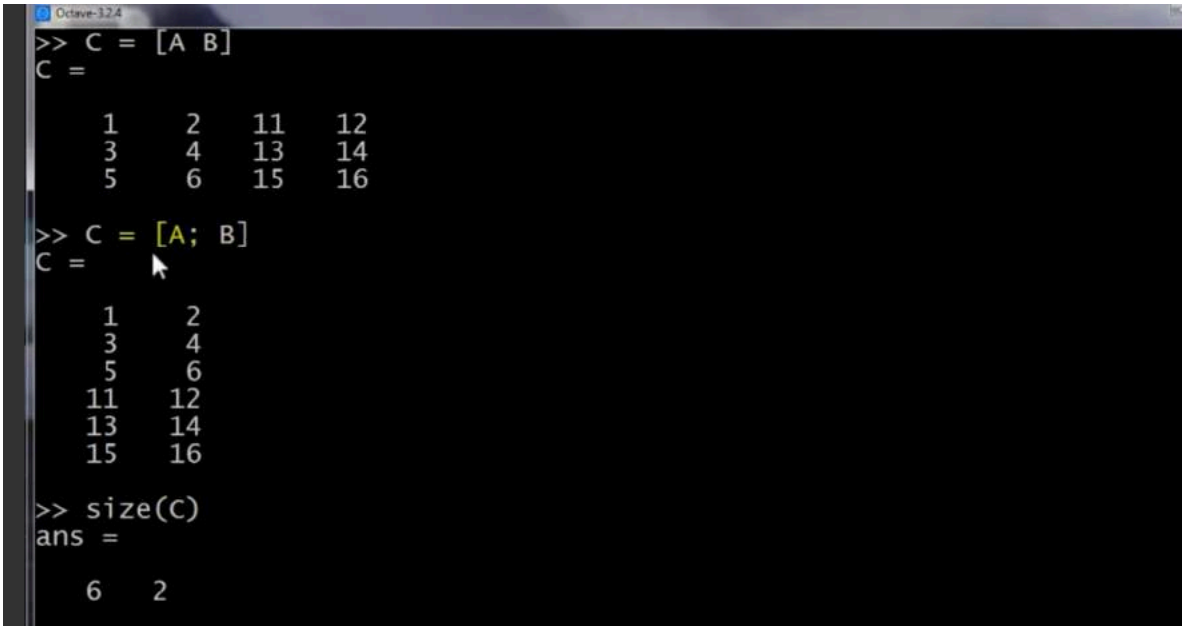


And finally, one neat trick that I sometimes use if you do just a and just a colon like so. This is a somewhat special case syntax. What this means is that put all elements with A into a single column vector and this gives me a 9 by 1 vector. They adjust the other ones are combined together.



Just a couple more examples. Let's see. Let's say I set A to be equal to 123456, okay? And let's say I set a B to B equal to 11, 12, 13, 14, 15, 16. I can create a new matrix C as A B. This just means my Matrix A. Here's my Matrix B and I've set C to be equal to AB. What I'm doing is I'm taking these two matrices and just concatenating onto each other. So the left, matrix A on the left. And I have the matrix B on the right. And that's how I formed this matrix C by putting them together. I can also do C equals A semicolon B. The semi colon notation means that I go put the next thing at the bottom. So, I'll do is a equals semicolon B. It also puts the matrices A and B together except that it now puts them on top of each other. so now I have A on top and B at the bottom and C here is now in 6 by 2 matrix. So, just say the semicolon thing usually means, you know, go to the next line. So, C is comprised by a and then go to the bottom of that and then put b in the bottom and by the way, this A B is the same as A, B and so you know, either of these gives you the same result.

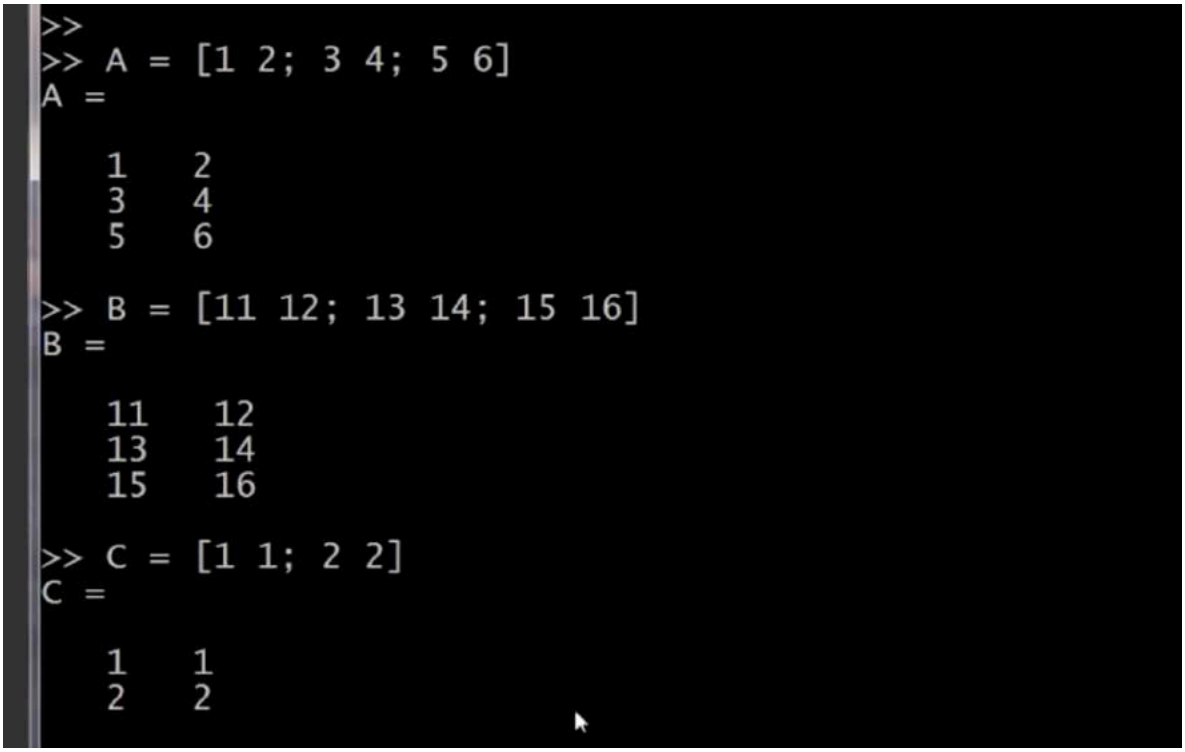




So, with that, hopefully you now know how to construct matrices and hopefully starts to show you some of the commands that you use to quickly put together matrices and take matrices and, you know, slam them together to form bigger matrices, and with just a few lines of code, Octave is very convenient in terms of how quickly we can assemble complex matrices and move data around. So that's it for moving data around. In the next video we'll start to talk about how to actually do complex computations on this, on our data. So, hopefully that gives you a sense of how, with just a few commands, you can very quickly move data around in Octave. You know, you load and save vectors and matrices, load and save data, put together matrices to create bigger matrices, index into or select specific elements on the matrices. I know I went through a lot of commands, so I think the best thing for you to do is afterward, to look at the transcript of the things I was typing. You know, look at it. Look at the coursework site and download the transcript of the session from there and look through the transcript and type some of those commands into Octave yourself and start to play with these commands and get it to work. And obviously, you know, there's no point at all to try to memorize all these commands. It's just, but what you should do is, hopefully from this video you have gotten a sense of the sorts of things you can do. So that when later on when you are trying to program a learning algorithms yourself, if you are trying to find a specific command that maybe you think Octave can do because you think you might have seen it here, you should refer to the transcript of the session and look through that in order to find the commands you wanna use. So, that's it for moving data around and in the next video what I'd like to do is start to tell you how to actually do complex computations on our data, and how to compute on the data, and actually start to implement learning algorithms.

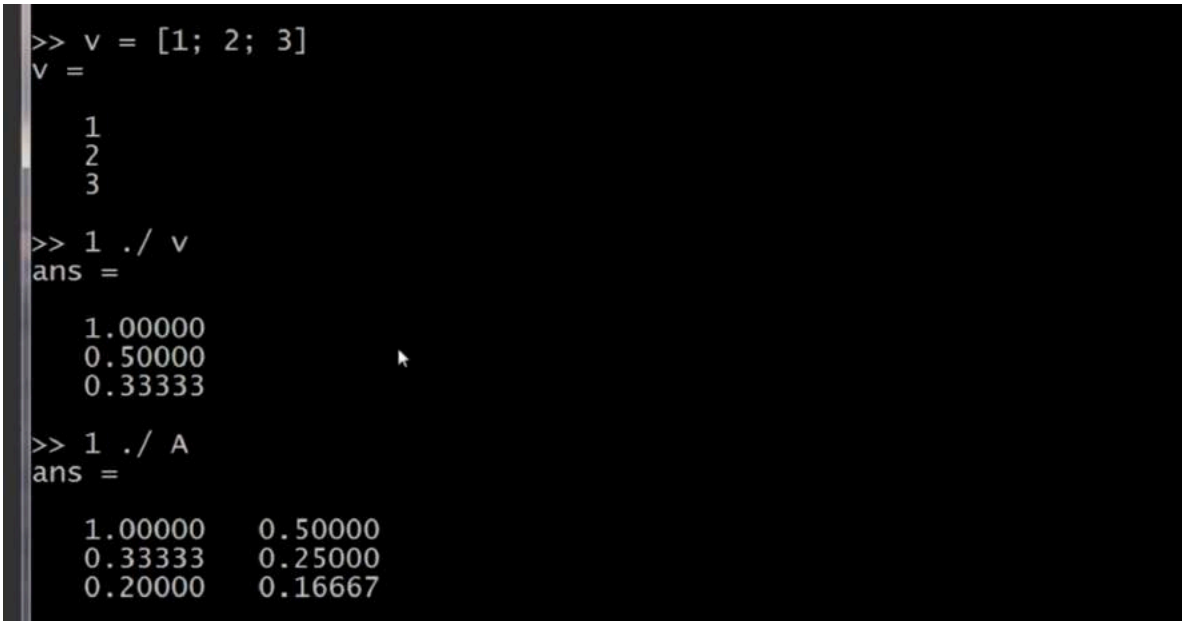
Computing on Data

Now that you know how to load and save data in Octave, put your data into matrices and so on. In this video, I'd like to show you how to do computational operations on data. And later on, we'll be using these source of computational operations to implement our learning algorithms. Let's get started. Here's my Octave window. Let me just quickly initialize some variables to use for our example. So set A to be a three by two matrix, and set B to a three by two matrix, and let's set C to a two by two matrix like so.

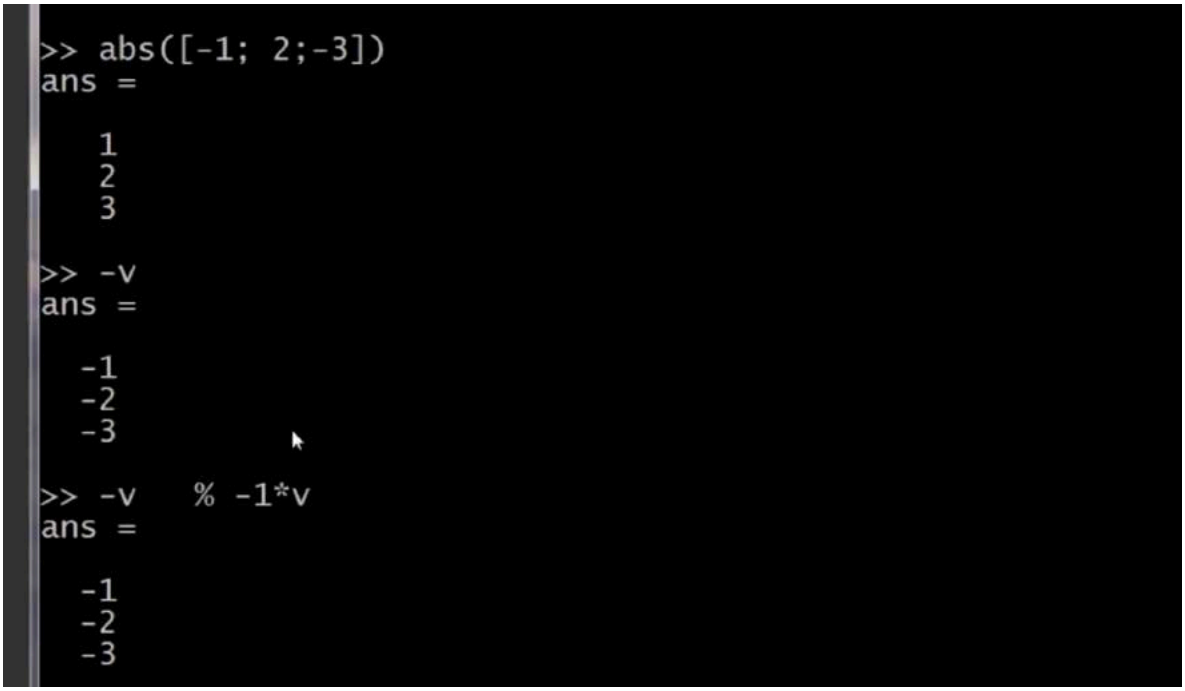


Now let's say I want to multiply two of my matrices. So let's say I want to compute A*C, I just type A*C, so it's a three by two matrix times a two by two matrix, this gives me this three by two matrix. You can also do element wise operations and do A.* B and what this will do is it'll take each element of A and multiply it by the corresponding elements B, so that's A, that's B, that's A .* B. So for example, the first element gives 1 times 11, which gives 11. The second element gives 2 time 12 Which gives 24, and so on. So this is element-wise multiplication of two matrices. And in general, the period tends to, is usually used to denote element-wise operations in Octave. So here's a matrix A, and if I do A .^ 2, this gives me the element wise squaring of A. So 1 squared is 1, 2 squared is 4, and so on.

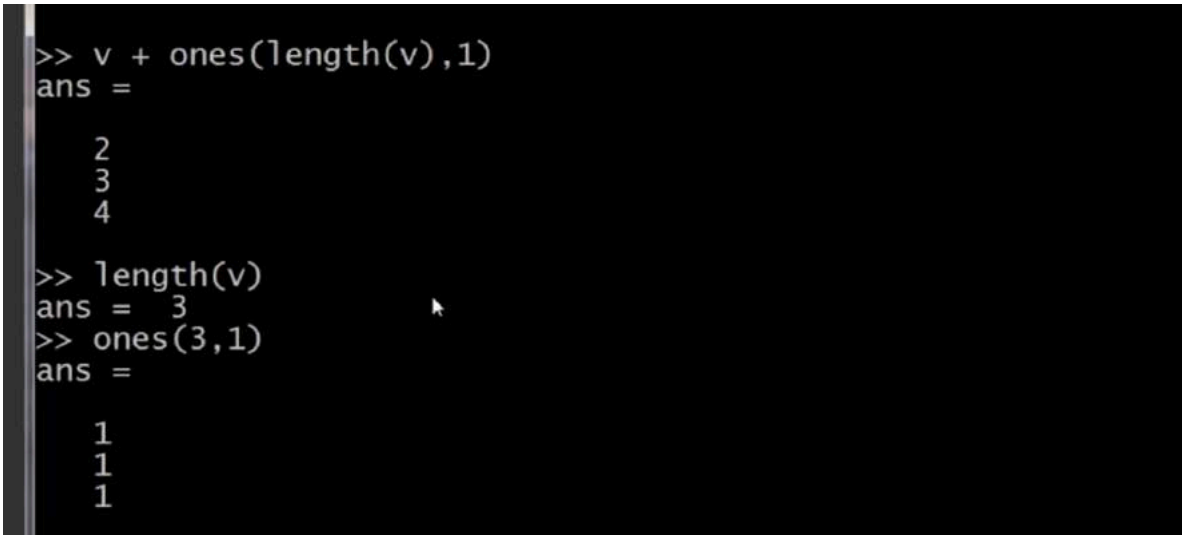
Let's set v as a vector. Let's set v as one, two, three as a column vector. You can also do one dot over v to do the element-wise reciprocal of v, so this gives me one over one, one over two, and one over three, and this is where I do the matrices, so one dot over a gives me the element wise inverse of a. And once again, the period here gives us a clue that this an element-wise operation.



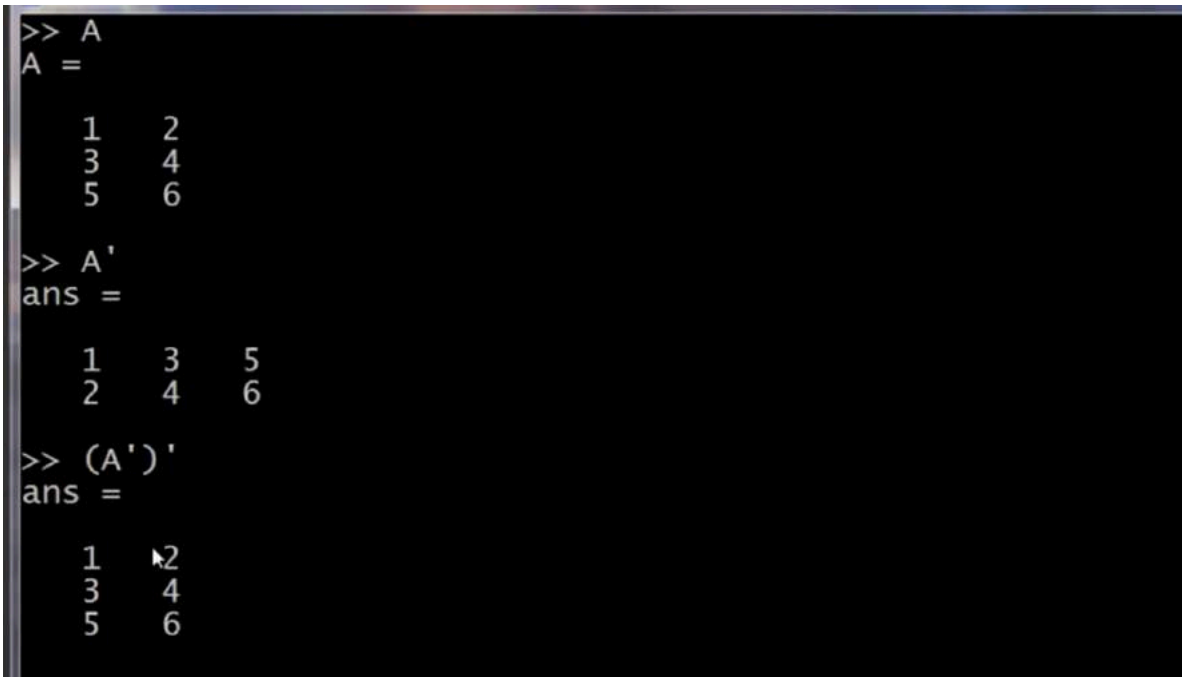
We can also do things like log(v), this is a element-wise logarithm of the v E to the V is base E exponentiation of these elements, so this is E, this is E squared EQ, because this was V, and I can also do abs V to take the element-wise absolute value of V. So here, V was our positive, abs, minus one, two minus 3, the element-wise absolute value gives me back these non-negative values. And negative v gives me the minus of v. This is the same as negative one times v, but usually you just write negative v instead of -1*v.



And what else can you do? Here's another neat trick. So, let's see. Let's say I want to take v an increment each of its elements by one. Well one way to do it is by constructing a three by one vector that's all ones and adding that to v. So if I do that, this increments v by from 1, 2, 3 to 2, 3, 4. The way I did that was, length(v) is 3, so ones(length(v),1), this is ones of 3 by 1, so that's ones(3,1) on the right and what I did was v plus ones v by one, which is adding this vector of our ones to v, and so this increments v by one, and another simpler way to do that is to type v plus one. So she has v, and v plus one also means to add one element wise to each of my elements of v.

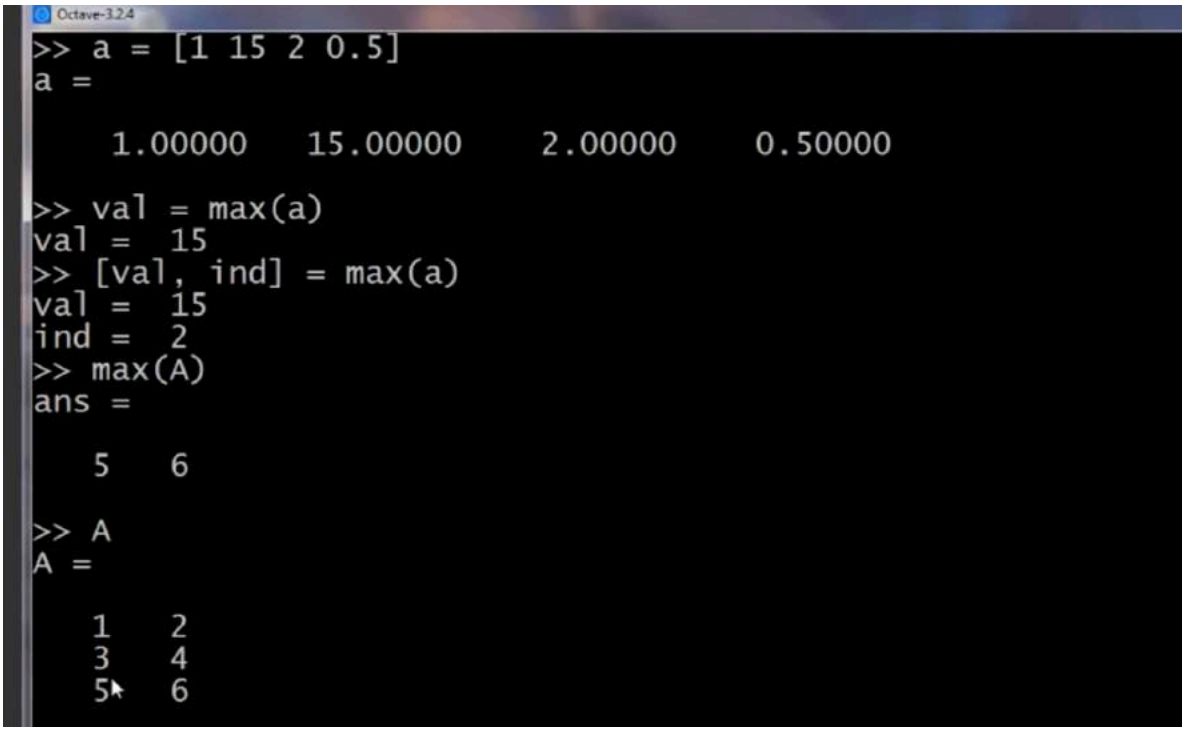


Now, let's talk about more operations. So here's my matrix A, if you want to buy A transposed, the way to do that is to write A prime, that's the apostrophe symbol, it's the left quote, so it's on your keyboard, you have a left quote and a right quote. So this is actually the standard quotation mark. Just type A transpose, this gives me the transpose of my matrix A. And, of course, A transpose, if I transpose that again, then I should get back my matrix A.

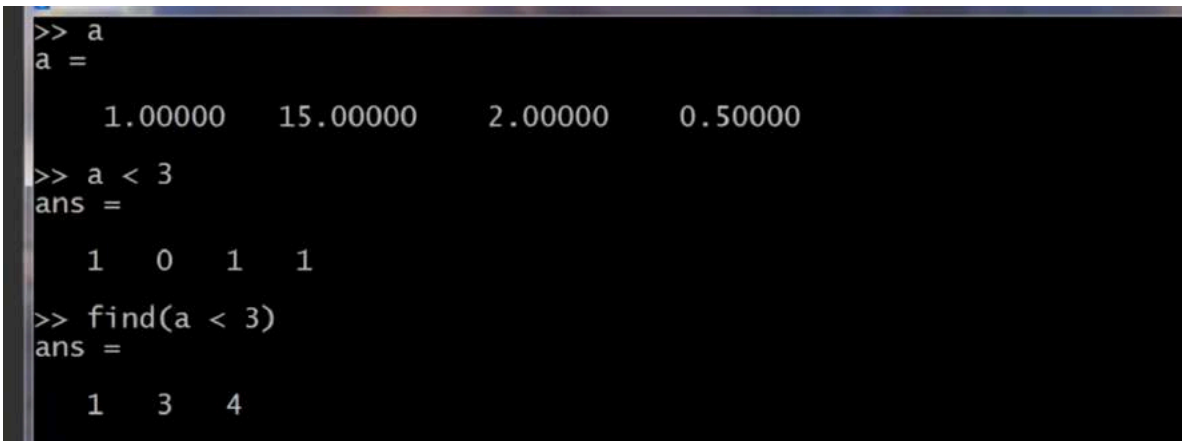


Some more useful functions. Let's say lower case a is 1 15 2 0.5, so it's 1 by 4 matrix. Let's say val equals max of A this returns the maximum value of A which in this case is 15 and I can do val, ind max(a) and this returns val and ind which are going to be the maximum value of A which is 15, as well as the index. So it was the element number two of A that was 15 so ind is my index

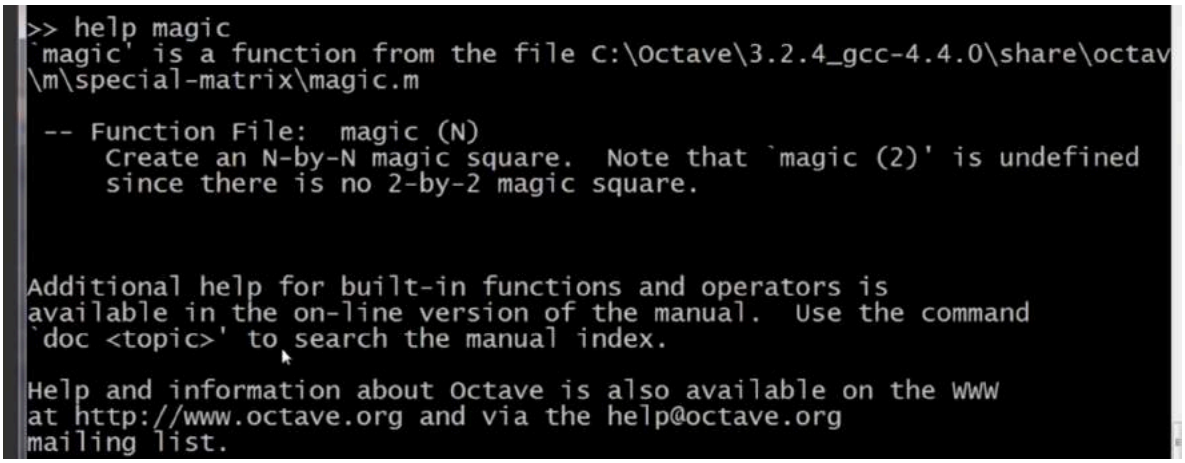
into this. Just as a warning, if you do max(A), where A is a matrix, what this does is this actually does the column wise maximum. But say a little more about this in a second.



Still using this example that there for lowercase a. If I do a < 3, this does the element wise operation. Element wise comparison, so the first element of A is less than three so this one. Second element of A is not less than three so this value says zero cuz it's false. The third and fourth elements of A are less than three, so that's just 1 1. So that's the element-wise comparison of all four elements of the variable a < 3. And it returns true or false depending on whether or not there's less than three. Now, if I do find(a < 3), this will tell me which are the elements of a, the variable a, that are less than 3, and in this case, the first, third and fourth elements are less than 3.



For our next example, let me set a to be equal to magic(3). The magic function returns, let's type help magic. The magic function returns these matrices called magic squares. They have this, you know, mathematical property that all of their rows and columns and diagonals sum up to the same thing. So, you know, it's not actually useful for machine learning as far as I know, but I'm just using this as a convenient way to generate a three by three matrix. And these magic squares have the property that each row, each column, and the diagonals all add up to the same thing, so it's kind of a mathematical construct. I use this magic function only when I'm doing demos or when I'm teaching octave like those in, I don't actually use it for any useful machine learning application.



But let's see, if I type RC = find(A > 7) this finds All the elements of A that are greater than equal to seven, and so r, c stands for row and column. So the 1,1 element is greater than 7, the 3,2 element is greater than 7, and the 2,3 element is greater than 7. So let's see. The 2,3 element, for example, is A(2,3), is 7 is this element out here, and that is indeed greater than equal seven. By the way, I actually don't even memorize myself what these find functions do and what all of these things do myself. And whenever I use the find function, sometimes I forget myself exactly what it does, and now I would type help find to look at the document.

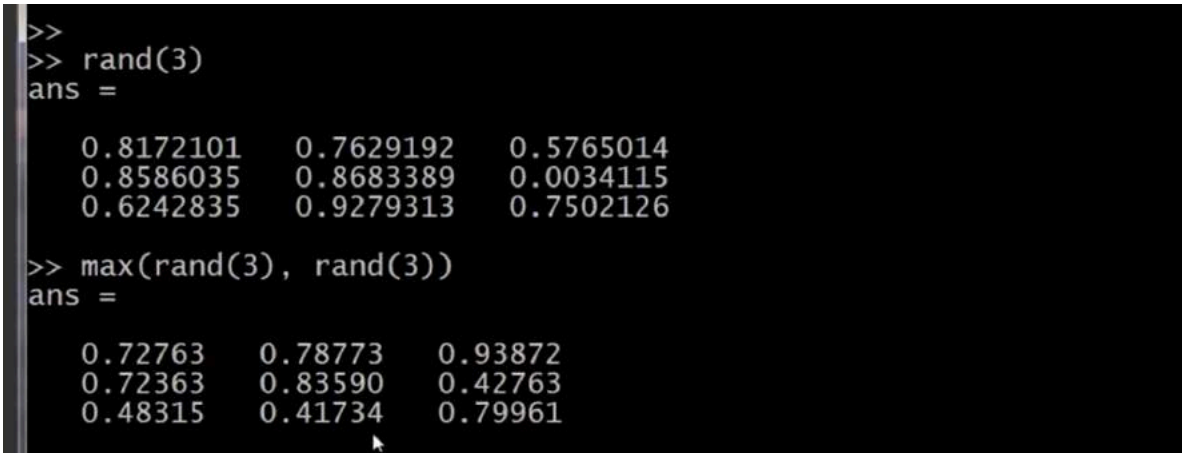
```
>> A = magic(3)
A =

    8    1    6
    3    5    7
    4    9    2

>>
>> [r,c] = find(A >= 7)
r =

    1
    3
    2

c =

    1
    2
    3
```

Okay, just two more things that I'll quickly show you. One is the sum function, so here's my a, and then type sum(a). This adds up all the elements of a, and if I want to multiply them together, I type prod(a) prod sends the product, and this returns the product of these four elements of A. Floor(a) rounds down these elements of A, so 0.5 gets rounded down to 0. And ceil, or ceiling(A) gets rounded up to the nearest integer, so 0.5 gets rounded up to 1.

```
a =

    1.00000   15.00000    2.00000    0.50000

>> sum(a)
ans =  18.500
>> prod(a)
ans =  15
>> floor(a)
ans =

    1   15    2    0

>> ceil(a)
ans =

    1   15    2    1
```

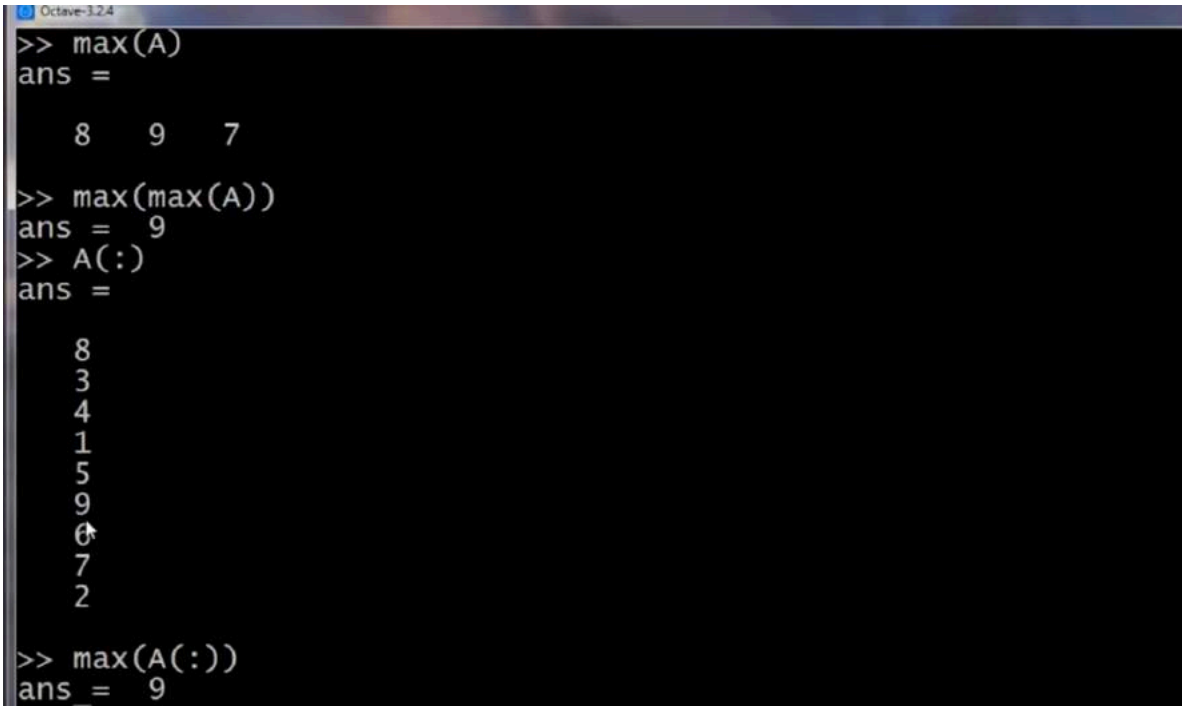
You can also, let's see. Let me type rand(3), this generates a three by three matrix. If i type max(rand(3), what this does is it takes the element-wise maximum of 3 random 3 by 3 matrices. So you notice all of these numbers

tend to be a bit on the large side because each of these is actually the max of a element wise max of two randomly generated matrices.

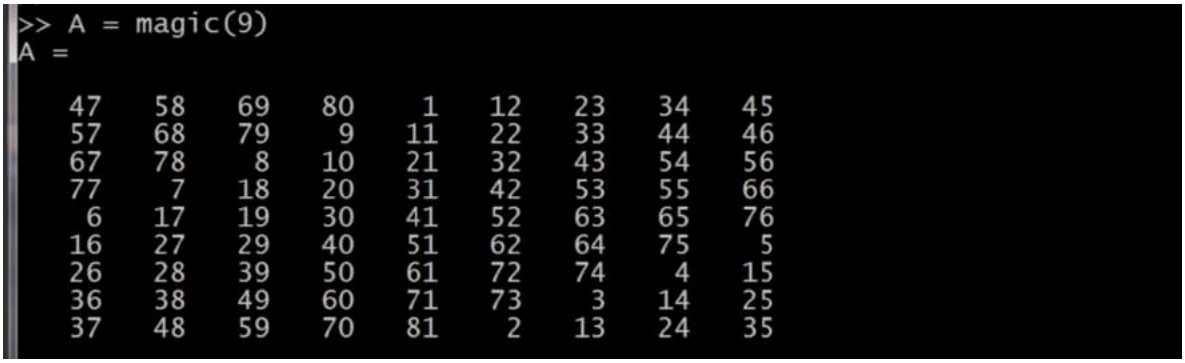


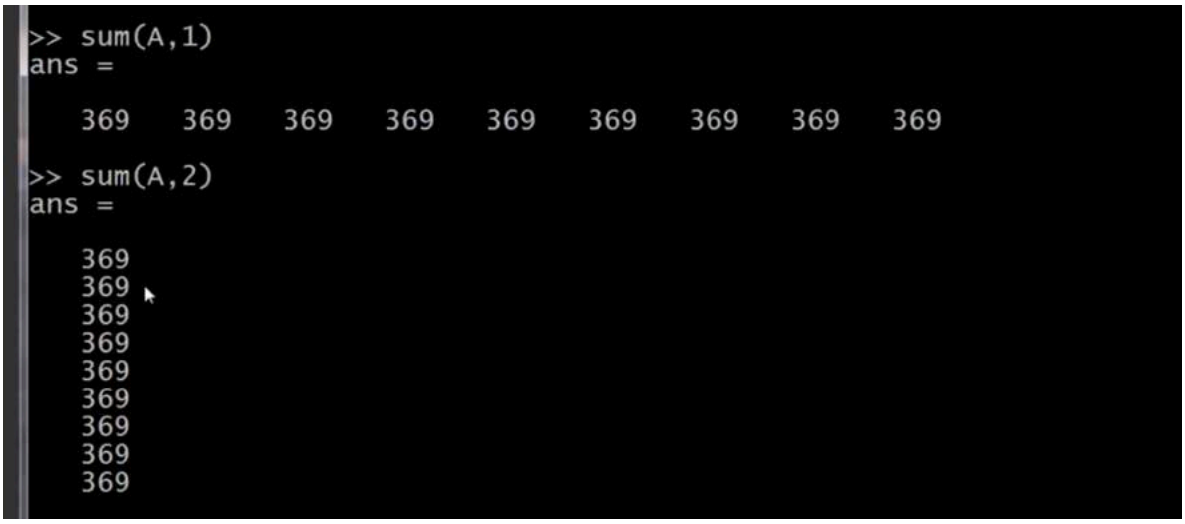
This is my magic number. This is my magic square, three by three A. Let's say I type max A, and then this will be a [], 1, what this does is this texts the column wise maximum. So the max of the first column is 8, max of second column is 9, the max of the third column is 7. This 1 means to take the max among the first dimension of 8. In contrast, if I were to type max A, this funny notation, two, then this takes the per row maximum. So the max of the first row is eight, max of second row is seven, max of the third row is nine, and so this allows you to take maxes either per row or per column. And remember the default's to a column wise element.

And remember the default's to a column wise element. So if you want to find the maximum element in the entire matrix A, you can type max(max(A)) like so, which is 9. Or you can turn A into a vector and type max(A(:)) like so and this treats this as a vector and takes the max element of that vector.

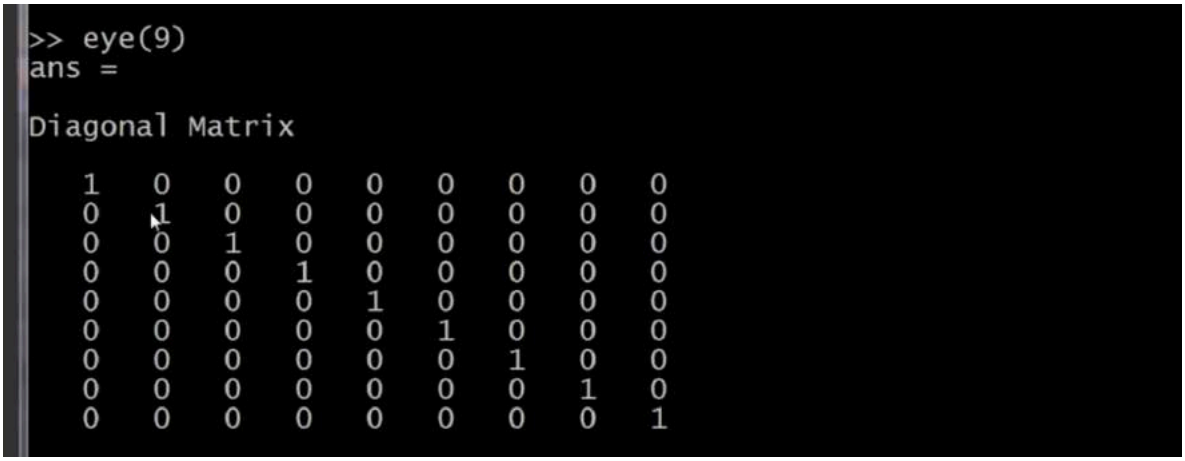


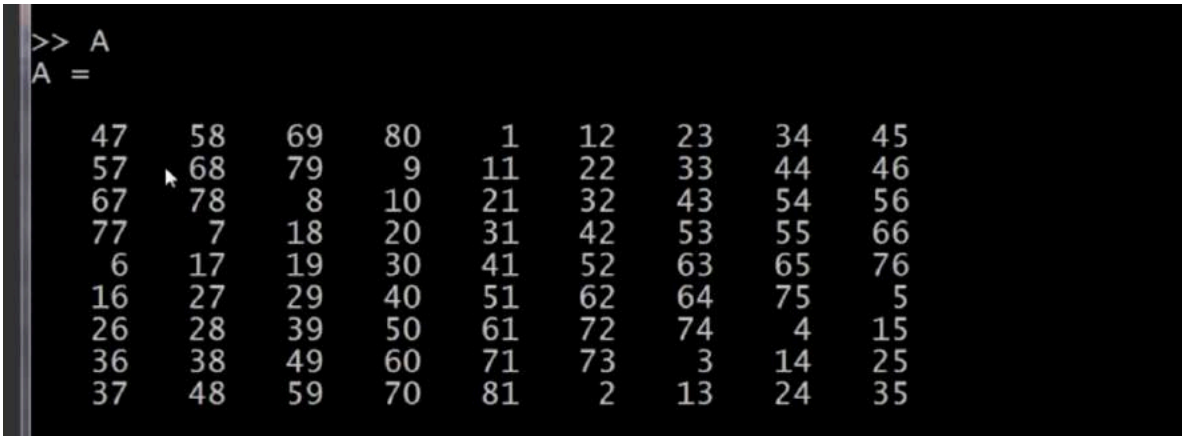
Finally let's set A to be a 9 by 9 magic square. So remember the magic square has this property that every column and every row sums the same thing, and also the diagonals, so just a nine by nine matrix square. So let me just sum(A, 1). So this does a per column sum, so we'll take each column of A and add them up and this is verified that indeed for a nine by nine matrix square, every column adds up to 369, adds up to the same thing. Now let's do the row wide sum. So the sum(A,2), and this sums up each row of A, and indeed each row of A also sums up to 369.

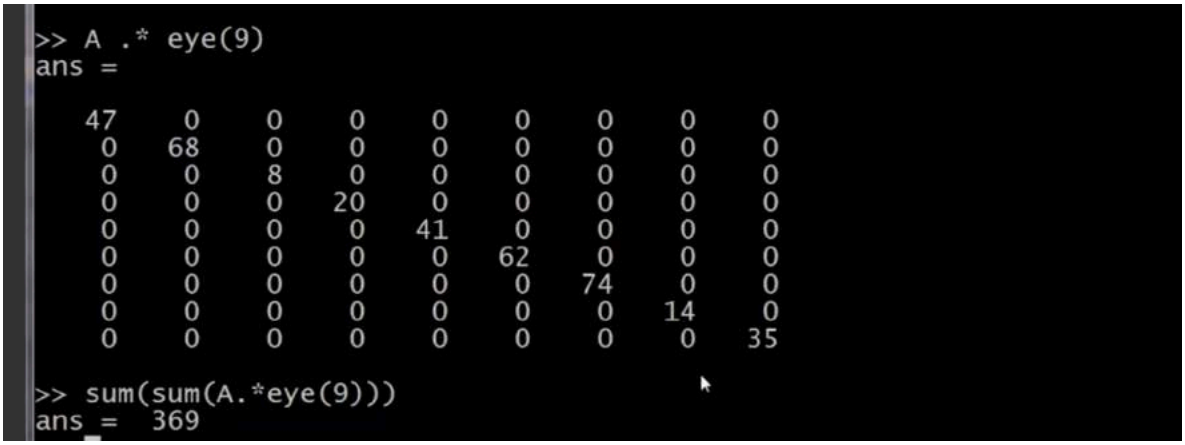




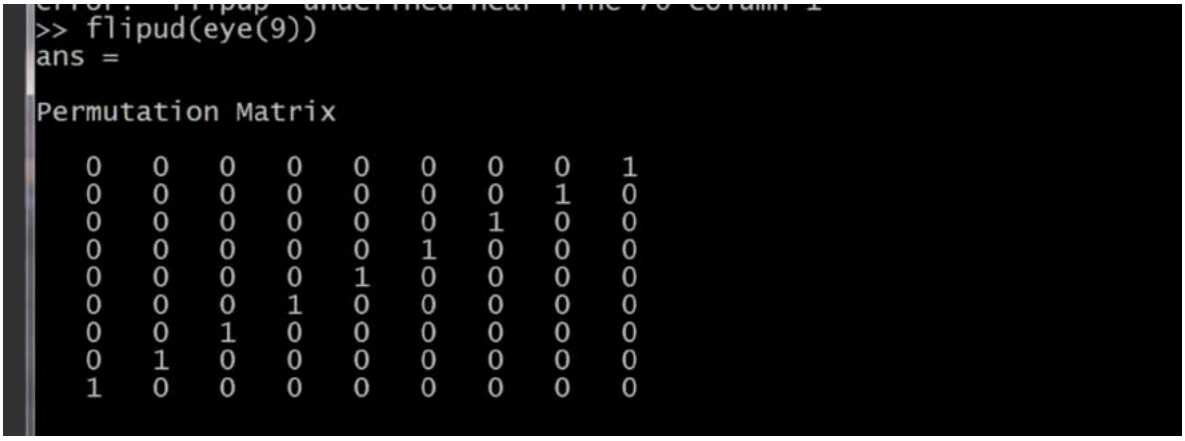
Now, let's sum the diagonal elements of A and make sure that also sums up to the same thing. So what I'm gonna do is construct a nine by nine identity matrix, that's eye nine. And let me take A and construct, multiply A element wise, so here's my matrix A. I'm going to do A .^ eye(9). What this will do is take the element wise product of these two matrices, and so this should Wipe out everything in A, except for the diagonal entries. And now, I'm gonna do sum sum of A of that and this gives me the sum of these diagonal elements, and indeed that is 369.



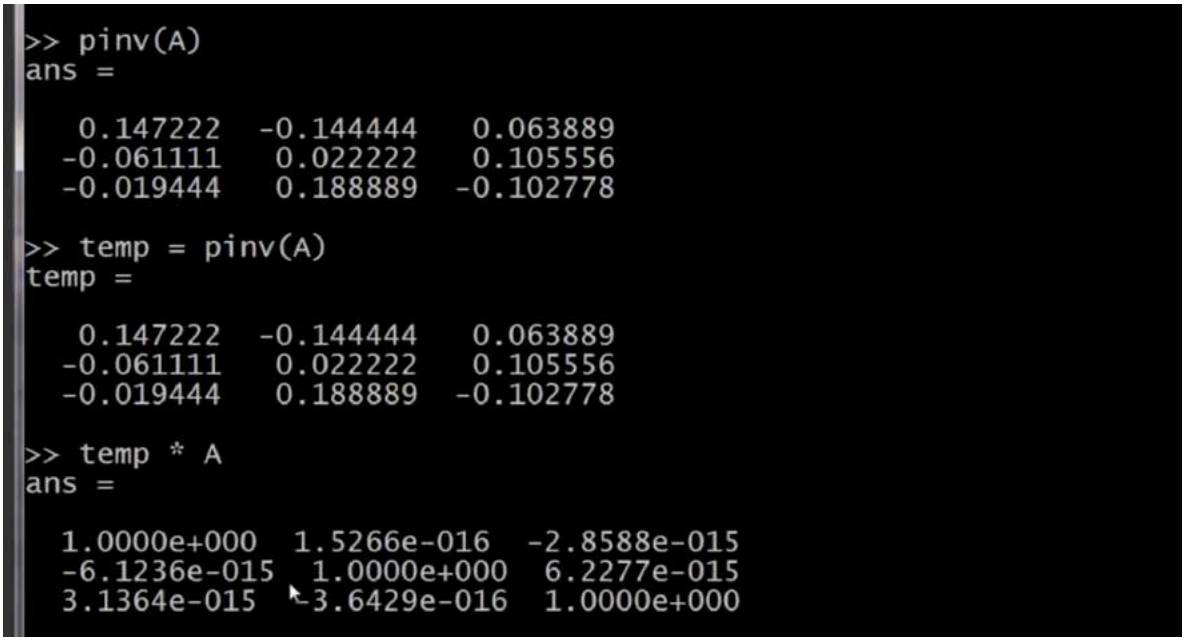




You can sum up the other diagonals as well. So this top left to bottom left, you can sum up the opposite diagonal from bottom left to top right. The commands for this is somewhat more cryptic, you don't really need to know this. I'm just showing you this in case any of you are curious. But let's see. flipud stands for flip up down. But if you do that, that turns out to sum up the elements in the opposite. So the other diagram, that also sums up to 369. Here, let me show you. Whereas eye(9) is this matrix. flipud(eye(9)), takes the identity matrix, and flips it vertically, so you end up with, excuse me, flip UD, end up with ones on this opposite diagonal as well.



Just one last command and then that's it, and then that'll be it for this video. Let's set A to be the three by three magic square game. If you want to invert a matrix, you type pinv(A). This is typically called the pseudo-inverse, but it does matter. Just think of it as basically the inverse of A, and that's the inverse of A. And so I can set temp = pinv(A) and temp times A, this is indeed the identity matrix, where it's essentially ones on the diagonals, and zeroes on the off-diagonals, up to a numeric round off.



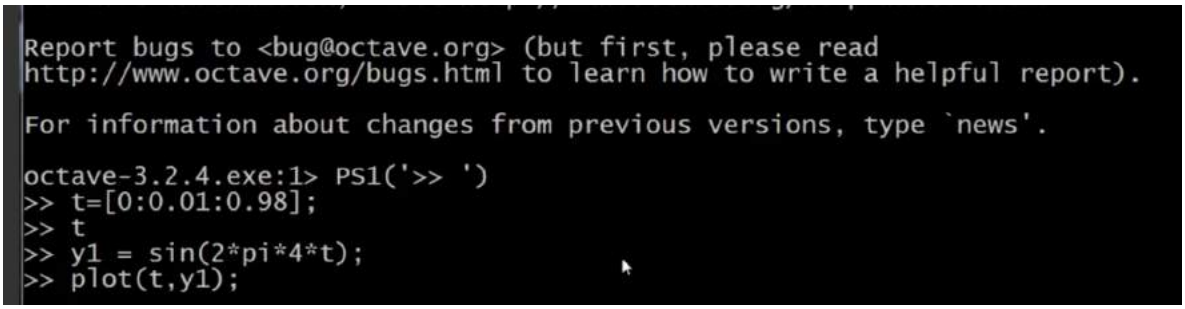
So, that's it for how to do different computational operations on data and matrices. And after running a learning algorithm, often one of the most useful things is to be able to look at your results, so to plot or visualize your result. And in the next video, I'm going to very quickly show you how again with one or two lines of code using Octave. You can quickly visualize your data or plot your data and use that to better understand what you're learning algorithms are

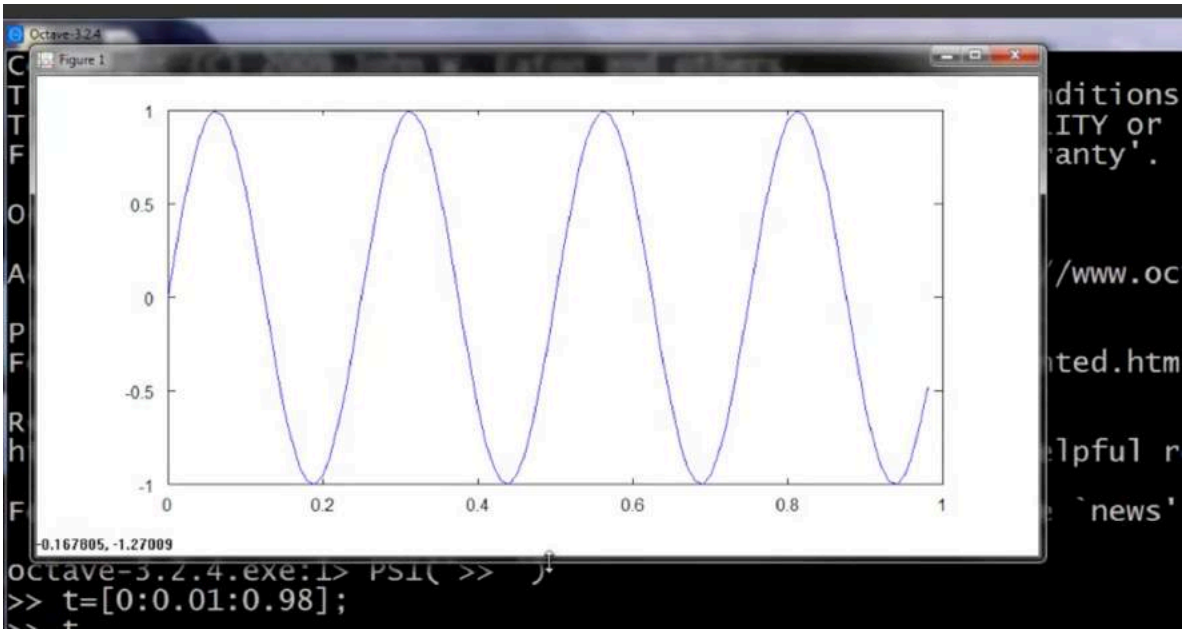
doing.

Plotting Data

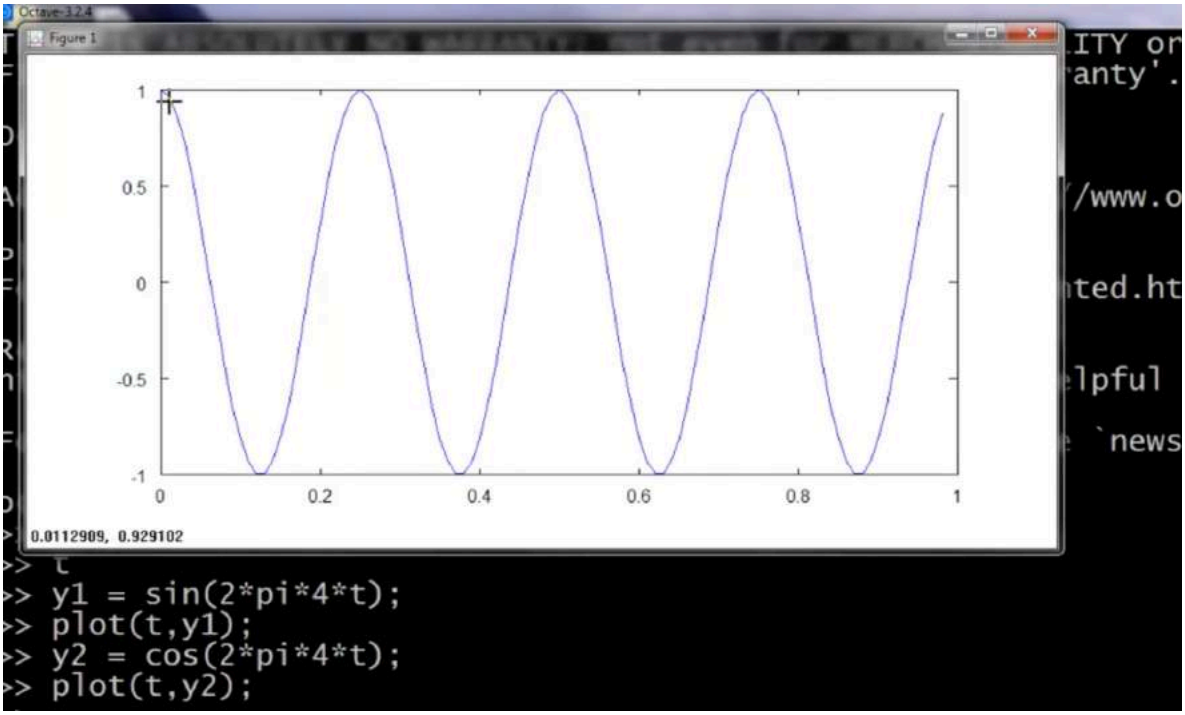
When developing learning algorithms, very often a few simple plots can give you a better sense of what the algorithm is doing and just sanity check that everything is going okay and the algorithms doing what is supposed to. For example, in an earlier video, I talked about how plotting the cost function J of theta can help you make sure that gradient descent is converging. Often, plots of the data or of all the learning algorithm outputs will also give you ideas for how to improve your learning algorithm. Fortunately, Octave has very simple tools to generate lots of different plots and when I use learning algorithms, I find that plotting the data, plotting the learning algorithm and so on are often an important part of how I get ideas for improving the algorithms and in this video, I'd like to show you some of these Octave tools for plotting and visualizing your data.

Here's my Octave window. Let's quickly generate some data for us to plot. So I'm going to set T to be equal to, you know, this array of numbers. Here's T, set of numbers going from 0 up to .98. Let's set y1 equals sine of 2 pie 40 and if I want to plot the sine function, it's very easy. I just type plot T comma Y 1 and hit enter. And up comes this plot where the horizontal axis is the T variable and the vertical axis is y1, which is the sine you saw in the function that we just computed.



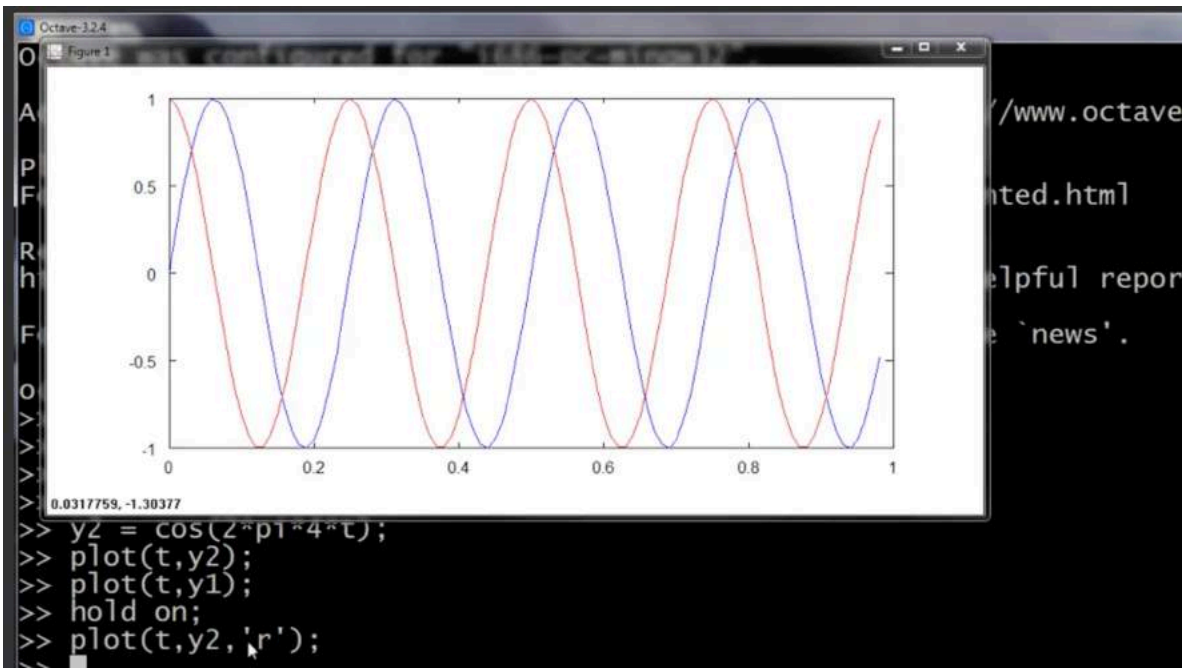


Let's set y2 to be equal to the cosine of two pi, four T, like so. And if I plot T comma y2, what octave will I do is I'll take my sine plot and it will replace with this cosine function and now, you know, cosine of xi of 1.

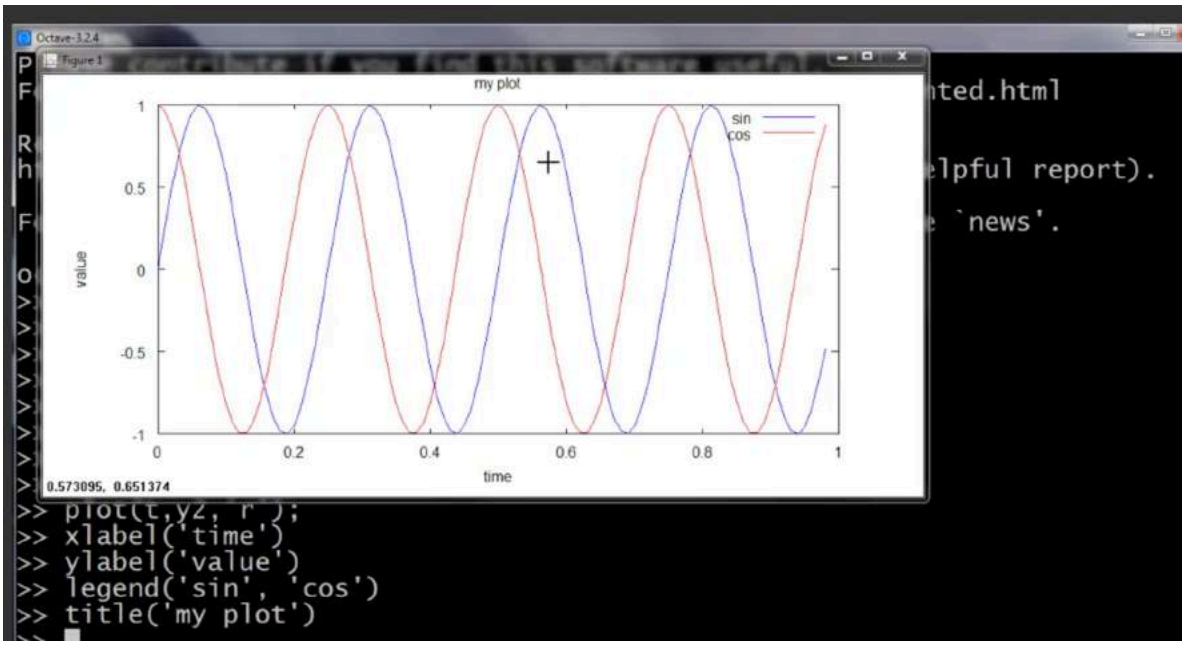


Now, what if I want to have both the sine and the cosine plots on top of each other? What I'm going to do is I'm going to type plot t,y1. So here's my sine function, and then I'm going to use the function hold on. And what hold does it

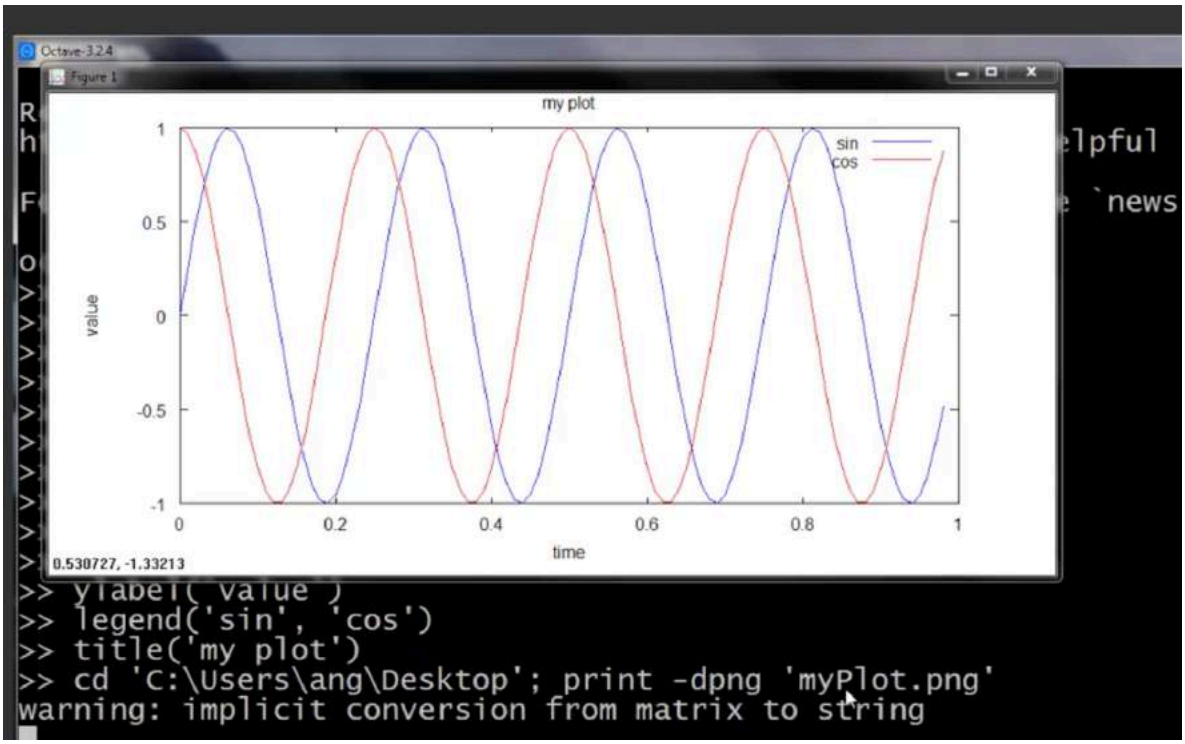
closes octaves to now figures on top of the old one and let me now plot t y2. I'm going to plot the cosine function in a different color. So, let me put there r in quotation marks there and instead of replacing the current figure, I'll plot the cosine function on top and the r indicates the what is an event color.



And here additional commands - x label times, to label the X axis, or the horizontal axis. And Y label values A, to label the vertical axis value, and I can also label my two lines with this command: legend sine cosine and this puts this legend up on the upper right showing what the 2 lines are, and finally title my plot is the title at the top of this figure.



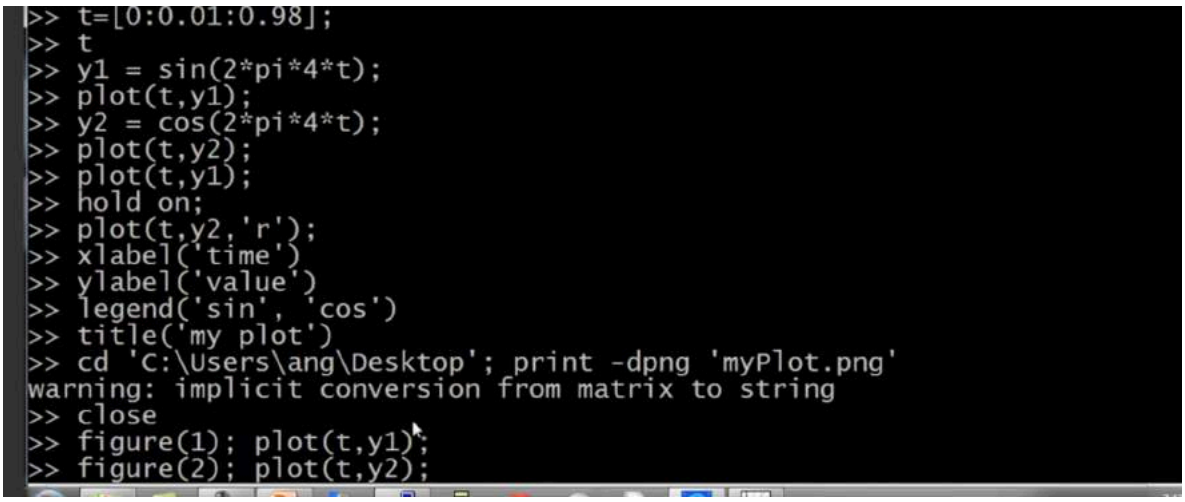
Lastly, if you want to save this figure, you type print -dpng myplot .png. So PNG is a graphics file format, and if you do this it will let you save this as a file. If I do that, let me actually change directory to, let's see, like that, and then I will print that out. So this will take a while depending on how your Octave configuration is setup, may take a few seconds, but change directory to my desktop and Octave is now taking a few seconds to save this. If I now go to my desktop, Let's hide these windows. Here's myplot.png which Octave has saved, and you know, there's the figure saved as the PNG file.

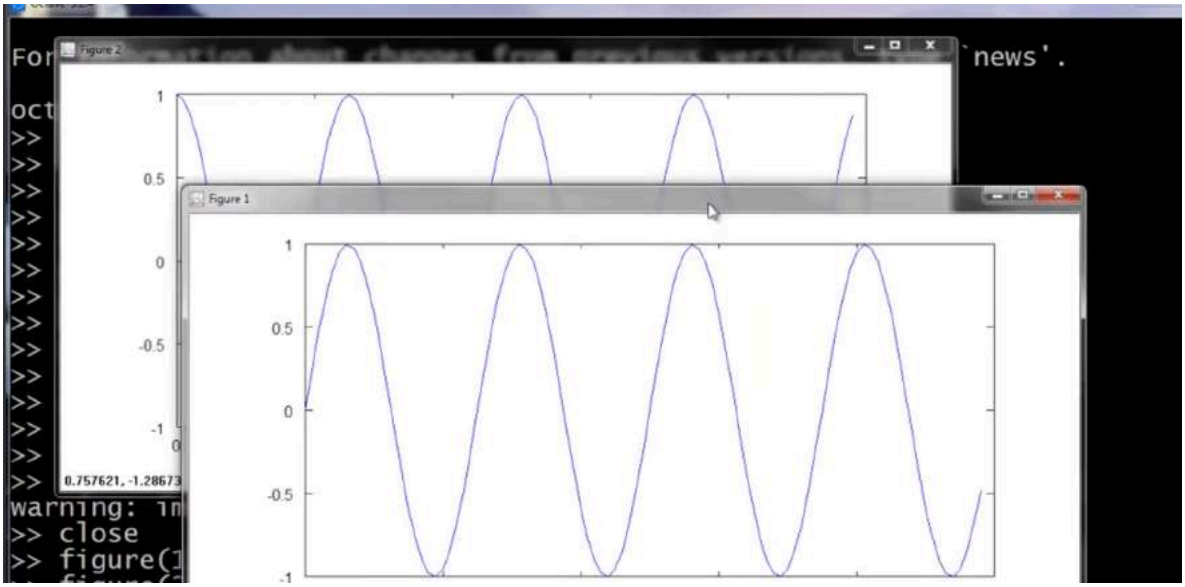


Octave can save thousand other formats as well. So, you can type help plot, if you want to see the other file formats, rather than PNG, that you can save figures in. And lastly, if you want to get rid of the plot, the close command causes the figure to go away. As I figure if I type close, that figure just disappeared from my desktop.

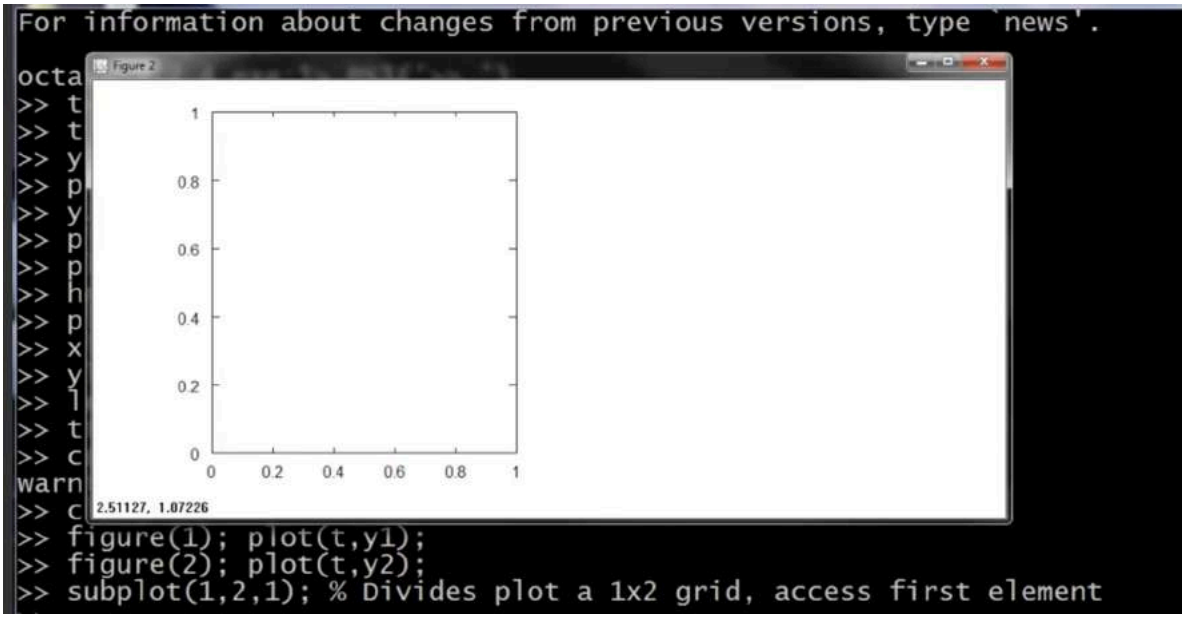
```
octave-3.2.4.exe:1> PS1('>> ')
>> t=[0:0.01:0.98];
>> t
>> y1 = sin(2*pi*4*t);
>> plot(t,y1);
>> y2 = cos(2*pi*4*t);
>> plot(t,y2);
>> plot(t,y1);
>> hold on;
>> plot(t,y2,'r');
>> xlabel('time')
>> ylabel('value')
>> legend('sin', 'cos')
>> title('my plot')
>> cd 'C:\Users\ang\Desktop'; print -dpng 'myPlot.png'
warning: implicit conversion from matrix to string
>> close
>>
```

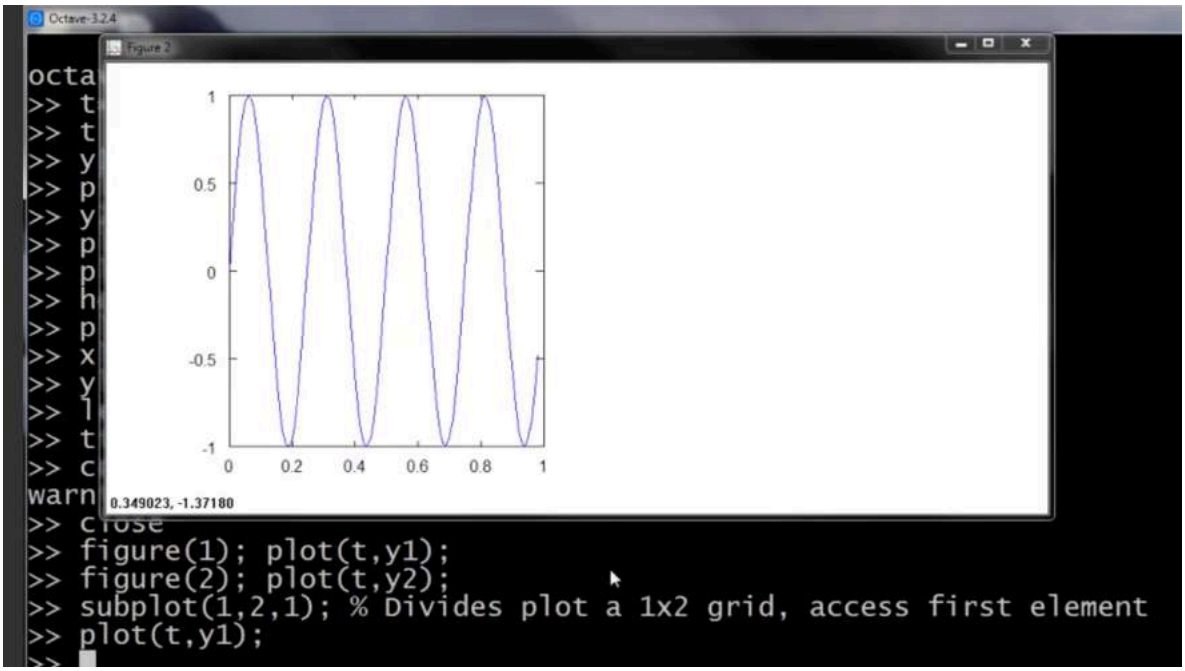
Octave also lets you specify a figure and numbers. You type figure 1 plots t, y1. That starts up first figure, and that plots t, y1. And then if you want a second figure, you specify a different figure number. So figure two, plot t, y2 like so, and now on my desktop, I actually have 2 figures. So, figure 1 and figure 2 thus 1 plotting the sine function, 1 plotting the cosine function.

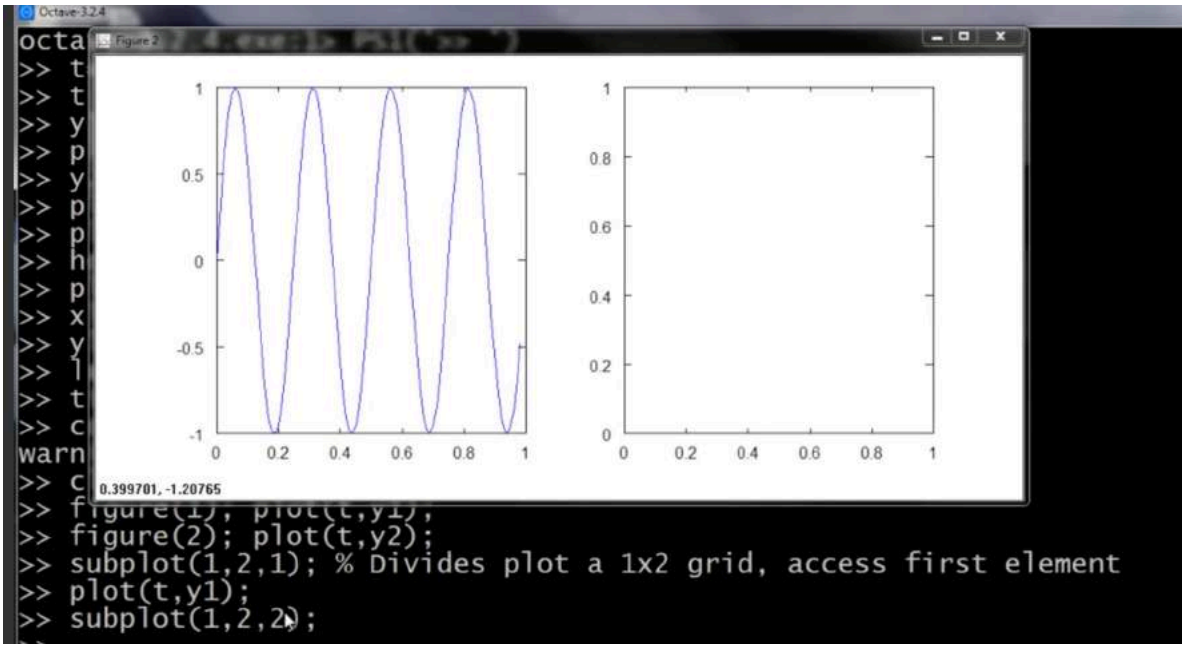


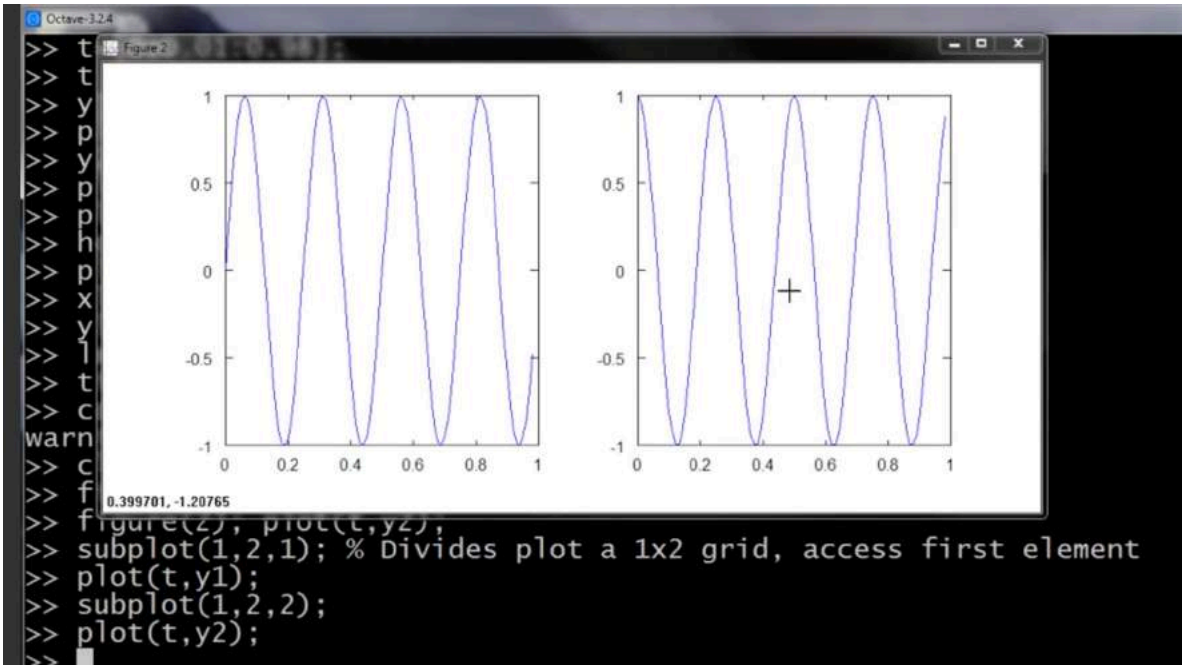


Here's one other neat command that I often use, which is the subplot command. So, we're going to use subplot 1 2 1. What it does it sub-divides the plot into a one-by-two grid with the first 2 parameters are, and it starts to access the first element. That's what the final parameter 1 is, right? So, divide my figure into a one by two grid, and I want to access the first element right now. And so, if I type that in, this product, this figure, is on the left. And if I plot t, y1, it now fills up this first element. And if I I'll do subplot 122. I'm going to start to access the second element and plot t, y2. Well, throw in y2 in the right hand side, or in the second element.

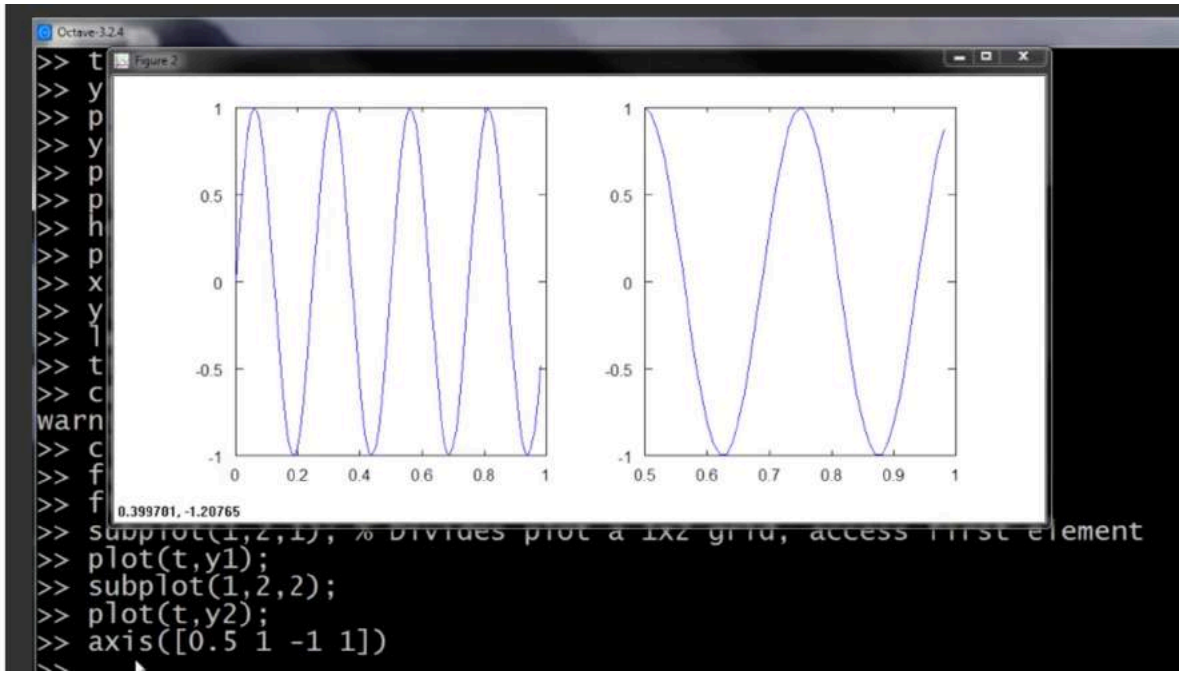



Octave-3.2.4
Figure 2
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
0.349023, -1.37180
>> figure(1); plot(t,y1);
>> figure(2); plot(t,y2);
>> subplot(1,2,1); % Divides plot a 1x2 grid, access first element
>> plot(t,y1);

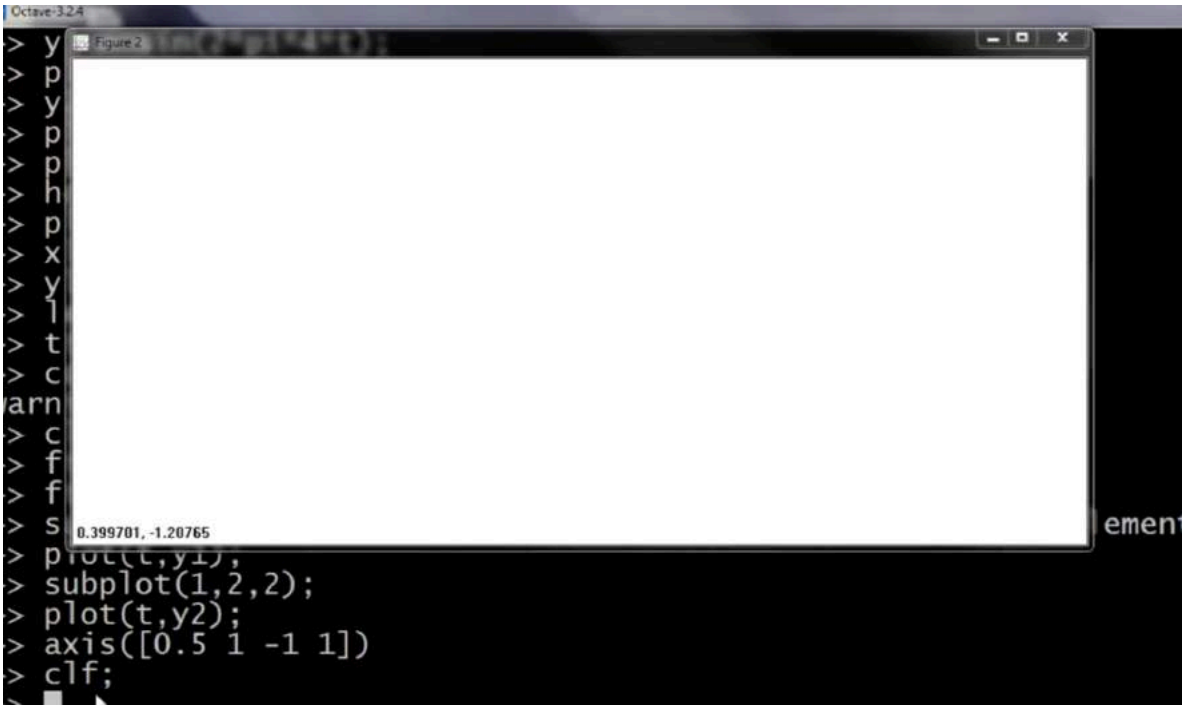

Octave-3.2.4
Figure 2
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
0.8
0.6
0.4
0.2
0.399701, -1.20765
>> figure(2); plot(t,y2);
>> subplot(1,2,1); % Divides plot a 1x2 grid, access first element
>> plot(t,y1);
>> subplot(1,2,2);



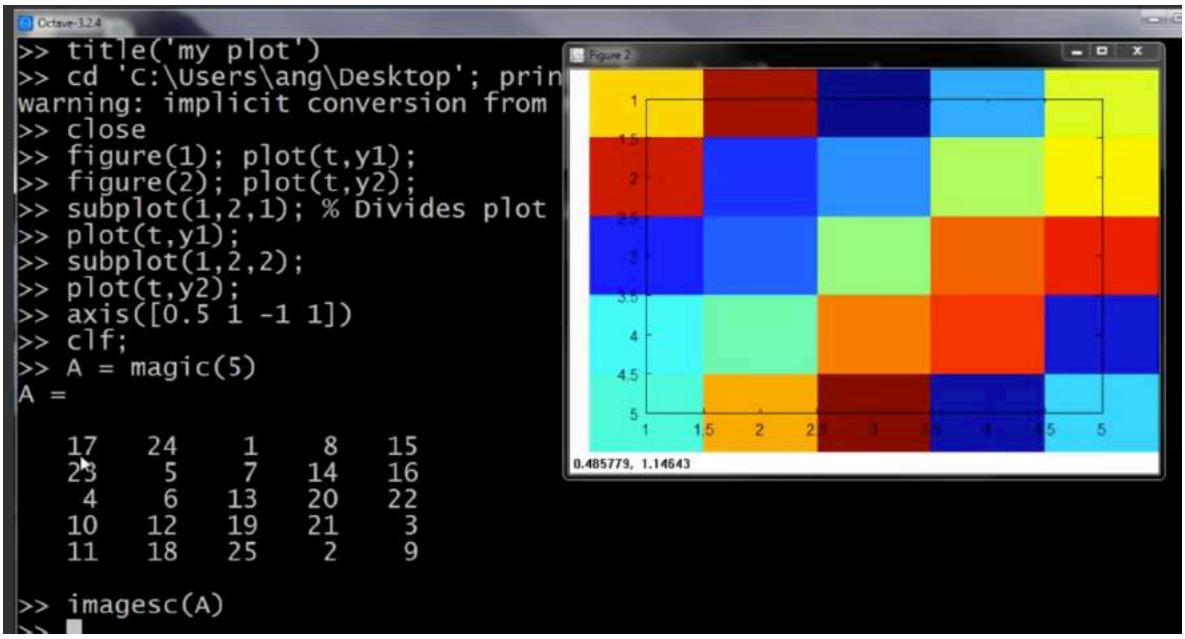
And last command, you can also change the axis scales and change axis these to 1.51 minus 1 1 and this sets the x range and y range for the figure on the right, and concretely, it assess the horizontal major values in the figure on the right to make sure 0.5 to 1, and the vertical axis values use the range from minus one to one. And, you know, you don't need to memorize all these commands. If you ever need to change the access or you need to know is that, you know, there's an access command and you can already get the details from the usual octave help command.



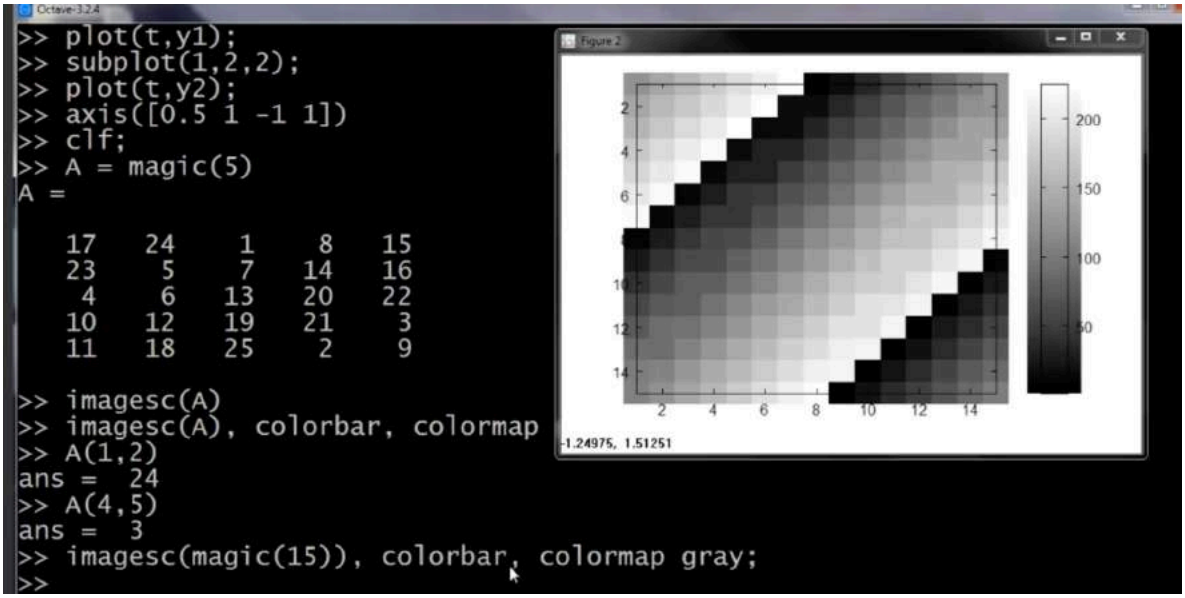
Finally, just a couple last commands CLF clears a figure



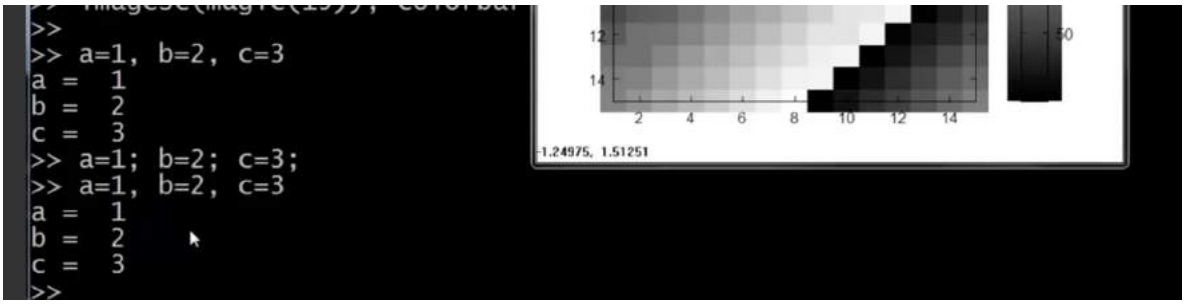
Here's one unique trait. Let's set a to be equal to a 5 by 5 magic squares a. So, a is now this 5 by 5 matrix does a neat trick that I sometimes use to visualize the matrix, which is I can use image sc of a what this will do is plot a five by five matrix, a five by five grid of color. where the different colors correspond to the different values in the A matrix. So concretely, I can also do color bar. Let me use a more sophisticated command, and image sc A color bar color map gray. This is actually running three commands at a time. I'm running image sc then running color bar, then running color map gray. And what this does, is it sets a color map, so a gray color map, and on the right it also puts in this color bar. And so this color bar shows what the different shades of color correspond to. Concretely, the upper left element of the A matrix is 17, and so that corresponds to kind of a mint shade of gray. Whereas in contrast the second element of A--sort of the 1 2 element of A--is 24. Right, so it's A 1 2 is 24. So that corresponds to this square out here, which is nearly a shade of white. And the small value, say A--what is that? A 4 5, you know, is a value 3 over here that corresponds-- you can see on my color bar that it corresponds to a much darker shade in this image.



So here's another example, I can plot a larger, you know, here's a magic 15 that gives you a 15 by 15 magic square and this gives me a plot of what my 15 by 15 magic squares values looks like.



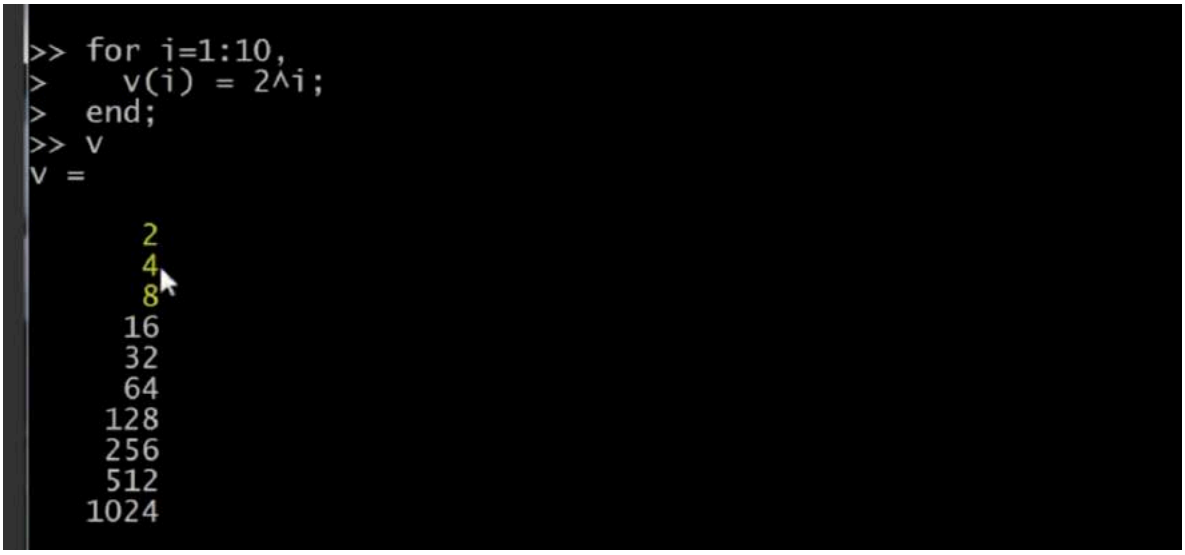
And finally to wrap up this video, what you've seen me do here is use comma chaining of function calls. Here's how you actually do this. If I type A equals 1, B equals 2, C equals 3, and hit Enter, then this is actually carrying out three commands at the same time. Or really carrying out three commands, one after another, and it prints out all three results. And this is a lot like A equals 1, B equals 2, C equals 3, except that if I use semicolons instead of a comma, it doesn't print out anything. So, this, you know, this thing here we call comma chaining of commands, or comma chaining of function calls. And, it's just another convenient way in Octave to put multiple commands like image sc color bar, colon map to put multi-commands on the same line.



So, that's it. You now know how to plot different figures and octave, and in next video the next main piece that I want to tell you about is how to write control statements like if, while, for statements and octave as well as hard to define and use functions

Control statements: for, while, if statement

In this video, I'd like to tell you how to write control statements for your Octave programs, so things like "for", "while" and "if" statements and also how to define and use functions. Here's my Octave window. Let me first show you how to use a "for" loop. I'm going to start by setting v to be a 10 by 1 vector 0. Now, here's I write a "for" loop for I equals 1 to 10. That's for I equals Y colon 10. And let's see, I'm going to set V of I equals two to the power of I, and finally end. The white space does not matter, so I am putting the spaces just to make it look nicely indented, but you know spacing doesn't matter. But if I do this, then the result is that V gets set to, you know, two to the power one, two to the power two, and so on. So this is syntax for I equals one colon 10 that makes I loop through the values one through 10.



And by the way, you can also do this by setting your indices equals one to 10, and so the indices in the array from one to 10. You can also write for I equals indices. And this is actually the same as if I equals one to 10. You can do, you know, display I and this would do the same thing.

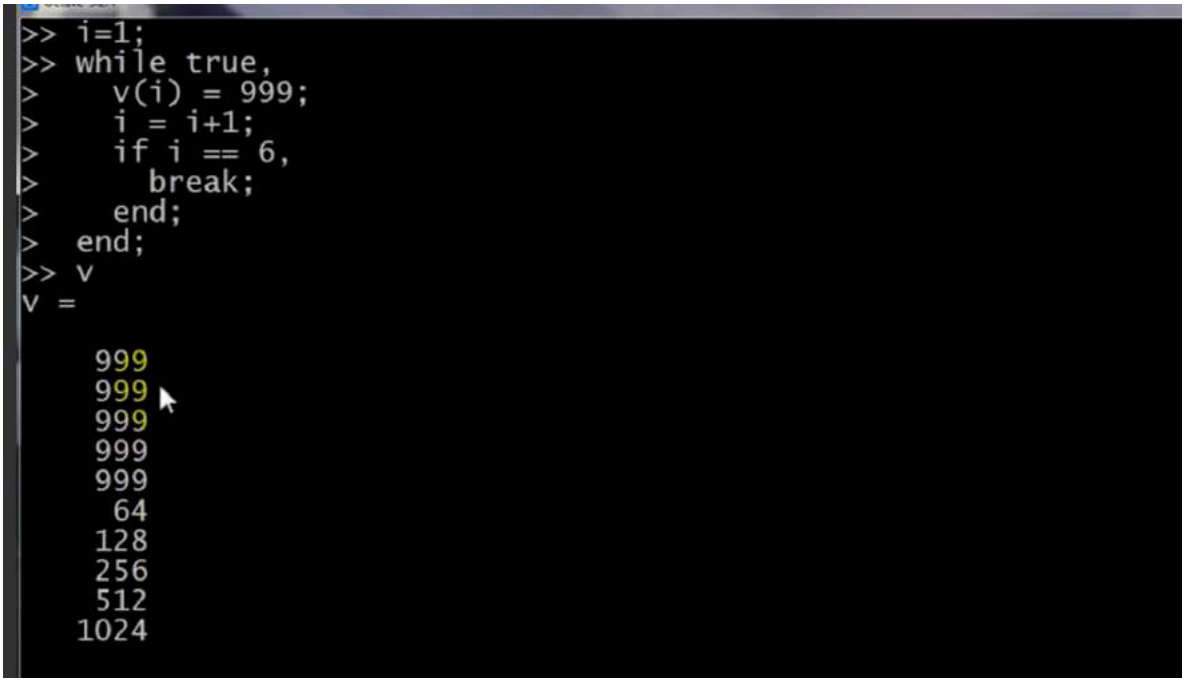
```
>> indices=1:10;
>> indices
indices =

    1    2    3    4    5    6    7    8    9   10

>> for i=indices,
>    disp(i);
>  end;
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
```

So, that is a "for" loop, if you are familiar with "break" and "continue", there's "break" and "continue" statements, you can also use those inside loops in octave, but first let me show you how a while loop works. So, here's my vector V. Let's write the while loop. I equals 1, while I is less than or equal to 5, let's set V I equals one hundred and increment I by one, end. So this says what? I starts off equal to one and then I'm going to set V I equals one hundred and increment I by one until I is, you know, greater than five. And as a result of that, whereas previously V was this powers of two vector. I've now taken the first five elements of my vector and overwritten them with this value one hundred. So that's a syntax for a while loop.

```
>> i = 1;
>> while i <= 5,
>    v(i) = 100;
>    i = i+1;
>  end;
>> v
v =

    100
    100
    100
    100
    100
     64
    128
    256
    512
   1024
```

Let's do another example. Y equals one while true and here I wanted to show

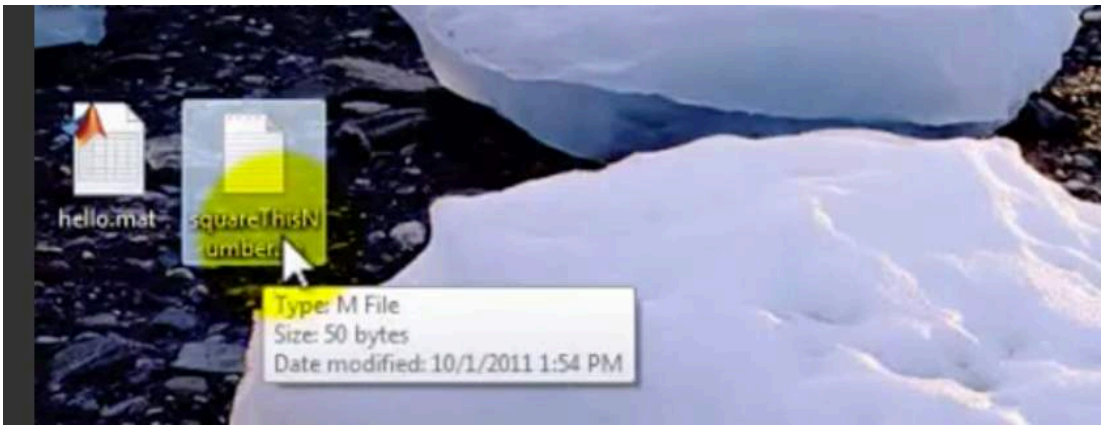
you how to use a break statement. Let's say V I equals 999 and I equals i+1 if i equals 6 break and end. And this is also our first use of an if statement, so I hope the logic of this makes sense. Since I equals one and, you know, increment loop. While repeatedly set V I equals 1 and increment i by 1, and then when 1 i gets up to 6, do a break which breaks here although the while do and so, the effective is should be to take the first five elements of this vector V and set them to 999. And yes, indeed, we're taking V and overwritten the first five elements with 999. So, this is the syntax for "if" statements, and for "while" statement, and notice the end. We have two ends here. This ends here ends the if statement and the second end here ends the while statement.

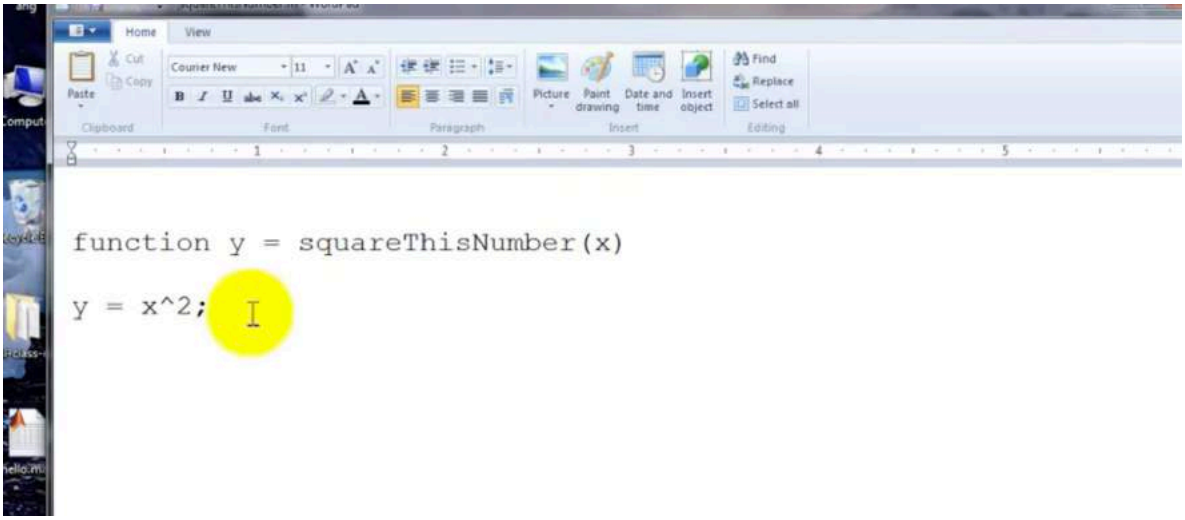


Now let me show you the more general syntax for how to use an if-else statement. So, let's see, V 1 is equal to 999, let's type V1 equals to 2 for this example. So, let me type if V 1 equals 1 display the value as one. Here's how you write an else statement, or rather here's an else if: V 1 equals 2. This is, if in case that's true in our example, display the value as 2, else display, the value is not one or two. Okay, so that's a if-else if-else statement it ends. And of course, here we've just set v 1 equals 2, so hopefully, yup, displays that the value is 2.

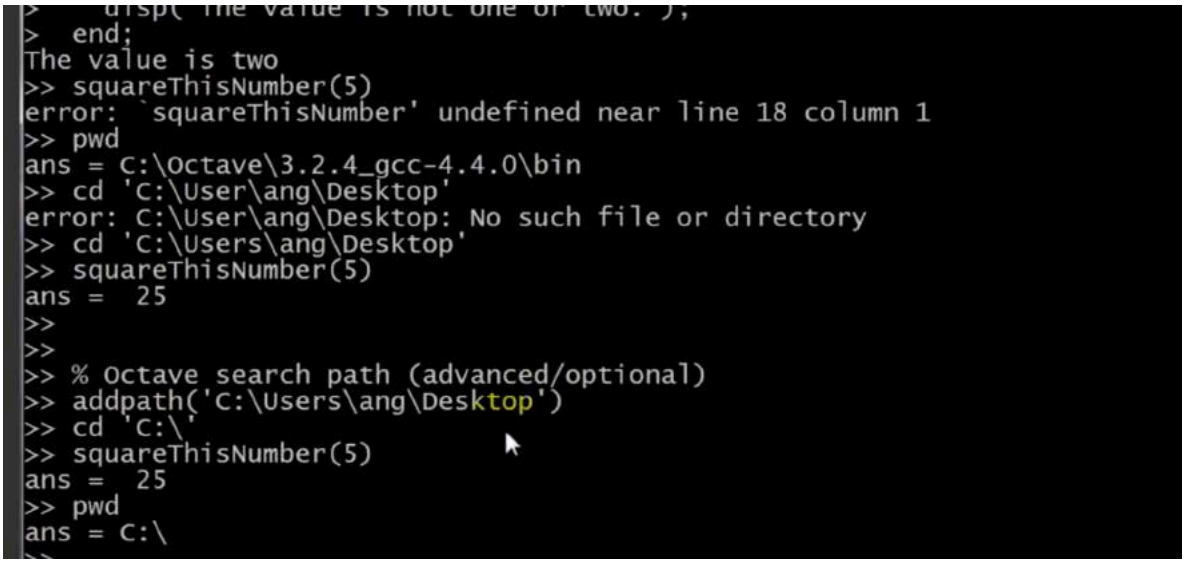
```
>> v(1)
ans =  999
>> v(1) = 2;
>> if v(1)==1,
>    disp('The value is one');
>  elseif v(1) == 2,
>    disp('The value is two');
>  else
>    disp('The value is not one or two.');
>  end;
The value is two
```

And finally, I don't think I talked about this earlier, but if you ever need to exit Octave, you can type the exit command and you hit enter that will cause Octave to quit or the 'q'--quits command also works. Finally, let's talk about functions and how to define them and how to use them. Here's my desktop, and I have predefined a file or pre-saved on my desktop a file called "squarethisnumber.m". This is how you define functions in Octave. You create a file called, you know, with your function name and then ending in .m, and when Octave finds this file, it knows that this where it should look for the definition of the function "squarethisnumber.m". Let's open up this file. Notice that I'm using the Microsoft program Wordpad to open up this file. I just want to encourage you, if your using Microsoft Windows, to use Wordpad rather than Notepad to open up these files, if you have a different text editor that's fine too, but notepad sometimes messes up the spacing. If you only have Notepad, that should work too, that could work too, but if you have Wordpad as well, I would rather use that or some other text editor, if you have a different text editor for editing your functions. So, here's how you define the function in Octave. Let me just zoom in a little bit. And this file has just three lines in it. The first line says function Y equals square root number of X, this tells Octave that I'm gonna return the value Y, I'm gonna return one value and that the value is going to be saved in the variable Y and moreover, it tells Octave that this function has one argument, which is the argument X, and the way the function body is defined, if Y equals X squared.





So, let's try to call this function "square", this number 5, and this actually isn't going to work, and Octave says square this number it's undefined. That's because Octave doesn't know where to find this file. So as usual, let's use PWD, or not in my directory, so let's see this c:\users\ang\desktop. That's where my desktop is. Oops, a little typo there. Users ANG desktop and if I now type square root number 5, it returns the answer 25. As kind of an advanced feature, this is only for those of you that know what the term search path means. But so if you want to modify the Octave search path and you could, you just think of this next part as advanced or optional material. Only for those who are either familiar with the concepts of search paths and permit languages, but you can use the term addpath, safety colon, slash users/ANG/ desktop to add that directory to the Octave search path so that even if you know, go to some other directory I can still, Octave still knows to look in the users ANG desktop directory for functions so that even though I'm in a different directory now, it still knows where to find the square this number function. Okay? But if you're not familiar with the concept of search path, don't worry about it. Just make sure as you use the CD command to go to the directory of your function before you run it and that actually works just fine.

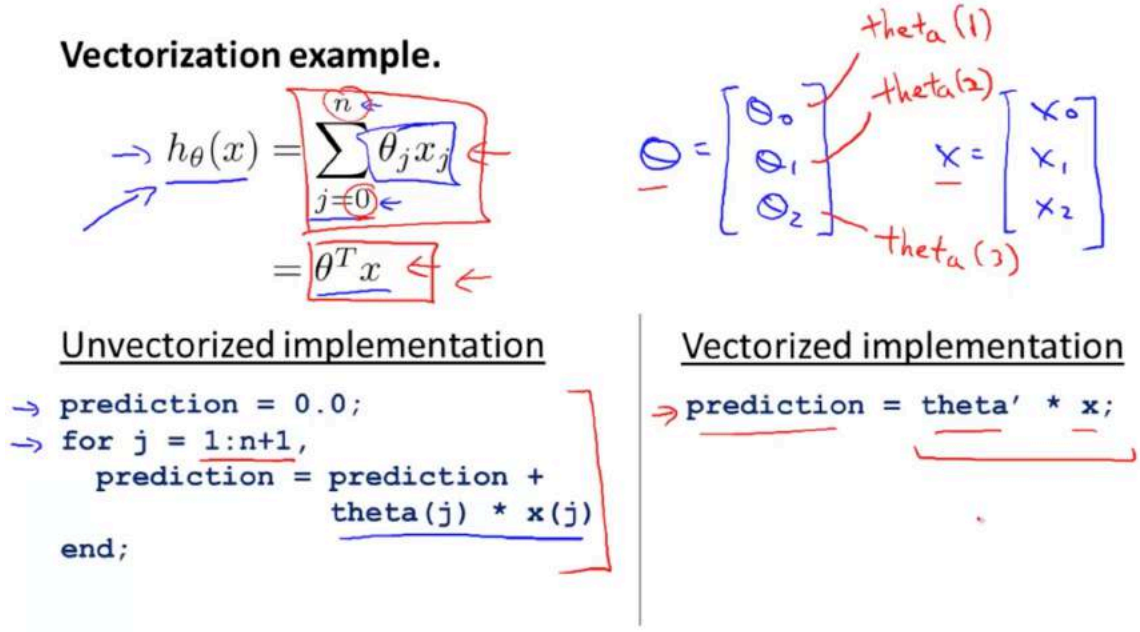


Vectorization

In this video I like to tell you about the idea of Vectorization. So, whether you using Octave or a similar language like MATLAB or whether you're using Python [INAUDIBLE], R, Java, C++, all of these languages have either built into them or have regularly and easily accessible difference in numerical linear algebra libraries. They're usually very well written, highly optimized, often sort of developed by people that have PhDs in numerical computing or they're really specialized in numerical computing. And when you're implementing machine learning algorithms, if you're able to take advantage of these linear algebra libraries or these numerical linear algebra libraries, and make some routine calls to them rather than sort of write code yourself to do things that these libraries could be doing. If you do that, then often you get code that, first, is more efficient, so you just run more quickly and take better advantage of any parallel hardware your computer may have and so on. And second, it also means that you end up with less code that you need to write, so it's a simpler implementation that is therefore maybe also more likely to be by free. And as a concrete example, rather than writing code yourself to multiply matrices, if you let Octave do it by typing a times b, that would use a very efficient routine to multiply the two matrices. And there's a bunch of examples like these, where if you use appropriate vectorization implementations you get much simpler code and much more efficient code. Let's look at some examples.

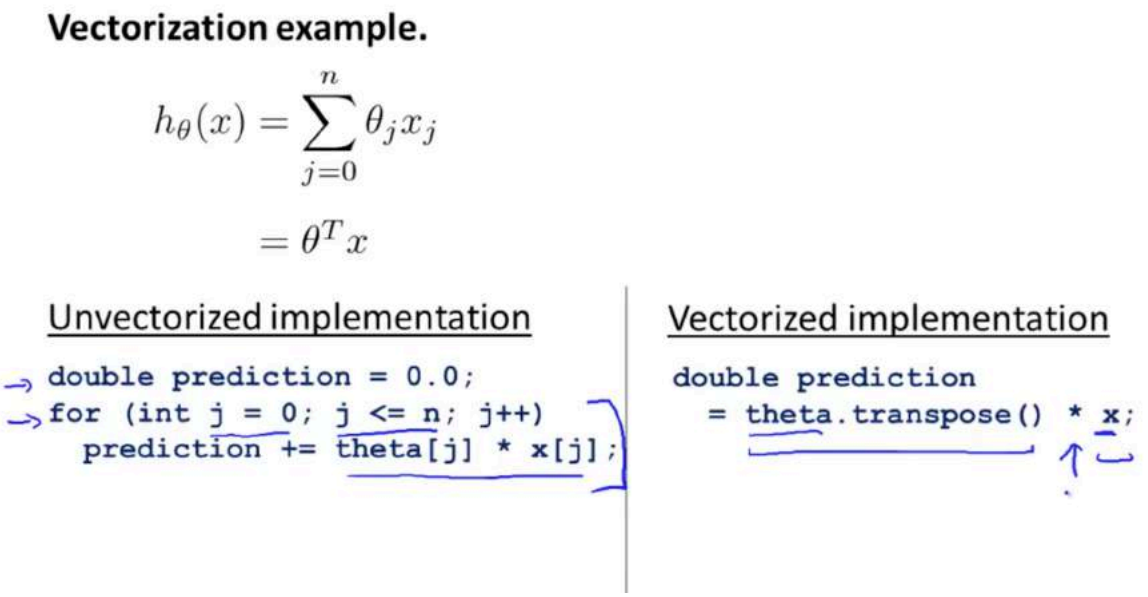
Here's our usual hypothesis for linear regression, and if you want to compute h(x), notice that there's a sum on the right. And so one thing you could do is, compute the sum from j = 0 to j = n yourself. Another way to think of this is to think of h(x) as theta transpose x, and what you can do is, think of this as you are computing this inner product between two vectors where theta is your vector, say, theta 0, theta 1, theta 2. If you have two features, if n equals two, and if you think x as this vector, x0, x1, x2, and these two views can give you two different implementations. Here's what I mean. Here's an unvectorized implementation for how to compute and by unvectorize, I mean without vectorization. We might first initialize prediction just to be 0.0. The prediction's going to eventually be h(x), and then I'm going to have a for loop for j=1

through n+1, prediction gets incremented by theta(j) * x(j). So it's kind of this expression over here. By the way, I should mention, in these vectors that I wrote over here, I had these vectors being 0 index. So I had theta 0, theta 1, theta 2. But because MATLAB is one index, theta 0 in that MATLAB, we would end up representing as theta 1 and the second element ends up as theta 2 and this third element may end up as theta 3, just because our vectors in MATLAB are indexed starting from 1, even though I wrote theta and x here, starting indexing from 0, which is why here I have a for loop. j goes from 1 through n+1 rather than j goes through 0 up to n, right? But so this is an unvectorized implementation in that we have for loop that is summing up the n elements of the sum. In contrast, here's how you would write a vectorized implementation, which is that you would think of a x and theta as vectors. You just said prediction = theta' * x. You're just computing like so. So instead of writing all these lines of code with a for loop, you instead just have one line of code. And what this line of code on the right will do is, it will use Octaves highly optimized numerical linear algebra routines to compute this inner product between the two vectors, theta and X, and not only is the vectorized implementation simpler, it will also run much more efficiently.

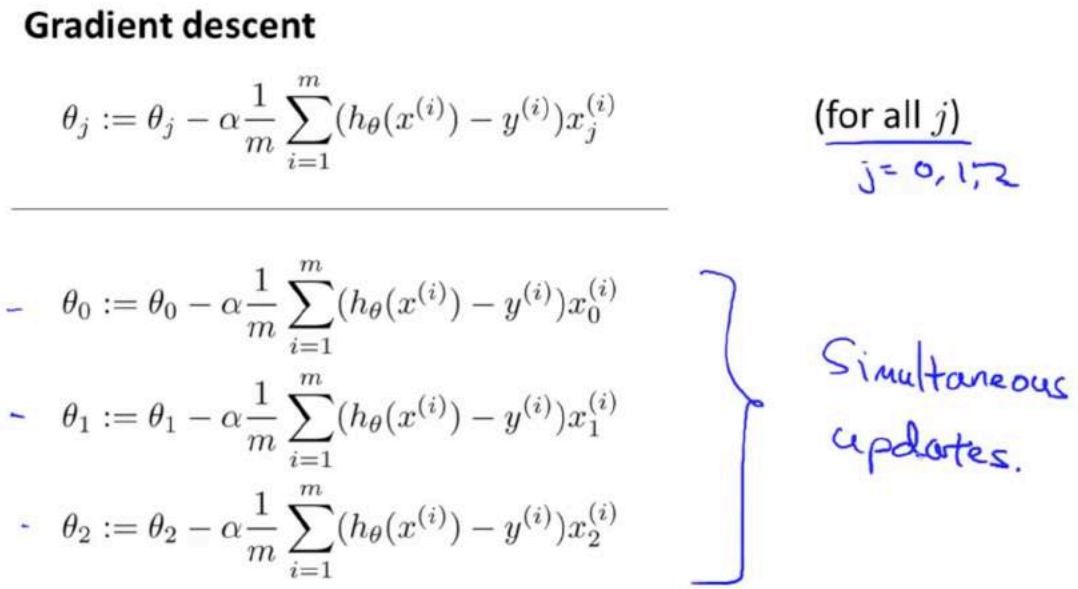


So that was octave, but the issue of vectorization applies to other programming language as well. Lets look on the example in C++. Here's what an unvectorized implementation might look like. We again initialize prediction to 0.0 and then we now how a for loop for j = 0 up to n. Prediction += theta j * x[j], where again, you have this explicit for loop that you write yourself. In contrast, using a good numerical linear algebra library in C++, you could write a function like, or rather. In contrast, using a good numerical linear algebra library in C++, you can instead write code that might look like this. So

depending on the details of your numerical linear algebra library, you might be able to have an object, this is a C++ object, which is vector theta, and a C++ object which is vector x, and you just take theta.transpose * x, where this times becomes a C++ sort of overload operator so you can just multiply these two vectors in C++. And depending on the details of your numerical linear algebra library, you might end up using a slightly different syntax, but by relying on the library to do this inner product, you can get a much simpler piece of code and a much more efficient one.

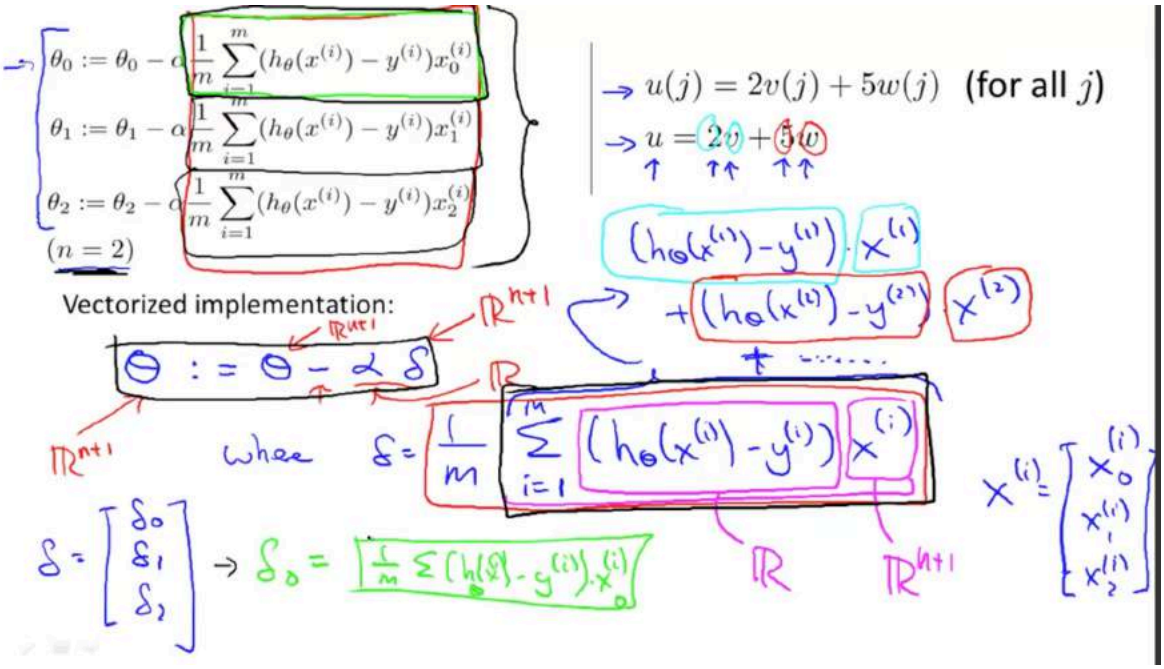


Let's now look at a more sophisticated example. Just to remind you, here's our update rule for a gradient descent of a linear regression. And so we update theta j using this rule for all values of j = 0, 1, 2, and so on. And if I just write out these equations for theta 0, theta 1, theta 2, assuming we have two features, so n = 2. Then these are the updates we perform for theta 0, theta 1, theta 2, where you might remember my saying in an earlier video, that these should be simultaneous updates.



So, let's see if we can come up with a vectorizing notation of this. Here are my same three equations written in a slightly smaller font, and you can imagine that one way to implement these three lines of code is to have a for loop that says for j = 0, 1 through 2 to update theta j, or something like that. But instead, let's come up with a vectorized implementation and see if we can have a simpler way to basically compress these three lines of code or a for loop that effectively does these three steps one set at a time. Let's see if we can take these three steps and compress them into one line of vectorized code. Here's the idea. What I'm going to do is, I'm going to think of theta as a vector, and I'm gonna update theta as theta- alpha times some other vector delta, where delta's is going to be equal to 1 over m, sum from i = 1 through m. And then this term over on the right, okay? So, let me explain what's going on here. Here, I'm going to treat theta as a vector, so this is n plus one dimensional vector, and I'm saying that theta gets here updated as that's a vector, Rn + 1. Alpha is a real number, and delta, here is a vector. So, this subtraction operation, that's a vector subtraction, okay? Cuz alpha times delta is a vector, and so I'm saying theta gets this vector, alpha times delta subtracted from it. So, what is a vector delta? Well this vector delta, looks like this, and what it's meant to be is really meant to be this thing over here. Concretely, delta will be a n plus one dimensional vector, and the very first element of the vector delta is going to be equal to that. So, if we have the delta, if we index it from 0, if it's delta 0, delta 1, delta 2, what I want is that delta 0 is equal to this first box in green up above. And indeed, you might be able to convince yourself that delta 0 is this 1 of the m sum of ho(x), x(i) minus y(i) times x(i) 0. So, let's just make sure we're on this same page about how delta really is computed. Delta is 1 over m times this sum over here, and what is this sum? Well, this term over here, that's a real

number, and the second term over here, x i, this term over there is a vector, right, because x(i) may be a vector that would be, say, x(i)0, x(i)1, x(i)2, right, and what is the summation? Well, what the summation is saying is that, this term, that is this term over here, this is equal to, (h of(x(1))- y(1)) * x(1) + (h of(x(2))- y(2) x x(2) +, and so on, okay? Because this is summation of i, so as i ranges from i = 1 through m, you get these different terms, and you're summing up these terms here. And the meaning of these terms, this is a lot like if you remember actually from the earlier quiz in this, right, you saw this equation. We said that in order to vectorize this code we will instead said u = 2v + 5w. So we're saying that the vector u is equal to two times the vector v plus five times the vector w. So this is an example of how to add different vectors and this summation's the same thing. This is saying that the summation over here is just some real number, right? That's kinda like the number two or some other number times the vector, x1. So it's kinda like 2v or say some other number times x1, and then plus instead of 5w we instead have some other real number, plus some other vector, and then you add on other vectors, plus dot, dot, dot, plus the other vectors, which is why, over all, this thing over here, that whole quantity, that delta is just some vector. And concretely, the three elements of delta correspond if n = 2, the three elements of delta correspond exactly to this thing, to the second thing, and this third thing. Which is why when you update theta according to theta- alpha delta, we end up carrying exactly the same simultaneous updates as the update rules that we have up top. So, I know that there was a lot that happened on this slide, but again, feel free to pause the video and if you aren't sure what just happened I'd encourage you to step through this slide to make sure you understand why is it that this update here with this definition of delta, right, why is it that that's equal to this update on top? And if it's still not clear, one insight is that, this thing over here, that's exactly the vector x, and so we're just taking all three of these computations, and compressing them into one step with this vector delta, which is why we can come up with a vectorized implementation of this step of the new refresh in this way. So, I hope this step makes sense and do look at the video and see if you can understand it. In case you don't understand quite the equivalence of this map, if you implement this, this turns out to be the right answer anyway. So, even if you didn't quite understand equivalence, if you just implement it this way, you'll be able to get linear regression to work. But if you're able to figure out why these two steps are equivalent, then hopefully that will give you a better understanding of vectorization as well.



Question

Suppose you have three vector valued variables u, v, w:

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}.$$

Your code implements the following:

```
for j = 1:3,
u(j) = 2 * v(j) + 5 * w(j);
end
```

How would you vectorize this code?

- ○ u = 2 * v' * v * w + 5 * w' * w * v; (where v' denotes the transpose of v)
- ● u = 2 * v + 5 * w
- ○ u = 5 * v + 2 * w
- ○ u = 2 + v + 5 + w

And finally, if you are implementing linear regression using more than one or

two features, so sometimes we use linear regression with 10's or 100's or 1,000's of features. But if you use the vectorized implementation of linear regression, you'll see that will run much faster than if you had, say, your old for loop that was updating theta zero, then theta one, then theta two yourself. So, using a vectorized implementation, you should be able to get a much more efficient implementation of linear regression. And when you vectorize later algorithms that we'll see in this class, there's good trick, whether in Octave or some other language like C++, Java, for getting your code to run more efficiently.

Quiz

1 point

1. Suppose I first execute the following Octave/Matlab commands:

```
A = [1 2; 3 4; 5 6];
B = [1 2 3; 4 5 6];
```

Which of the following are then valid commands? Check all that apply. (Hint: A' denotes the transpose of A.)

- [x] C = A * B;
- [x] C = B' + A;
- [ ] C = A' * B;
- [ ] C = B + A;

1 point

2. Let $A = \begin{bmatrix} 16 & 2 & 3 & 13 \\ 5 & 11 & 10 & 8 \\ 9 & 7 & 6 & 12 \\ 4 & 14 & 15 & 1 \end{bmatrix}$.

Which of the following indexing expressions gives $B = \begin{bmatrix} 16 & 2 \\ 5 & 11 \\ 9 & 7 \\ 4 & 14 \end{bmatrix}$? Check all that apply.

- [x] B = A(:, 1:2);
- [ ] B = A(1:4, 1:2);
- [ ] B = A(0:2, 0:4)
- [ ] B = A(1:2, 1:4);

1 point

3. Let A be a 10x10 matrix and x be a 10-element vector. Your friend wants to compute the product Ax and writes the following code:

```
1  v = zeros(10, 1);
2  for i = 1:10
3    for j = 1:10
4      v(i) = v(i) + A(i, j) * x(j);
5    end
6  end
```

How would you vectorize this code to run without any FOR loops? Check all that apply.

- [x] v = A * x;
- [ ] v = Ax;
- [ ] v = A .* x;
- [ ] v = sum (A * x);

4. Say you have two column vectors v and w, each with 7 elements (i.e., they have dimensions 7x1). Consider the following code:

```
z = 0;
for i = 1:7
  z = z + v(i) * w(i)
end
```

Which of the following vectorizations correctly compute z? Check all that apply.

- [x] z = sum (v .* w);
- [x] z = v' * w;
- [ ] z = v * w';
- [ ] z = v .* w;

1 point

5. In Octave/Matlab, many functions work on single numbers, vectors, and matrices. For example, the sin function when applied to a matrix will return a new matrix with the sin of each element. But you have to be careful, as certain functions have different behavior. Suppose you have an 7x7 matrix X. You want to compute the log of every element, the square of every element, add 1 to every element, and divide every element by 4. You will store the results in four matrices, A, B, C, D. One way to do so is the following code:

```
for i = 1:7
  for j = 1:7
    A(i, j) = log(X(i, j));
    B(i, j) = X(i, j) ^ 2;
    C(i, j) = X(i, j) + 1;
    D(i, j) = X(i, j) / 4;
  end
end
```

Which of the following correctly compute A, B, C, or D? Check all that apply.

- [x] C = X + 1;
- [x] D = X / 4;
- [x] A = log (X);
- [ ] B = X ^ 2;

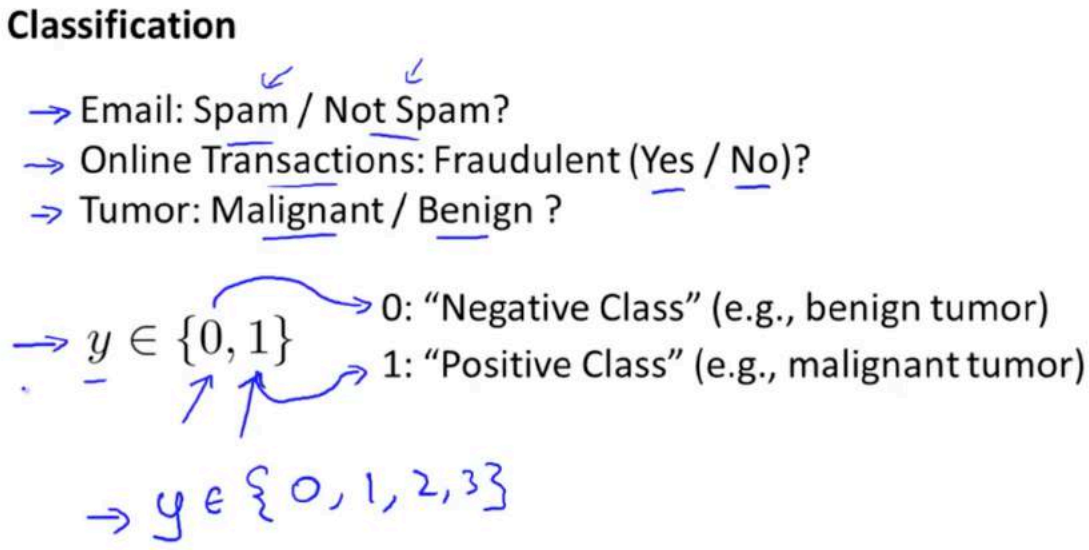
WEEK 3

Classification and Representation

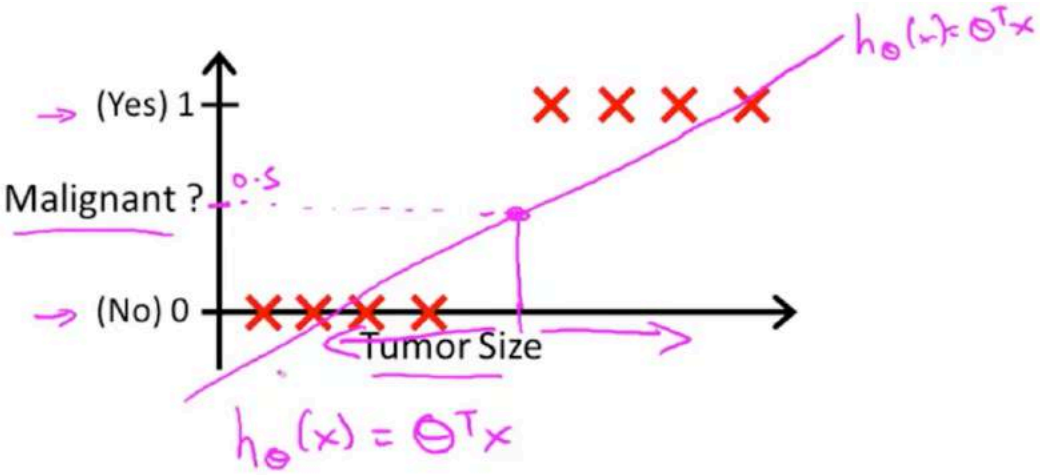
Classification (Video)

In this and the next few videos, I want to start to talk about classification problems, where the variable y that you want to predict is valued. We'll develop an algorithm called logistic regression, which is one of the most popular and most widely used learning algorithms today. Here are some examples of classification problems. Earlier we talked about email spam classification as an example of a classification problem. Another example would be classifying online transactions. So if you have a website that sells stuff and if you want to know if a particular transaction is fraudulent or not, whether someone is using a stolen credit card or has stolen the user's password. There's another classification problem. And earlier we also talked about the example of classifying tumors as cancerous, malignant or as benign tumors. In all of these problems the variable that we're trying to predict is a variable y that we can think of as taking on two values either zero or one, either spam or not spam, fraudulent or not fraudulent, related malignant or benign. Another name for the class that we denote with zero is the negative class, and another name for the class that we denote with one is the positive class. So zero we denote as the benign tumor, and one, positive class we denote a malignant tumor. The assignment of the two classes, spam not spam and so on. The assignment of the two classes to positive and negative to zero and one is somewhat arbitrary and it doesn't really matter but often there is this intuition that a negative class is conveying the absence of something like the absence of a malignant tumor. Whereas one the positive class is conveying the presence of something that we may be looking for, but the definition of which is negative and which is positive is somewhat arbitrary and it doesn't matter that much. For now we're

going to start with classification problems with just two classes zero and one. Later one we'll talk about multi class problems as well where therefore y may take on four values zero, one, two, and three. This is called a multiclass classification problem. But for the next few videos, let's start with the two class or the binary classification problem and we'll worry about the multiclass setting later.



So how do we develop a classification algorithm? Here's an example of a training set for a classification task for classifying a tumor as malignant or benign. And notice that malignancy takes on only two values, zero or no, one or yes. So one thing we could do given this training set is to apply the algorithm that we already know. Linear regression to this data set and just try to fit the straight line to the data. So if you take this training set and fill a straight line to it, maybe you get a hypothesis that looks like that, right. So that's my hypothesis. H(x) equals theta transpose x. If you want to make predictions one thing you could try doing is then threshold the classifier outputs at 0.5 that is at a vertical axis value 0.5 and if the hypothesis outputs a value that is greater than equal to 0.5 you can take y = 1. If it's less than 0.5 you can take y=0. Let's see what happens if we do that. So 0.5 and so that's where the threshold is and that's using linear regression this way. Everything to the right of this point we will end up predicting as the positive cross. Because the output values is greater than 0.5 on the vertical axis and everything to the left of that point we will end up predicting as a negative value.In this particular example, it looks like linear regression is actually doing something reasonable. Even though this is a classification toss we're interested in.



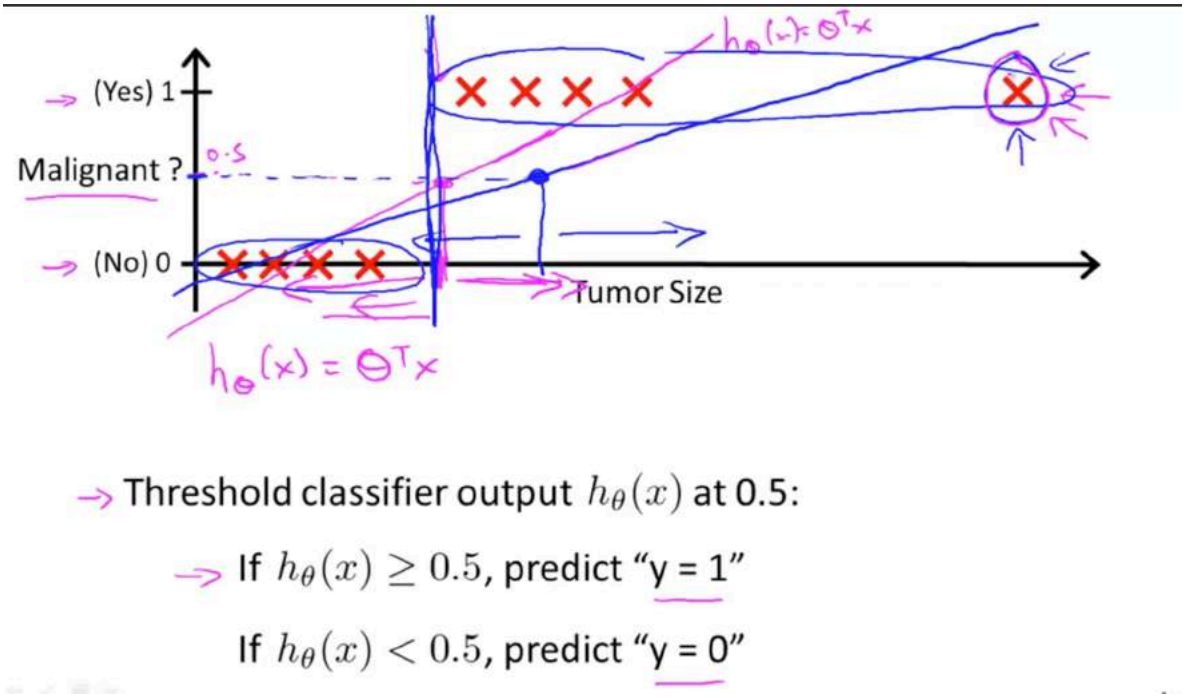
→ Threshold classifier output $h_\theta(x)$ at 0.5:

→ If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

But now let's try changing the problem a bit. Let me extend out the horizontal access a little bit and let's say we got one more training example way out there on the right. Notice that that additional training example, this one out here, it doesn't actually change anything, right. Looking at the training set it's pretty clear what a good hypothesis is. Is that well everything to the right of somewhere around here, to the right of this we should predict this positive. Everything to the left we should probably predict as negative because from this training set, it looks like all the tumors larger than a certain value around here are malignant, and all the tumors smaller than that are not malignant, at least for this training set. But once we've added that extra example over here, if you now run linear regression, you instead get a straight line fit to the data. That might maybe look like this. And if you know threshold hypothesis at 0.5, you end up with a threshold that's around here, so that everything to the right of this point you predict as positive and everything to the left of that point you predict as negative. And this seems a pretty bad thing for linear regression to have done, right, because you know these are our positive examples, these are our negative examples. It's pretty clear we really should be separating the two somewhere around there, but somehow by adding one example way out here to the right, this example really isn't giving us any new information. I mean, there should be no surprise to the learning algorithm. That the example way out here turns out to be malignant. But somehow having that example out there caused linear regression to change its straight-line fit to the data from

this magenta line out here to this blue line over here, and caused it to give us a worse hypothesis. So, applying linear regression to a classification problem often isn't a great idea. In the first example, before I added this extra training example, previously linear regression was just getting lucky and it got us a hypothesis that worked well for that particular example, but usually applying linear regression to a data set, you might get lucky but often it isn't a good idea. So I wouldn't use linear regression for classification problems.



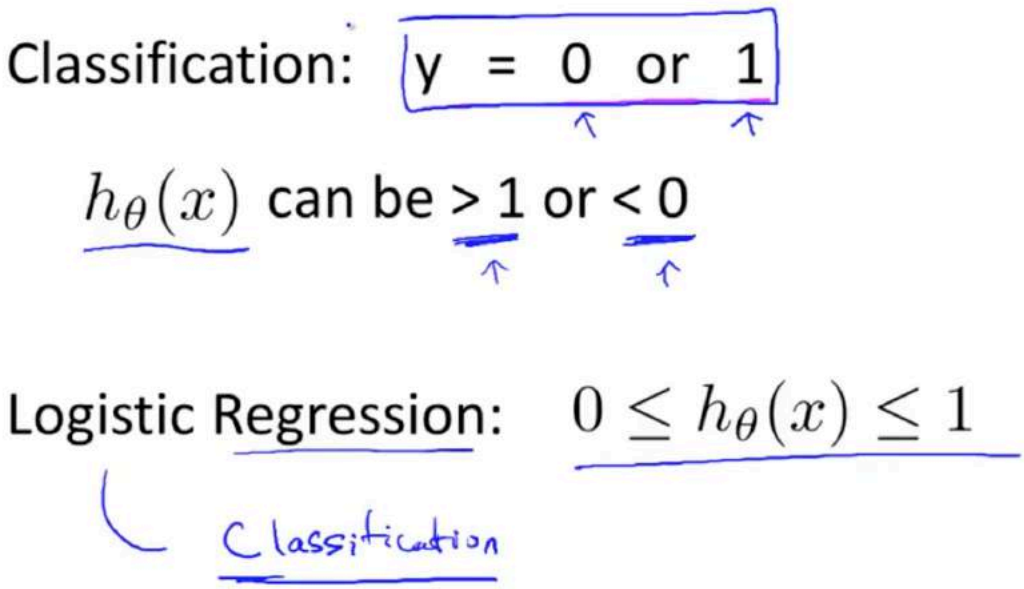
Question

Which of the following statements is true?

- If linear regression doesn't work on a classification task as in the previous example shown in the video, applying feature scaling may help.
- If the training set satisfies $0 \leq y^{(i)} \leq 1$ for every training example $(x^{(i)}, y^{(i)})$, then linear regression's prediction will also satisfy $0 \leq h_\theta(x) \leq 1$ for all values of x.
- If there is a feature x that perfectly predicts y, i.e. if $y = 1$ when $x \geq c$ and $y = 0$ whenever $x < c$ (for some constant c), then linear regression will obtain zero classification error.
- None of the above statements are true.

Here's one other funny thing about what would happen if we were to use linear

regression for a classification problem. For classification we know that y is either zero or one. But if you are using linear regression where the hypothesis can output values that are much larger than one or less than zero, even if all of your training examples have labels y equals zero or one. And it seems kind of strange that even though we know that the labels should be zero, one it seems kind of strange if the algorithm can output values much larger than one or much smaller than zero. So what we'll do in the next few videos is develop an algorithm called logistic regression, which has the property that the output, the predictions of logistic regression are always between zero and one, and doesn't become bigger than one or become less than zero. And by the way, logistic regression is, and we will use it as a classification algorithm, is some, maybe sometimes confusing that the term regression appears in this name even though logistic regression is actually a classification algorithm. But that's just a name it was given for historical reasons. So don't be confused by that logistic regression is actually a classification algorithm that we apply to settings where the label y is discrete value, when it's either zero or one. So hopefully you now know why, if you have a classification problem, using linear regression isn't a good idea. In the next video, we'll start working out the details of the logistic regression algorithm.



Classification (Transcript)

To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this

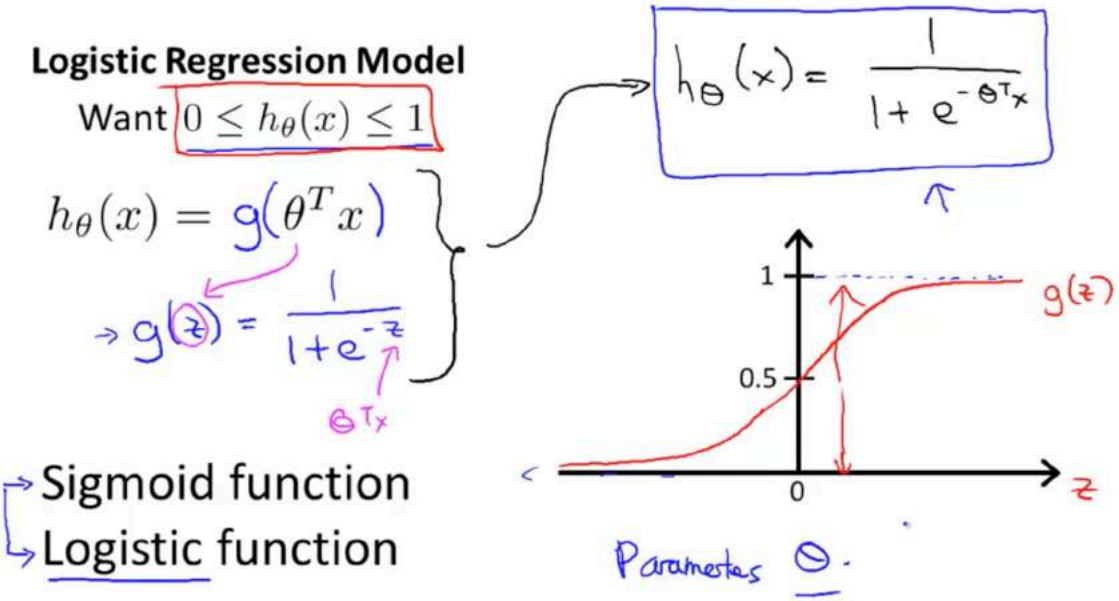
method doesn't work well because classification is not actually a linear function.

The classification problem is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the binary classification problem in which y can take on only two values, 0 and 1. (Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then x(i) may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. Hence, $y \in \{0,1\}$. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given x(i), the corresponding y(i) is also called the label for the training example.

Hypothesis Representation (Video)

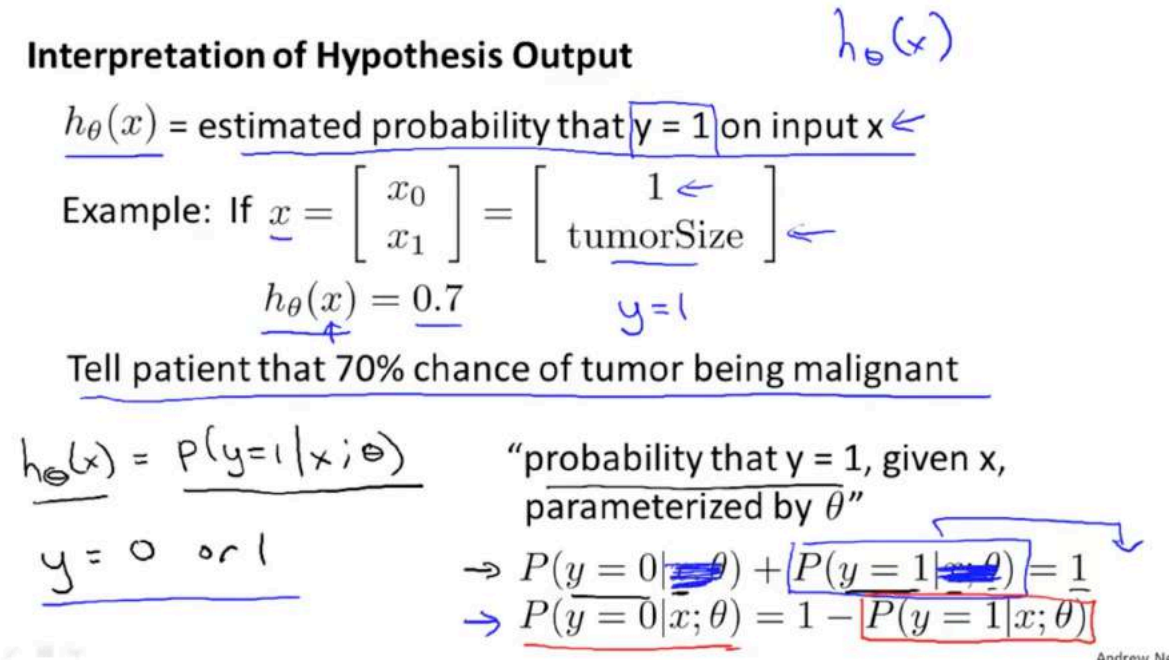
Let's start talking about logistic regression. In this video, I'd like to show you the hypothesis representation. That is, what is the function we're going to use to represent our hypothesis when we have a classification problem? Earlier, we said that we would like our classifier to output values that are between 0 and 1. So we'd like to come up with a hypothesis that satisfies this property, that is, predictions are maybe between 0 and 1. When we were using linear regression, this was the form of a hypothesis, where h(x) is theta transpose x. For logistic regression, I'm going to modify this a little bit and make the hypothesis g of theta transpose x. Where I'm going to define the function g as follows. G(z), z is a real number, is equal to one over one plus e to the negative z. This is called the sigmoid function, or the logistic function, and the term logistic function, that's what gives rise to the name logistic regression. And by the way, the terms sigmoid function and logistic function are basically synonyms and mean the same thing. So the two terms are basically interchangeable, and either term can be used to refer to this function g. And if we take these two equations and put them together, then here's just an alternative way of writing out the form of my hypothesis. I'm saying that h(x) Is 1 over 1 plus e to the negative theta transpose x. And all I've do is I've taken this variable z, z here is a real number, and plugged in theta transpose x. So I end up with theta transpose x in place of z there. Lastly, let me show you what the sigmoid function looks like. We're gonna plot it on this figure here. The sigmoid function, g(z), also called the logistic function, it looks like this. It starts off near 0 and then it rises until it

crosses 0.5 and the origin, and then it flattens out again like so. So that's what the sigmoid function looks like. And you notice that the sigmoid function, while it asymptotes at one and asymptotes at zero, as a z axis, the horizontal axis is z. As z goes to minus infinity, g(z) approaches zero. And as g(z) approaches infinity, g(z) approaches one. And so because g(z) upwards values are between zero and one, we also have that h(x) must be between zero and one. Finally, given this hypothesis representation, what we need to do, as before, is fit the parameters theta to our data. So given a training set we need to a pick a value for the parameters theta and this hypothesis will then let us make predictions.



We'll talk about a learning algorithm later for fitting the parameters theta, but first let's talk a bit about the interpretation of this model. Here's how I'm going to interpret the output of my hypothesis, h(x). When my hypothesis outputs some number, I am going to treat that number as the estimated probability that y is equal to one on a new input, example x. Here's what I mean, here's an example. Let's say we're using the tumor classification example, so we may have a feature vector x, which is this x zero equals one as always. And then one feature is the size of the tumor. Suppose I have a patient come in and they have some tumor size and I feed their feature vector x into my hypothesis. And suppose my hypothesis outputs the number 0.7. I'm going to interpret my hypothesis as follows. I'm gonna say that this hypothesis is telling me that for a patient with features x, the probability that y equals 1 is 0.7. In other words, I'm going to tell my patient that the tumor, sadly, has a 70 percent chance, or a 0.7 chance of being malignant. To write this out slightly more formally, or to write this out in math, I'm going to interpret my hypothesis output as. P of y=1 given x parameterised by theta. So for those of you that are familiar with probability, this equation may make sense. If you're a little less familiar with probability,

then here's how I read this expression. This is the probability that y is equal to one. Given x, given that my patient has features x, so given my patient has a particular tumor size represented by my features x. And this probability is parameterized by theta. So I'm basically going to count on my hypothesis to give me estimates of the probability that y is equal to 1. Now, since this is a classification task, we know that y must be either 0 or 1, right? Those are the only two values that y could possibly take on, either in the training set or for new patients that may walk into my office, or into the doctor's office in the future. So given h(x), we can therefore compute the probability that y = 0 as well, completely because y must be either 0 or 1. We know that the probability of y = 0 plus the probability of y = 1 must add up to 1. This first equation looks a little bit more complicated. It's basically saying that probability of y=0 for a particular patient with features x, and given our parameters theta. Plus the probability of y=1 for that same patient with features x and given theta parameters theta must add up to one. If this equation looks a little bit complicated, feel free to mentally imagine it without that x and theta. And this is just saying that the product of y equals zero plus the product of y equals one, must be equal to one. And we know this to be true because y has to be either zero or one, and so the chance of y equals zero, plus the chance that y is one. Those two must add up to one. And so if you just take this term and move it to the right hand side, then you end up with this equation. That says probability that y equals zero is 1 minus probability of y equals 1, and thus if our hypothesis feature of x gives us that term. You can therefore quite simply compute the probability or compute the estimated probability that y is equal to 0 as well. So, you now know what the hypothesis representation is for logistic regression and we're seeing what the mathematical formula is, defining the hypothesis for logistic regression.



Question

Suppose we want to predict, from data x about a tumor, whether it is malignant ($y = 1$) or benign ($y = 0$). Our logistic regression classifier outputs, for a specific tumor, $h_\theta(x) = P(y = 1|x; \theta) = 0.7$, so we estimate that there is a 70% chance of this tumor being malignant. What should be our estimate for $P(y = 0|x; \theta)$, the probability the tumor is benign?

- (●) $P(y = 0|x; \theta) = 0.3$
- ( ) $P(y = 0|x; \theta) = 0.7$
- ( ) $P(y = 0|x; \theta) = 0.7^2$
- ( ) $P(y = 0|x; \theta) = 0.3 \times 0.7$

In the next video, I'd like to try to give you better intuition about what the hypothesis function looks like. And I wanna tell you about something called the decision boundary. And we'll look at some visualizations together to try to get a better sense of what this hypothesis function of logistic regression really looks like.

Hypothesis Representation (Transcript)

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn’t make sense for hθ(x) to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$. To fix this, let’s change the form for our hypotheses hθ(x) to satisfy $0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging θTx into the Logistic Function.

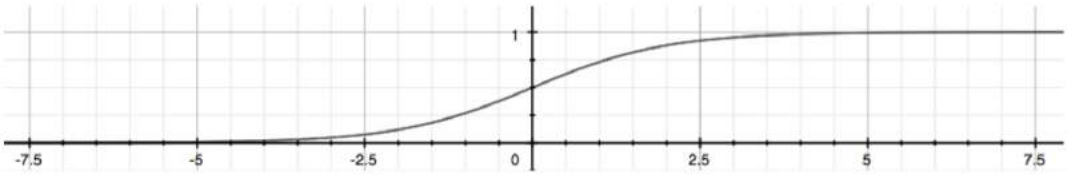
Our new form uses the "Sigmoid Function," also called the "Logistic Function":

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

The following image shows us what the sigmoid function looks like:



The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

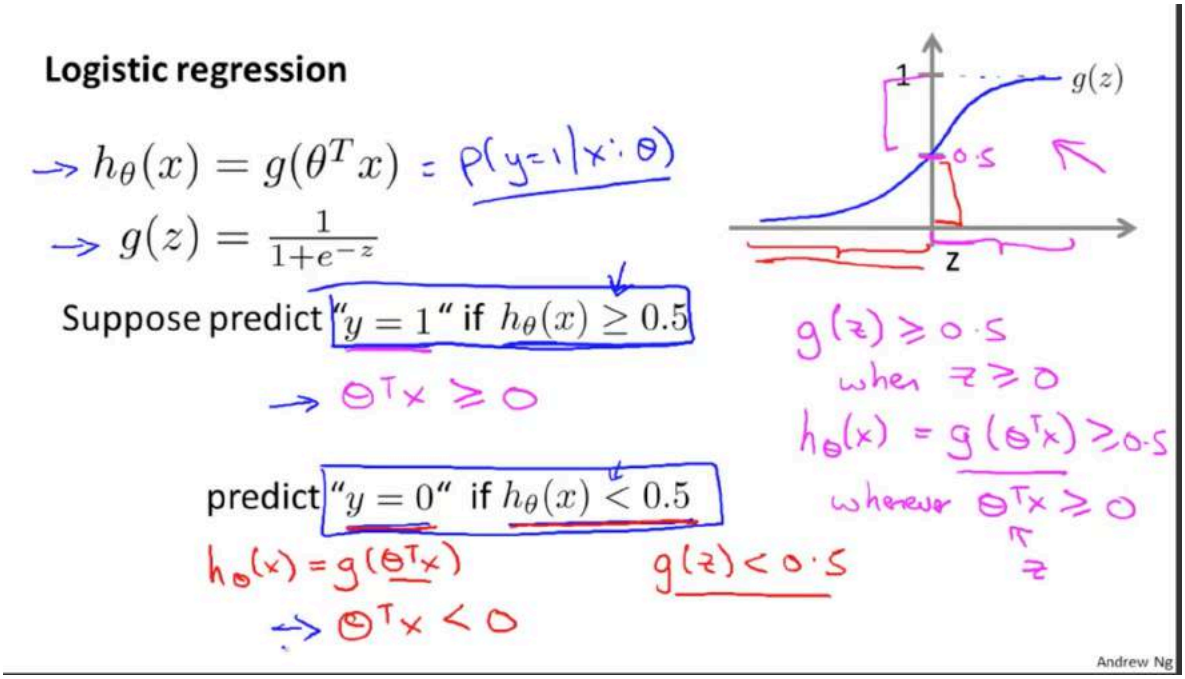
$h_\theta(x)$ will give us the **probability** that our output is 1. For example, $h_\theta(x) = 0.7$ gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

$$h_\theta(x) = P(y = 1 | x; \theta) = 1 - P(y = 0 | x; \theta)$$
$$P(y = 0 | x; \theta) + P(y = 1 | x; \theta) = 1$$

Decision boundary (Video)

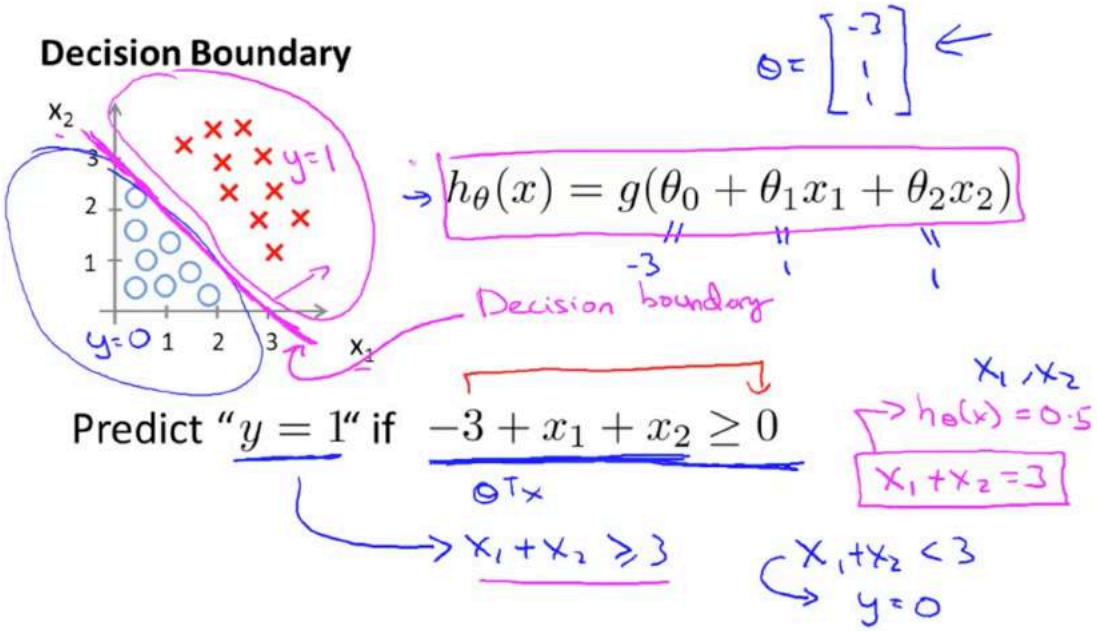
In the last video, we talked about the hypothesis representation for logistic regression. What Id like to do now is tell you about something called the decision boundary, and this will give us a better sense of what the logistic regressions hypothesis function is computing. To recap, this is what we wrote out last time, where we said that the hypothesis is represented as h of x equals g of theta transpose x, where g is this function called the sigmoid function, which looks like this. It slowly increases from zero to one, asymptoting at one.What I want to do now is try to understand better when this hypothesis will make predictions that y is equal to 1 versus when it might make predictions that y is equal to 0. And understand better what hypothesis function looks like particularly when we have more than one feature. Concretely, this hypothesis is outputting estimates of the probability that y is equal to one, given x and parameterized by theta. So if we wanted to predict is y equal to one or is y equal to zero, here's something we might do. Whenever the hypothesis outputs that the probability of y being one is greater than or equal

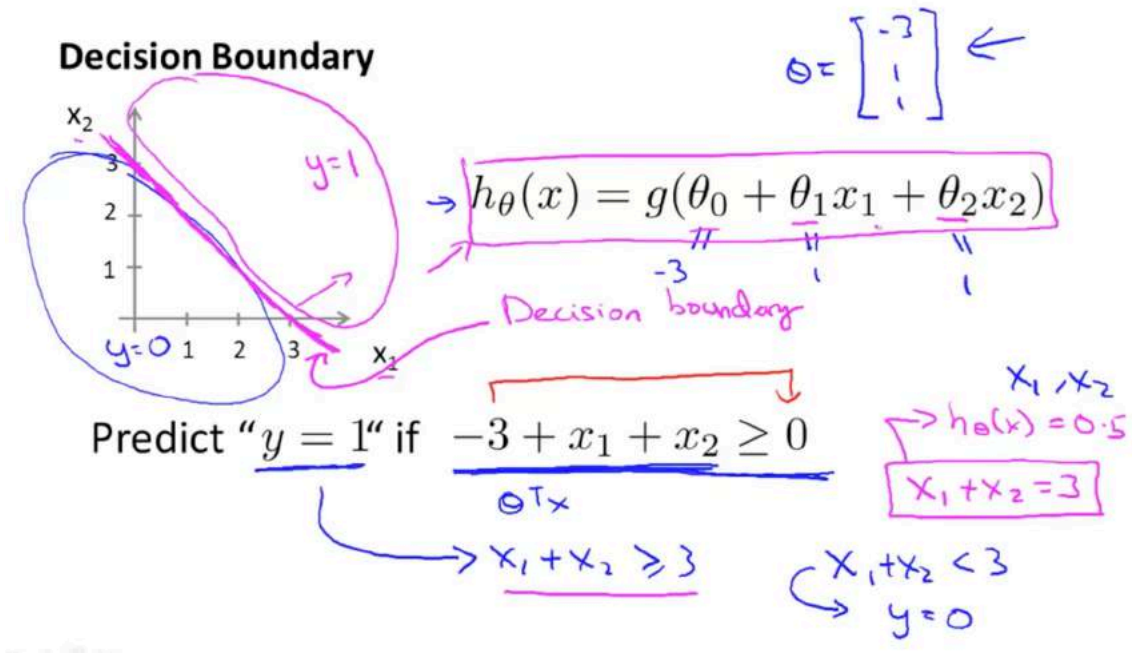
to 0.5, so this means that if there is more likely to be y equals 1 than y equals 0, then let's predict y equals 1. And otherwise, if the probability, the estimated probability of y being over 1 is less than 0.5, then let's predict y equals 0. And I chose a greater than or equal to here and less than here If h of x is equal to 0.5 exactly, then you could predict positive or negative, but I probably created a loophole here, so we default maybe to predicting positive if h of x is 0.5, but that's a detail that really doesn't matter that much. What I want to do is understand better when is it exactly that h of x will be greater than or equal to 0.5, so that we'll end up predicting y is equal to 1. If we look at this plot of the sec y function, we'll notice that the sec y function, g of z is greater than or equal to 0.5 whenever z is greater than or equal to zero. So is in this half of the figure that g takes on values that are 0.5 and higher. This notch here, that's 0.5, and so when z is positive, g of z, the sigmoid function is greater than or equal to 0.5. Since the hypothesis for logistic regression is h of x equals g of theta and transpose x, this is therefore going to be greater than or equal to 0.5, whenever theta transpose x is greater than or equal to 0. So what we're shown, right, because here theta transpose x takes the role of z. So what we're shown is that a hypothesis is gonna predict y equals 1 whenever theta transpose x is greater than or equal to 0. Let's now consider the other case of when a hypothesis will predict y is equal to 0. Well, by similar argument, h(x) is going to be less than 0.5 whenever g(z) is less than 0.5 because the range of values of z that cause g(z) to take on values less than 0.5, well, that's when z is negative. So when g(z) is less than 0.5, a hypothesis will predict that y is equal to 0. And by similar argument to what we had earlier, h(x) is equal to g of theta transpose x and so we'll predict y equals 0 whenever this quantity theta transpose x is less than 0To summarize what we just worked out, we saw that if we decide to predict whether y=1 or y=0 depending on whether the estimated probability is greater than or equal to 0.5, or whether less than 0.5, then that's the same as saying that when we predict y=1 whenever theta transpose x is greater than or equal to 0. And we'll predict y is equal to 0 whenever theta transpose x is less than 0.



Let's use this to better understand how the hypothesis of logistic regression makes those predictions. Now, let's suppose we have a training set like that shown on the slide. And suppose a hypothesis is h of x equals g of theta zero plus theta one x one plus theta two x two. We haven't talked yet about how to fit the parameters of this model. We'll talk about that in the next video. But suppose that via a procedure to specified. We end up choosing the following values for the parameters. Let's say we choose theta 0 equals 3, theta 1 equals 1, theta 2 equals 1. So this means that my parameter vector is going to be theta equals minus 3, 1, 1. So, when given this choice of my hypothesis parameters, let's try to figure out where a hypothesis would end up predicting y equals one and where it would end up predicting y equals zero. Using the formulas that we were taught on the previous slide, we know that y equals one is more likely, that is the probability that y equals one is greater than or equal to 0.5, whenever theta transpose x is greater than zero. And this formula that I just underlined, -3 + x1 + x2, is, of course, theta transpose x when theta is equal to this value of the parameters that we just chose. So for any example, for any example which features x1 and x2 that satisfy this equation, that minus 3 plus x1 plus x2 is greater than or equal to 0, our hypothesis will think that y equals 1, the small x will predict that y is equal to 1. We can also take -3 and bring this to the right and rewrite this as x1+x2 is greater than or equal to 3, so equivalently, we found that this hypothesis would predict y=1 whenever x1+x2 is greater than or equal to 3. Let's see what that means on the figure, if I write down the equation, X1 + X2 = 3, this defines the equation of a straight line and if I draw what that straight line looks like, it gives me the following line which passes through 3 and 3 on the x1 and the x2 axis. So the part of the infospace, the part of the X1 X2 plane that corresponds to when X1 plus X2 is greater

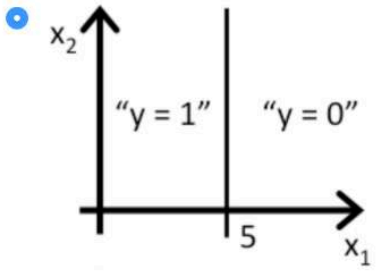
than or equal to 3, that's going to be this right half thing, that is everything to the up and everything to the upper right portion of this magenta line that I just drew. And so, the region where our hypothesis will predict y = 1, is this region, just really this huge region, this half space over to the upper right. And let me just write that down, I'm gonna call this the y = 1 region. And, in contrast, the region where x1 + x2 is less than 3, that's when we will predict that y is equal to 0. And that corresponds to this region. And there's really a half plane, but that region on the left is the region where our hypothesis will predict y = 0. I wanna give this line, this magenta line that I drew a name. This line, there, is called the decision boundary. And concretely, this straight line, X1 plus X equals 3. That corresponds to the set of points, so that corresponds to the region where H of X is equal to 0.5 exactly and the decision boundary that is this straight line, that's the line that separates the region where the hypothesis predicts Y equals 1 from the region where the hypothesis predicts that y is equal to zero. And just to be clear, the decision boundary is a property of the hypothesis including the parameters theta zero, theta one, theta two. And in the figure I drew a training set, I drew a data set, in order to help the visualization. But even if we take away the data set this decision boundary and the region where we predict y =1 versus y = 0, that's a property of the hypothesis and of the parameters of the hypothesis and not a property of the data set. Later on, of course, we'll talk about how to fit the parameters and there we'll end up using the training set, using our data. To determine the value of the parameters. But once we have particular values for the parameters theta0, theta1, theta2 then that completely defines the decision boundary and we don't actually need to plot a training set in order to plot the decision boundary.

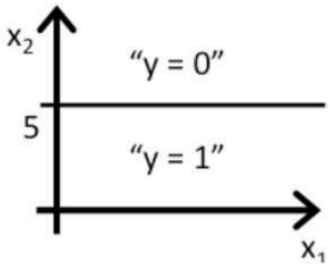



Decision Boundary
$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
-3
1
1
Decision boundary
X_2
X_1
y=1
y=0
Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$
$\theta^T x$
$X_1 + X_2 \geq 3$
X_1, X_2
$h_\theta(x) = 0.5$
$X_1 + X_2 = 3$
$X_1 + X_2 < 3$
y=0

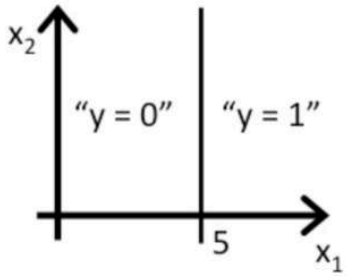
Question

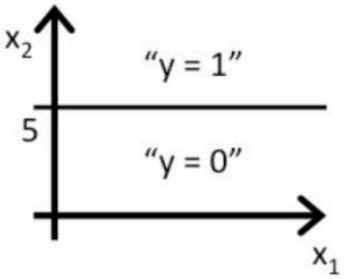
Consider logistic regression with two features x_1 and x_2. Suppose $\theta_0 = 5, \theta_1 = -1, \theta_2 = 0$, so t hat $h_\theta(x) = g(5 - x_1)$. Which of these shows the decision boundary of $h_\theta(x)$?





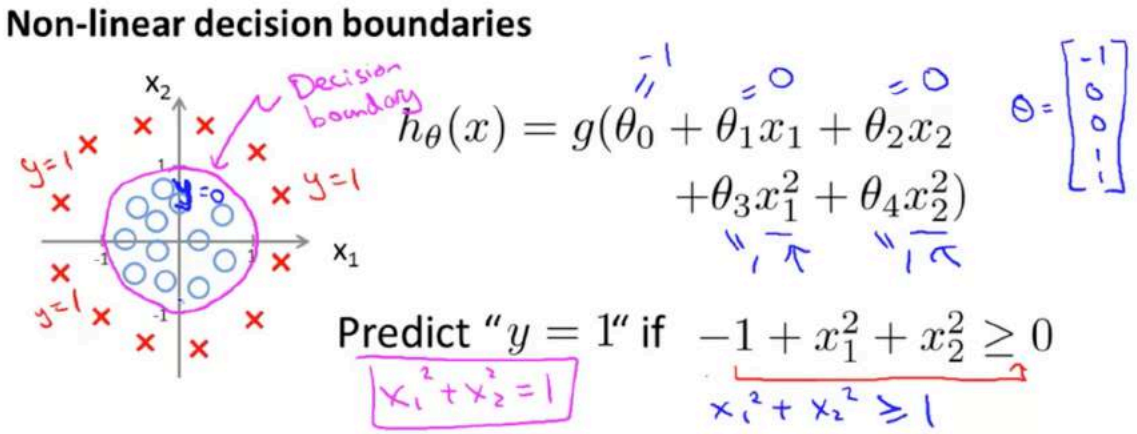
^1

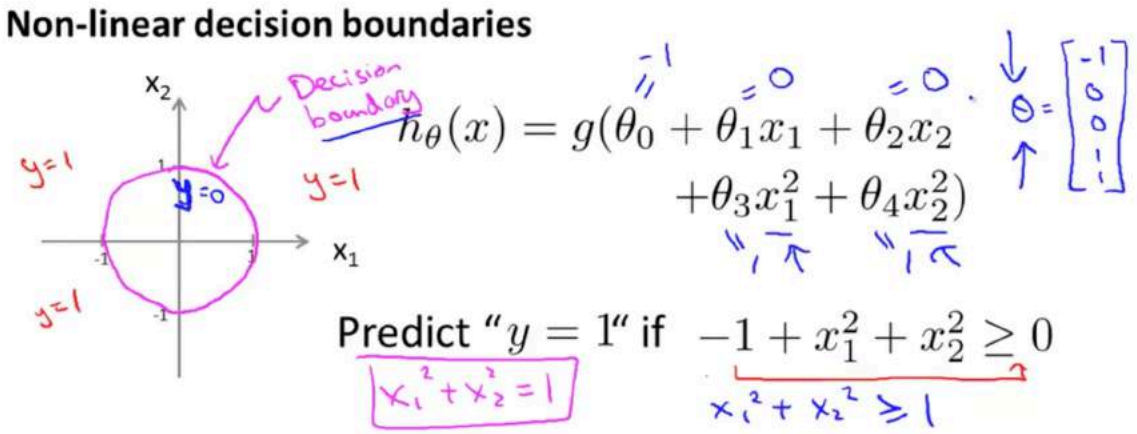




Let's now look at a more complex example where as usual, I have crosses to denote my positive examples and Os to denote my negative examples. Given a training set like this, how can I get logistic regression to fit the sort of data? Earlier when we were talking about polynomial regression or when we're talking about linear regression, we talked about how we could add extra higher order polynomial terms to the features. And we can do the same for logistic regression. Concretely, let's say my hypothesis looks like this where I've added two extra features, x1 squared and x2 squared, to my features. So that I now

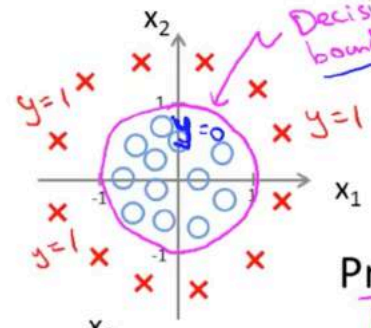
have five parameters, theta zero through theta four. As before, we'll defer to the next video, our discussion on how to automatically choose values for the parameters theta zero through theta four. But let's say that varied procedure to be specified, I end up choosing theta zero equals minus one, theta one equals zero, theta two equals zero, theta three equals one and theta four equals one. What this means is that with this particular choose of parameters, my parameter effect theta theta looks like minus one, zero, zero, one, one. Following our earlier discussion, this means that my hypothesis will predict that y=1 whenever -1 + x1 squared + x2 squared is greater than or equal to 0. This is whenever theta transpose times my theta transfers, my features is greater than or equal to zero. And if I take minus 1 and just bring this to the right, I'm saying that my hypothesis will predict that y is equal to 1 whenever x1 squared plus x2 squared is greater than or equal to 1. So what does this decision boundary look like? Well, if you were to plot the curve for x1 squared plus x2 squared equals 1 Some of you will recognize that, that is the equation for circle of radius one, centered around the origin. So that is my decision boundary. And everything outside the circle, I'm going to predict as y=1. So out here is my y equals 1 region, we'll predict y equals 1 out here and inside the circle is where I'll predict y is equal to 0. So by adding these more complex, or these polynomial terms to my features as well, I can get more complex decision boundaries that don't just try to separate the positive and negative examples in a straight line that I can get in this example, a decision boundary that's a circle. Once again, the decision boundary is a property, not of the trading set, but of the hypothesis under the parameters. So, so long as we're given my parameter vector theta, that defines the decision boundary, which is the circle. But the training set is not what we use to define the decision boundary. The training set may be used to fit the parameters theta. We'll talk about how to do that later. But, once you have the parameters theta, that is what defines the decisions boundary. Let me put back the training set just for visualization.





And finally let's look at a more complex example. So can we come up with even more complex decision boundaries then this? If I have even higher polynomial terms so things like X1 squared, X1 squared X2, X1 squared equals squared and so on. And have much higher polynomials, then it's possible to show that you can get even more complex decision boundaries and the regression can be used to find decision boundaries that may, for example, be an ellipse like that or maybe a little bit different setting of the parameters maybe you can get instead a different decision boundary which may even look like some funny shape like that. Or for even more complete examples maybe you can also get this decision boundaries that could look like more complex shapes like that where everything in here you predict y = 1 and everything outside you predict y = 0. So this higher autopolynomial features you can a very complex decision boundaries. So, with these visualizations, I hope that gives you a sense of what's the range of hypothesis functions we can represent using the representation that we have for logistic regression.

Non-linear decision boundaries



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

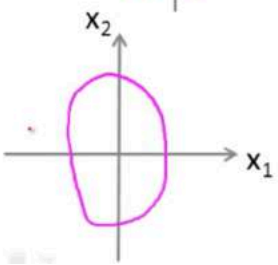
$\theta_0 = -1$, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$, $\theta_4 = 1$

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

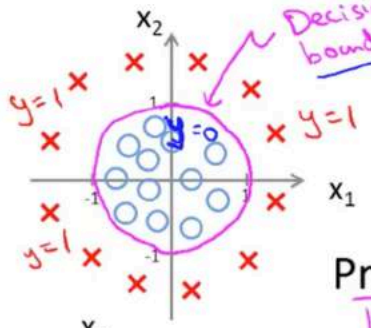
$x_1^2 + x_2^2 = 1$

$x_1^2 + x_2^2 \geq 1$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Non-linear decision boundaries



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

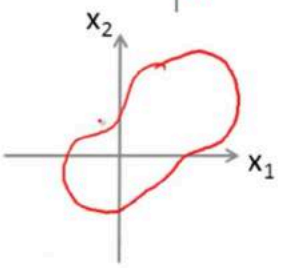
$\theta_0 = -1$, $\theta_1 = 0$, $\theta_2 = 0$, $\theta_3 = 1$, $\theta_4 = 1$

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

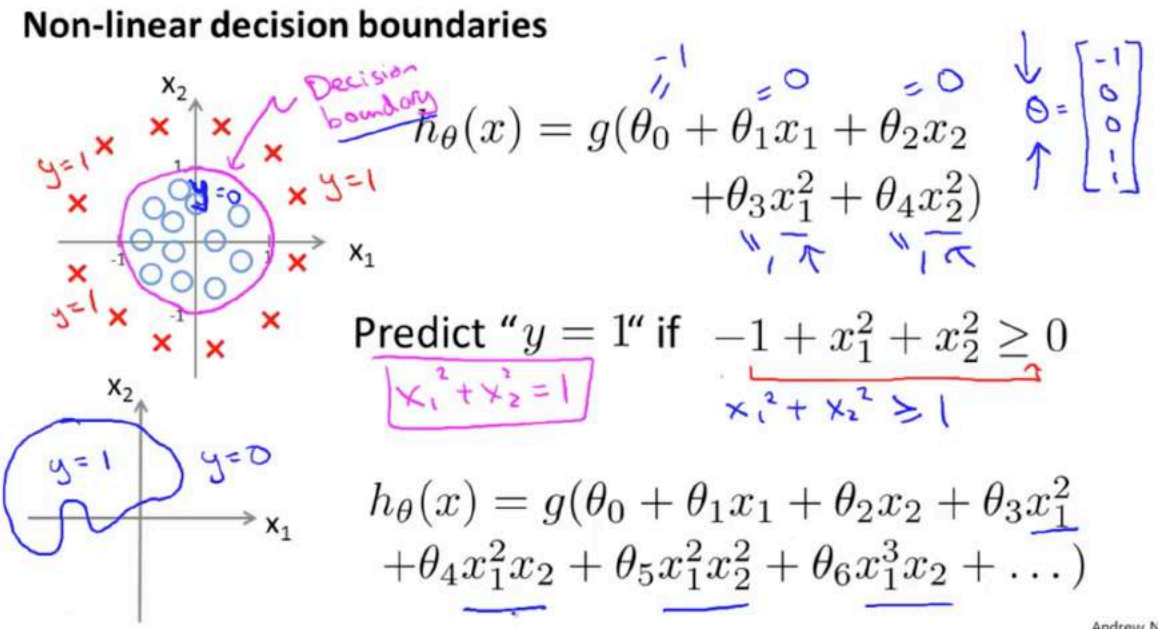
$x_1^2 + x_2^2 = 1$

$x_1^2 + x_2^2 \geq 1$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Andrew



Now that we know what h(x) can represent, what I'd like to do next in the following video is talk about how to automatically choose the parameters theta so that given a training set we can automatically fit the parameters to our data.

Decision boundary (Transcript)

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(x) < 0.5 \rightarrow y = 0$$

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

$$g(z) \geq 0.5$$
$$when\ z \geq 0$$

Remember.

$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \rightarrow \infty, e^{-\infty} \rightarrow 0 \Rightarrow g(z) = 1$$
$$z \rightarrow -\infty, e^{\infty} \rightarrow \infty \Rightarrow g(z) = 0$$

So if our input to g is $\theta^T X$, then that means:

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$

From these statements we can now say:

$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

Example:

$$\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$$
$$y = 1\ if\ 5 + (-1)x_1 + 0x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

In this case, our decision boundary is a straight vertical line placed on the graph where $x_1 = 5$, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.

Again, the input to the sigmoid function g(z) (e.g. $\theta^T X$) doesn't need to be linear, and could be a function that describes a circle (e.g. $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.