MACHINE LEARNING-2

Matrix Vector Multiplication

In this video, I'd like to start talking about how to multiply together two matrices. We'll start with a special case of that, of matrix vector multiplication - multiplying a matrix together with a vector. Let's start with an example. Here is a matrix, and here is a vector, and let's say we want to multiply together this matrix with this vector, what's the result? Let me just work through this example and then we can step back and look at just what the steps were. It turns out the result of this multiplication process is going to be, itself, a vector. And I'm just going work with this first and later we'll come back and see just what I did here. To get the first element of this vector I am going to take these two numbers and multiply them with the first row of the matrix and add up the corresponding numbers. Take one multiplied by one, and take three and multiply it by five, and that's what, that's one plus fifteen so that gives me sixteen. I'm going to write sixteen here. then for the second row, second element, I am going to take the second row and multiply it by this vector, so I have four times one, plus zero times five, which is equal to four, so you'll have four there. And finally for the last one I have two one times one five, so two by one, plus one by 5, which is equal to a 7, and so I get a 7 over there. It turns out that the results of multiplying that's a 3x2 matrix by a 2x1 matrix is also just a two-dimensional vector. The result of this is going to be a 3x1 matrix, so that's why three by one 3x1 matrix, in other words a 3x1 matrix is just a three dimensional vector.

Example

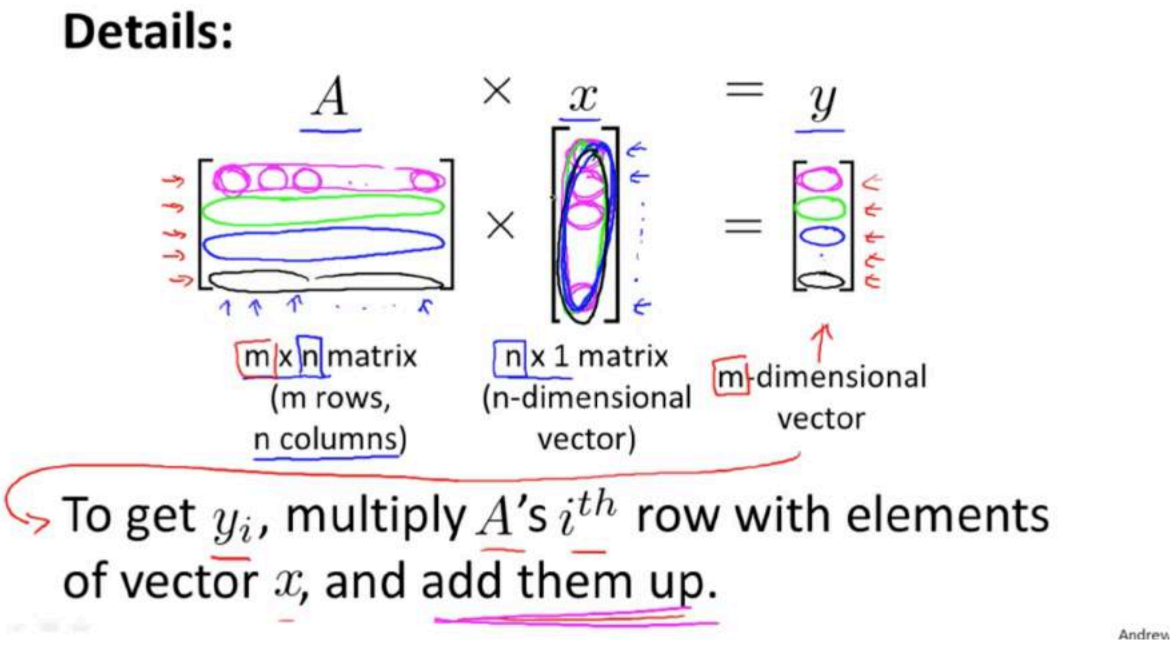
$$\underbrace{\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix}}_{\textcircled{3}\times 2} \underbrace{\begin{bmatrix} 1 \\ 5 \end{bmatrix}}_{2\times\textcircled{1}} = \underbrace{\begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}}_{3\times 1 \text{ matrix}}$$

$$1\times 1 + 3\times 5 = 16$$

$$4\times 1 + 0\times 5 = 4$$

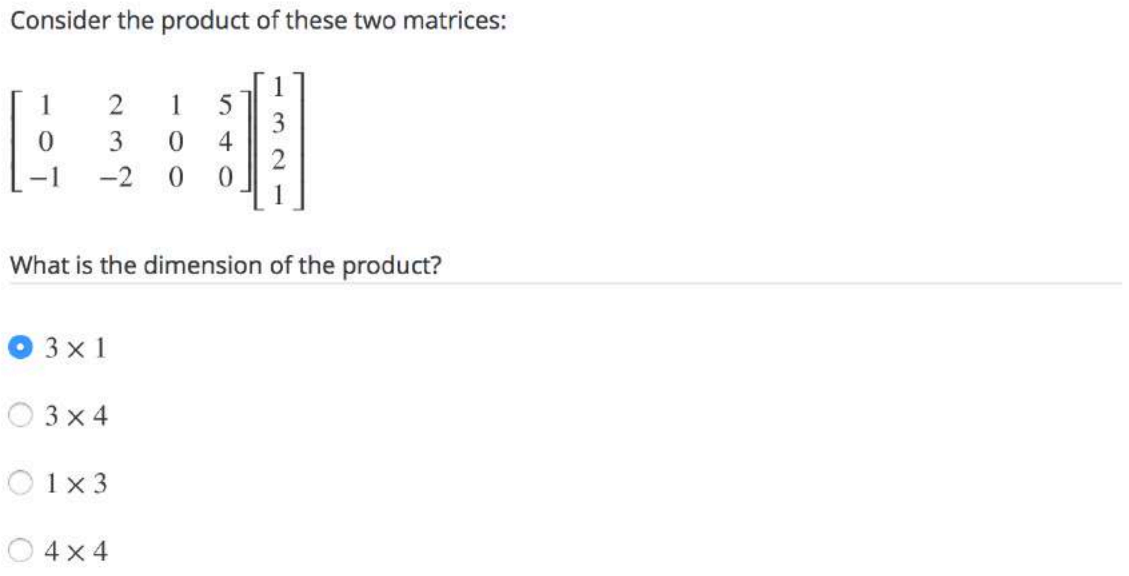
$$2\times 1 + 1\times 5 = 7$$

So I realize that I did that pretty quickly, and you're probably not sure that you can repeat this process yourself, but let's look in more detail at what just happened and what this process of multiplying a matrix by a vector looks like. Here's the details of how to multiply a matrix by a vector. Let's say I have a matrix A and want to multiply it by a vector x. The result is going to be some vector y. So the matrix A is a m by n dimensional matrix, so m rows and n columns and we are going to multiply that by a n by 1 matrix, in other words an n dimensional vector. It turns out this "n" here has to match this "n" here. In other words, the number of columns in this matrix, so it's the number of n columns. The number of columns here has to match the number of rows here. It has to match the dimension of this vector. And the result of this product is going to be an n-dimensional vector y. Rows here. "M" is going to be equal to the number of rows in this matrix "A". So how do you actually compute this vector "Y"? Well it turns out to compute this vector "Y", the process is to get "Y""I", multiply "A's" "I'th" row with the elements of the vector "X" and add them up. So here's what I mean. In order to get the first element of "Y", that first number--whatever that turns out to be--we're gonna take the first row of the matrix "A" and multiply them one at a time with the elements of this vector "X". So I take this first number multiply it by this first number. Then take the second number multiply it by this second number. Take this third number whatever that is, multiply it the third number and so on until you get to the end. And I'm gonna add up the results of these products and the result of paying that out is going to give us this first element of "Y". Then when we want to get the second element of "Y", let's say this element. The way we do that is we take the second row of A and we repeat the whole thing. So we take the second row of A, and multiply it elements-wise, so the elements of X and add up the results of the products and that would give me the second element of Y. And you keep going to get and we going to take the third row of A, multiply element Ys with the vector x, sum up the results and then I get the third element and so on, until I get down to the last row like so, okay? So that's the procedure.



To get y_i, multiply A's i^{th} row with elements of vector x, and add them up.

Question



Consider the product of these two matrices:

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -1 & -2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

What is the dimension of the product?

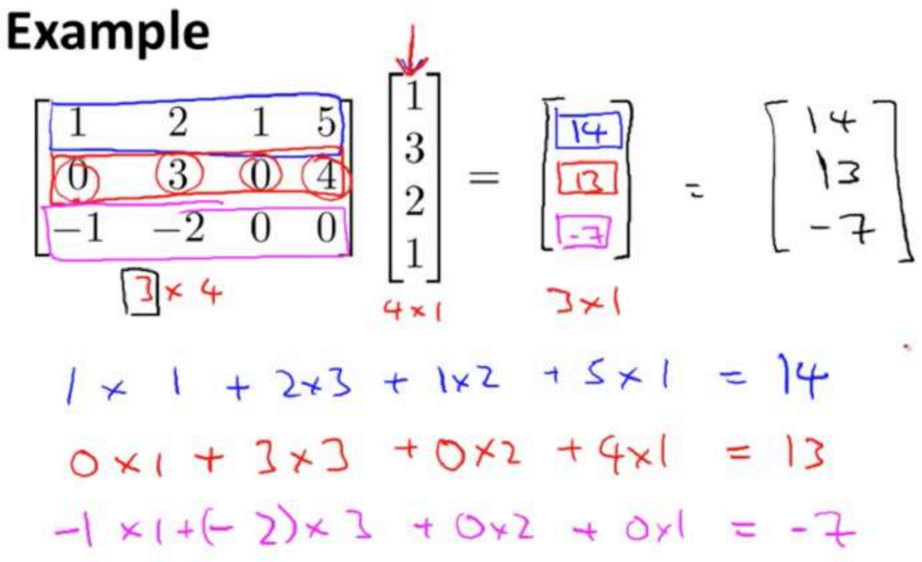
[✓ 3×1]

3×4

1×3

4×4

Let's do one more example. Here's the example: So let's look at the dimensions. Here, this is a three by four dimensional matrix. This is a four-dimensional vector, or a 4 x 1 matrix, and so the result of this, the result of this product is going to be a three-dimensional vector. Write, you know, the vector, with room for three elements. Let's do the, let's carry out the products. So for the first element, I'm going to take these four numbers and multiply them with the vector X. So I have 1x1, plus 2x3, plus 1x2, plus 5x1, which is equal to - that's 1+6, plus 2+6, which gives me 14. And then for the second element, I'm going to take this row now and multiply it with this vector (0x1)+3. All right, so 0x1+ 3x3 plus 0x2 plus 4x1, which is equal to, let's see that's 9+4, which is 13. And finally, for the last element, I'm going to take this last row, so I have minus one times one. You have minus two, or really there's a plus next to a two I guess. Times three plus zero times two plus zero times one, and so that's going to be minus one minus six, which is going to make this seven, and so that's vector seven. Okay? So my final answer is this vector fourteen, just to write to that without the colors, fourteen, thirteen, negative seven. And as promised, the result here is a three by one matrix. So that's how you multiply a matrix and a vector. I know that a lot just happened on this slide, so if you're not quite sure where all these numbers went, you know, feel free to pause the video you know, and so take a slow careful look at this big calculation that we just did and try to make sure that you understand the steps of what just happened to get us these numbers, fourteen, thirteen and eleven.



Example

$$\underbrace{\begin{bmatrix}1&2&1&5\\0&3&0&4\\-1&-2&0&0\end{bmatrix}}_{\boxed{3}\times 4}\underbrace{\begin{bmatrix}1\\3\\2\\1\end{bmatrix}}_{4\times 1}=\underbrace{\begin{bmatrix}\boxed{14}\\\boxed{13}\\\boxed{-7}\end{bmatrix}}_{3\times 1}=\begin{bmatrix}14\\13\\-7\end{bmatrix}$$

$1\times 1 + 2\times 3 + 1\times 2 + 5\times 1 = 14$

$0\times 1 + 3\times 3 + 0\times 2 + 4\times 1 = 13$

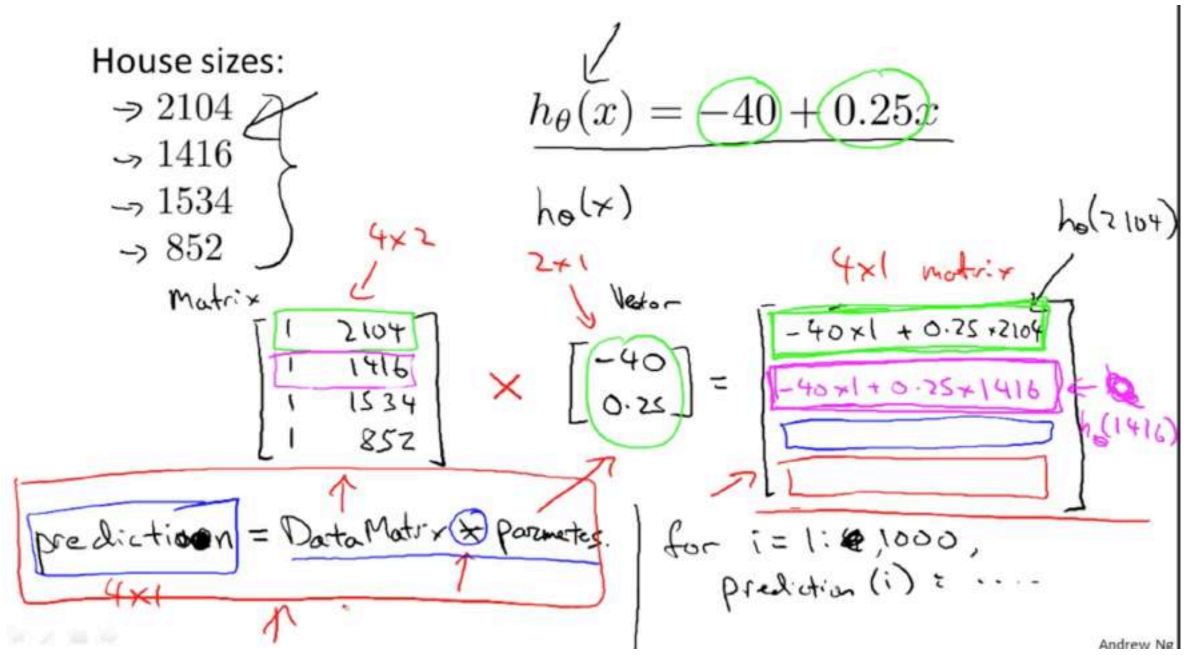
$-1\times 1 + (-2)\times 3 + 0\times 2 + 0\times 1 = -7$

Question

What is $\begin{bmatrix} 1 & 0 & 3 \\ 2 & 1 & 5 \\ 3 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 \\ 6 \\ 2 \end{bmatrix}$?

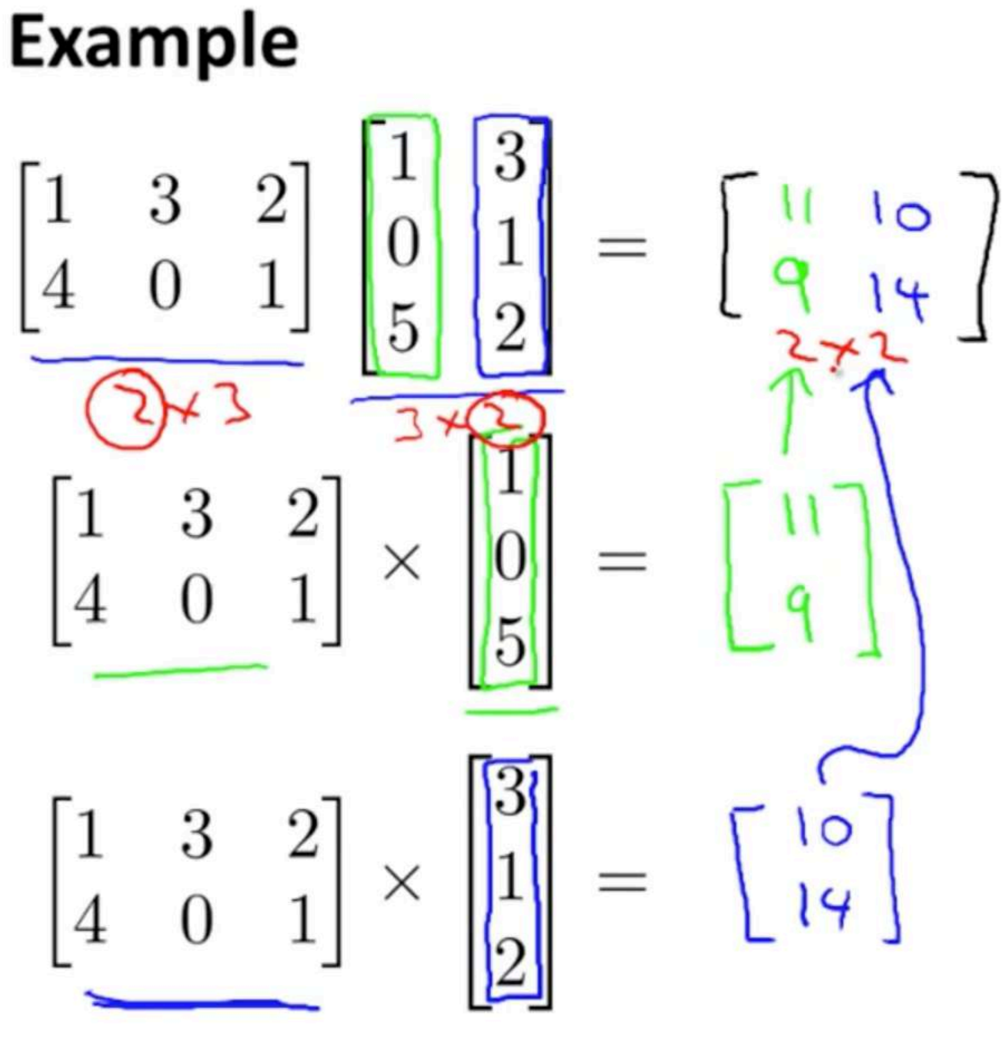
- ○ $\begin{bmatrix} 5 \\ 10 \\ 1 \end{bmatrix}$
- ○ $\begin{bmatrix} 7 \\ 12 \\ 7 \end{bmatrix}$
- [✓ $\begin{bmatrix} 7 \\ 18 \\ 13 \end{bmatrix}$]
- ○ $\begin{bmatrix} 1 \\ 18 \\ 13 \end{bmatrix}$

Finally, let me show you a neat trick. Let's say we have a set of four houses so 4 houses with 4 sizes like these. And let's say I have a hypotheses for predicting what is the price of a house, and let's say I want to compute, you know, H of X for each of my 4 houses here. It turns out there's neat way of posing this, applying this hypothesis to all of my houses at the same time. It turns out there's a neat way to pose this as a Matrix Vector multiplication. So, here's how I'm going to do it. I am going to construct a matrix as follows. My matrix is going to be 1111 times, and I'm going to write down the sizes of my four houses here and I'm going to construct a vector as well, And my vector is going to this vector of two elements, that's minus 40 and 0.25. That's these two coefficients; data 0 and data 1. And what I am going to do is to take matrix and that vector and multiply them together, that times is that multiplication symbol. So what do I get? Well this is a four by two matrix. This is a two by one matrix. So the outcome is going to be a four by one vector, all right. So, let me, so this is going to be a 4 by 1 matrix is the outcome or really a four dimensional vector, so let me write it as one of my four elements in my four real numbers here. Now it turns out and so this first element of this result, the way I am going to get that is, I am going to take this and multiply it by the vector. And so this is going to be -40 x 1 + 4.25 x 2104. By the way, on the earlier slides I was writing 1 x -40 and 2104 x 0.25, but the order doesn't matter, right? -40 x 1 is the same as 1 x -40. And this first element, of course, is "H" applied to 2104. So it's really the predicted price of my first house. Well, how about the second element? Hope you can see where I am going to get the second element. Right? I'm gonna take this and multiply it by my vector. And so that's gonna be -40 x 1 + 0.25 x 1416. And so this is going be "H" of 1416. Right? And so on for the third and the fourth elements of this 4 x 1 vector. And just there, right? This thing here that I just drew the green box around, that's a real number, OK? That's a single real number, and this thing here that I drew the magenta box around--the purple, magenta color box around--that's a real number, right? And so this thing on the right--this thing on the right overall, this is a 4 by 1 dimensional matrix, was a 4 dimensional vector. And, the neat thing about this is that when you're actually implementing this in software--so when you have four houses and when you want to use your hypothesis to predict the prices, predict the price "Y" of all of these four houses. What this means is that, you know, you can write this in one line of code. When we talk about octave and program languages later, you can actually, you'll actually write this in one line of code. You write prediction equals my, you know, data matrix times parameters, right? Where data matrix is this thing here, and parameters is this thing here, and this times is a matrix vector multiplication. And if you just do this then this variable prediction - sorry for my bad handwriting - then just implement this one line of code assuming you have an appropriate library to do matrix vector multiplication. If you just do this, then prediction becomes this 4 by 1 dimensional vector, on the right, that just gives you all the predicted prices. And your alternative to doing this as a matrix vector multiplication would be to write something like , you know, for I equals 1 to 4, right? And you have say a thousand houses it would be for I equals 1 to a thousand or whatever. And then you have to write a prediction, you know, if I equals. and then do a bunch more work over there and it turns out that When you have a large number of houses, if you're trying to predict the prices of not just four but maybe of a thousand houses then it turns out that when you implement this in the computer, implementing it like this, in any of the various languages. This is not only true for Octave, but for Supra Server Java or Python, other high-level, other languages as well. It turns out, that, by writing code in this style on the left, it allows you to not only simplify the code, because, now, you're just writing one line of code rather than the form of a bunch of things inside. But, for subtle reasons, that we will see later, it turns out to be much more computationally efficient to make predictions on all of the prices of all of your houses doing it the way on the left than the way on the right than if you were to write your own formula. I'll say more about this later when we talk about vectorization, but, so, by posing a prediction this way, you get not only a simpler piece of code, but a more efficient one. So, that's it for matrix vector multiplication and we'll make good use of these sorts of operations as we develop the living regression in other models further. But, in the next video we're going to take this and generalize this to the case of matrix matrix multiplication.

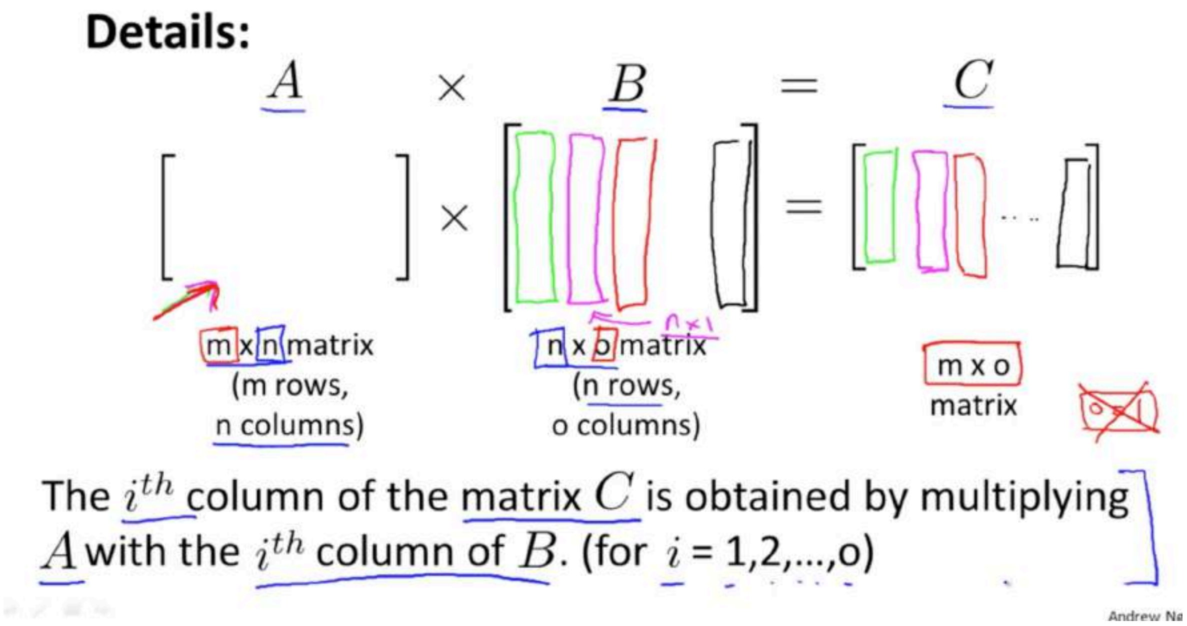


Matrix Matrix Multiplication

In this video we'll talk about matrix-matrix multiplication, or how to multiply two matrices together. When we talk about the method in linear regression for how to solve for the parameters theta 0 and theta 1 all in one shot, without needing an iterative algorithm like gradient descent. When we talk about that algorithm, it turns out that matrix-matrix multiplication is one of the key steps that you need to know. So let's, as usual, start with an example. Let's say I have two matrices and I want to multiply them together. Let me again just run through this example and then I'll tell you a little bit of what happened. So the first thing I'm gonna do is I'm going to pull out the first column of this matrix on the right. And I'm going to take this matrix on the left and multiply it by a vector that is just this first column. And it turns out, if I do that, I'm going to get the vector 11, 9. So this is the same matrix-vector multiplication as you saw in the last video. I worked this out in advance, so I know it's 11, 9. And then the second thing I want to do is I'm going to pull out the second column of this matrix on the right. And I'm then going to take this matrix on the left, so take that matrix, and multiply it by that second column on the right. So again, this is a matrix-vector multiplication step which you saw from the previous video. And it turns out that if you multiply this matrix and this vector you get 10, 14. And by the way, if you want to practice your matrix-vector multiplication, feel free to pause the video and check this product yourself. Then I'm just gonna take these two results and put them together, and that'll be my answer. So it turns out the outcome of this product is gonna be a two by two matrix. And the way I'm gonna fill in this matrix is just by taking my elements 11, 9, and plugging them here. And taking 10, 14 and plugging them into the second column, okay? So that was the mechanics of how to multiply a matrix by another matrix. You basically look at the second matrix one column at a time and you assemble the answers. And again, we'll step through this much more carefully in a second. But I just want to point out also, this first example is a 2x3 matrix. Multiply that by a 3x2 matrix, and the outcome of this product turns out to be a 2x2 matrix. And again, we'll see in a second why this was the case. All right, that was the mechanics of the calculation.


Example
$\begin{bmatrix}1&3&2\\4&0&1\end{bmatrix}\begin{bmatrix}1&3\\0&1\\5&2\end{bmatrix}=\begin{bmatrix}11&10\\9&14\end{bmatrix}$
2x3
3x2
2x2
$\begin{bmatrix}1&3&2\\4&0&1\end{bmatrix}\times\begin{bmatrix}1\\0\\5\end{bmatrix}=\begin{bmatrix}11\\9\end{bmatrix}$
$\begin{bmatrix}1&3&2\\4&0&1\end{bmatrix}\times\begin{bmatrix}3\\1\\2\end{bmatrix}=\begin{bmatrix}10\\14\end{bmatrix}$

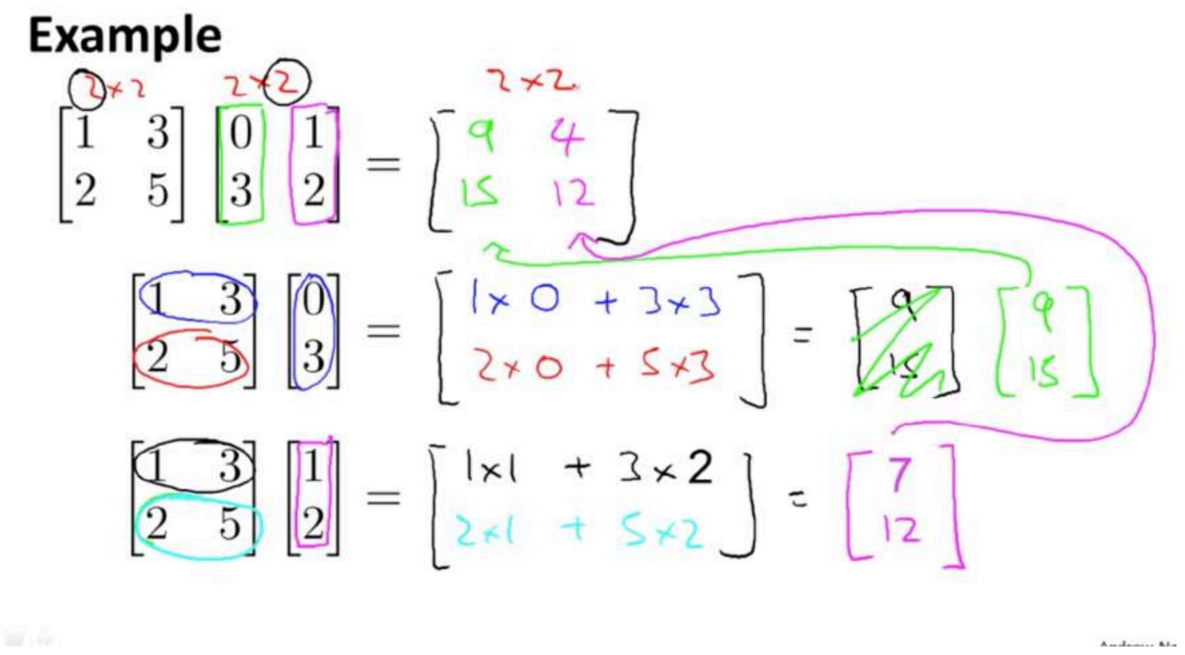
Let's actually look at the details and look at what exactly happened. Here are the details. I have a matrix A and I want to multiply that with a matrix B and the result will be some new matrix C. It turns out you can only multiply together matrices whose dimensions match. So A is an m x n matrix, so m rows, n columns. And we multiply with an n x o matrix. And it turns out this n here must match this n here. So the number of columns in the first matrix must equal to the number of rows in the second matrix. And the result of this product will be a m x o matrix, like the matrix C here. And in the previous video everything we did corresponded to the special case of o being equal to 1. That was to the case of B being a vector. But now we're gonna deal with the case of values of o larger than 1. So here's how you multiply together the two matrices. What I'm going to do is I'm going to take the first column of B and treat that as a vector, and multiply the matrix A by the first column of B. And the result of that will be a n by 1 vector, and I'm gonna put that over here. Then I'm gonna take the second column of B, right? So this is another n by 1 vector. So this column here, this is n by 1. It's an n-dimensional vector. Gonna multiply this matrix with this n by 1 vector. The result will be a m-dimensional vector, which we'll put there, and so on. And then I'm gonna take the third column, multiply it by this matrix. I get a m-dimensional vector. And so on, until you get to the last column. The matrix times the last column gives you the last column of C. Just to say that again, the ith column of the matrix C is obtained by taking the matrix A and multiplying the matrix A with the ith column of the matrix B for the values of i = 1, 2, up through o. So this is just a summary of what we did up there in order to compute the matrix C.



Details:

The i^{th} column of the matrix C is obtained by multiplying A with the i^{th} column of B. (for i = 1,2,...,o)

Let's look at just one more example. Let's say I want to multiply together these two matrices. So what I'm going to do is first pull out the first column of my second matrix. That was my matrix B on the previous slide and I therefore have this matrix times that vector. And so, oh, let's do this calculation quickly. This is going to be equal to the 1, 3 x 0, 3, so that gives 1 x 0 + 3 x 3. And the second element is going to be 2, 5 x 0, 3, so that's gonna be 2 x 0 + 5 x 3. And that is 9, 15. Oh, actually let me write that in green. So this is 9, 15. And then next I'm going to pull out the second column of this and do the corresponding calculations. So that's this matrix times this vector 1, 2. Let's also do this quickly, so that's 1 x 1 + 3 x 2, so that was that row. And let's do the other one. So let's see, that gives me 2 x 1 + 5 x 2 and so that is going to be equal to, lets see, 1 x 1 + 3 x 1 is 7 and 2 x 1 + 5 x 2 is 12. So now I have these two and so my outcome, the product of these two matrices, is going to be this goes here and this goes here. So I get 9, 15 and 4, 12. [It should be 7,12] And you may notice also that the result of multiplying a 2x2 matrix with another 2x2 matrix, the resulting dimension is going to be that first 2 times that second 2. So the result is itself also a 2x2 matrix.


Example
2×2
2×2
2×2
$\begin{bmatrix}1 & 3\\2 & 5\end{bmatrix}\begin{bmatrix}0 & 1\\3 & 2\end{bmatrix} = \begin{bmatrix}9 & 4\\15 & 12\end{bmatrix}$
$\begin{bmatrix}1 & 3\\2 & 5\end{bmatrix}\begin{bmatrix}0\\3\end{bmatrix} = \begin{bmatrix}1\times0+3\times3\\2\times0+5\times3\end{bmatrix} = \begin{bmatrix}9\\15\end{bmatrix}$
$\begin{bmatrix}1 & 3\\2 & 5\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}1\times1+3\times2\\2\times1+5\times2\end{bmatrix} = \begin{bmatrix}7\\12\end{bmatrix}$

Question

In the equation $\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 7 & 9 \\ a & b \\ c & d \end{bmatrix}$, what is a?

Hint: Compute $\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and

$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 0 \\ 3 \end{bmatrix}$.

○ 7

○ 12

[✓ 10]

○ 6

In the equation $\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 7 & 9 \\ a & b \\ c & d \end{bmatrix}$, what is b?

Hint: Compute $\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and

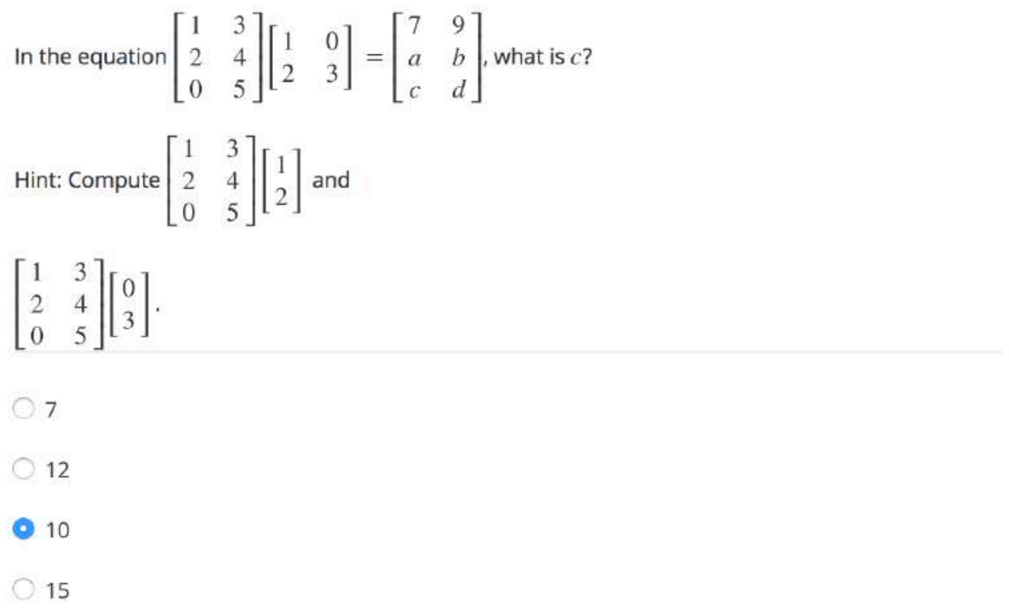
$\begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 0 & 5 \end{bmatrix}\begin{bmatrix} 0 \\ 3 \end{bmatrix}$.

○ 7

○ 10

[✓ 12]

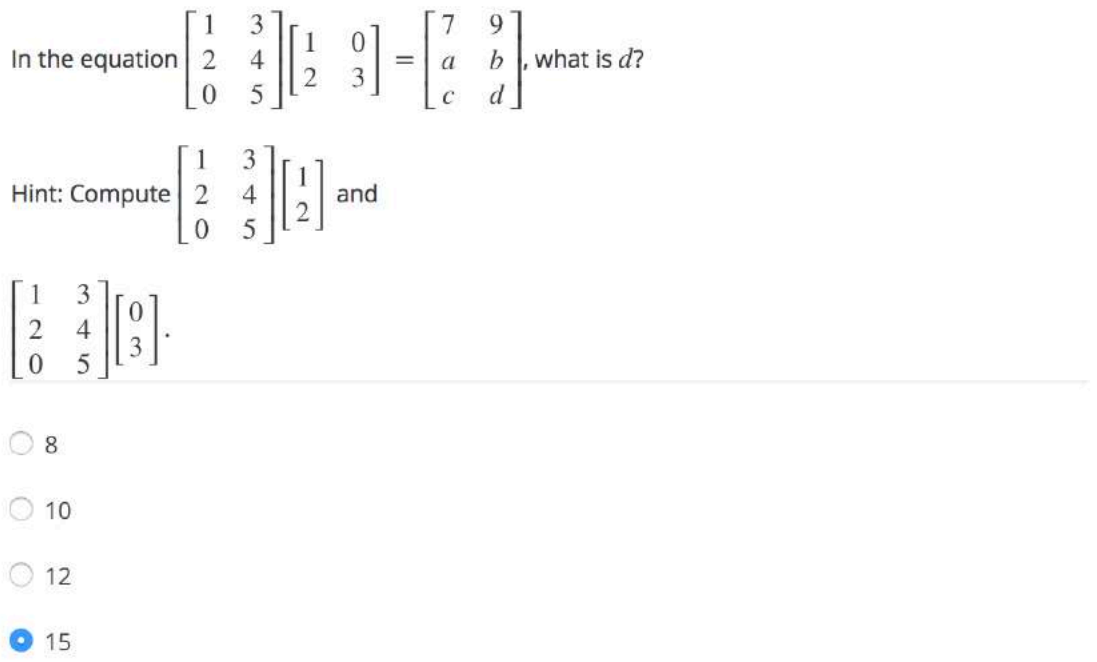
○ 15



In the equation $\begin{bmatrix}1 & 3\\ 2 & 4\\ 0 & 5\end{bmatrix}\begin{bmatrix}1 & 0\\ 2 & 3\end{bmatrix} = \begin{bmatrix}7 & 9\\ a & b\\ c & d\end{bmatrix}$, what is c?

Hint: Compute $\begin{bmatrix}1 & 3\\ 2 & 4\\ 0 & 5\end{bmatrix}\begin{bmatrix}1\\ 2\end{bmatrix}$ and $\begin{bmatrix}1 & 3\\ 2 & 4\\ 0 & 5\end{bmatrix}\begin{bmatrix}0\\ 3\end{bmatrix}$.

- 7
- 12
- [✓ 10]
- 15

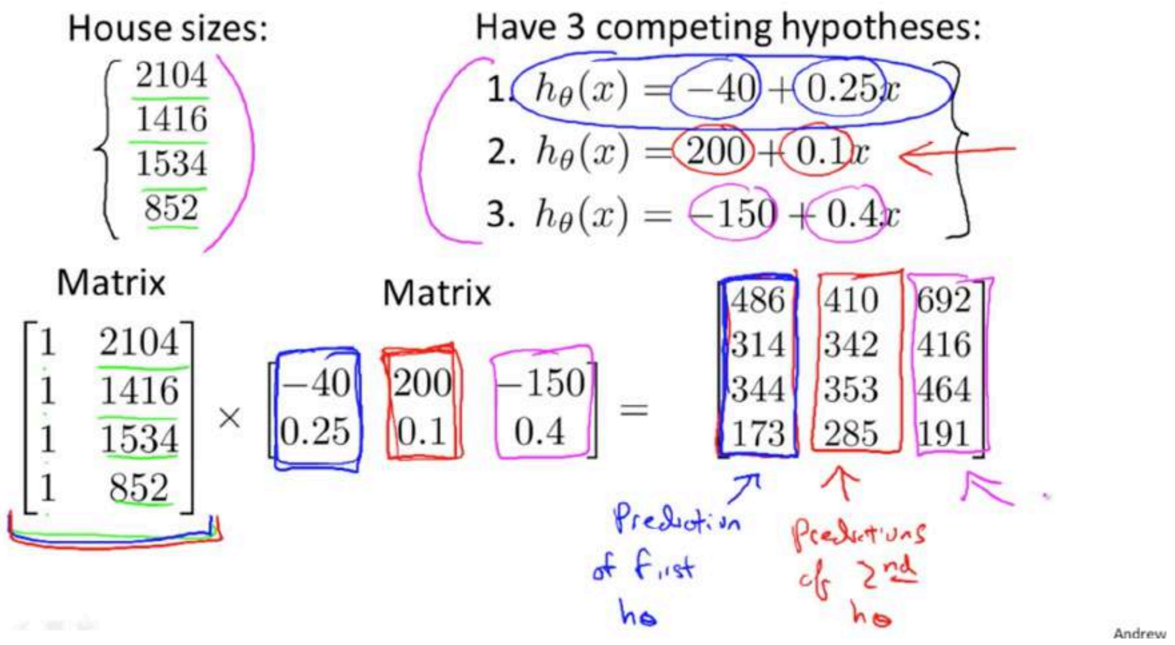


In the equation $\begin{bmatrix}1 & 3\\ 2 & 4\\ 0 & 5\end{bmatrix}\begin{bmatrix}1 & 0\\ 2 & 3\end{bmatrix} = \begin{bmatrix}7 & 9\\ a & b\\ c & d\end{bmatrix}$, what is d?

Hint: Compute $\begin{bmatrix}1 & 3\\ 2 & 4\\ 0 & 5\end{bmatrix}\begin{bmatrix}1\\ 2\end{bmatrix}$ and $\begin{bmatrix}1 & 3\\ 2 & 4\\ 0 & 5\end{bmatrix}\begin{bmatrix}0\\ 3\end{bmatrix}$.

- 8
- 10
- 12
- [✓ 15]

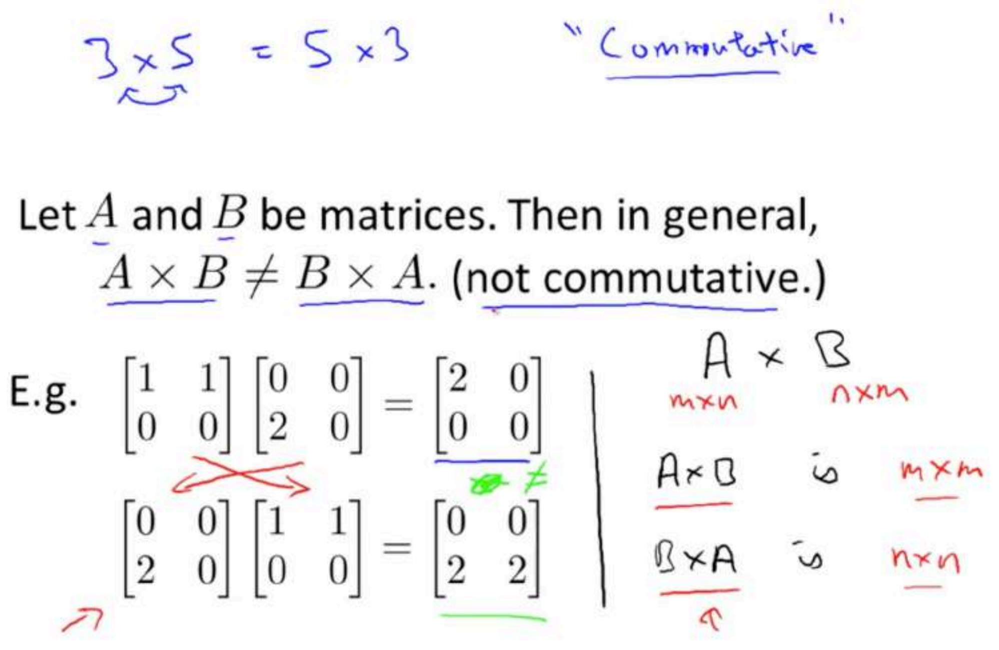
Finally, let me show you one more neat trick that you can do with matrix-matrix multiplication. Let's say, as before, that we have four houses whose prices we wanna predict. Only now, we have three competing hypotheses shown here on the right. So if you want to apply all three competing hypotheses to all four of your houses, it turns out you can do that very efficiently using a matrix-matrix multiplication. So here on the left is my usual matrix, same as from the last video where these values are my housing prices [he means housing sizes] and I've put 1s here on the left as well. And what I am going to do is construct another matrix where here, the first column is this -40 and 0.25 and the second column is this 200, 0.1 and so on. And it turns out that if you multiply these two matrices, what you find is that this first column, I'll draw that in blue. Well, how do you get this first column? Our procedure for matrix-matrix multiplication is, the way you get this first column is you take this matrix and you multiply it by this first column. And we saw in the previous video that this is exactly the predicted housing prices of the first hypothesis, right, of this first hypothesis here.And how about the second column? Well, [INAUDIBLE] second column. The way you get the second column is, well, you take this matrix and you multiply it by this second column. And so the second column turns out to be the predictions of the second hypothesis up there, and similarly for the third column. And so I didn't step through all the details, but hopefully you can just feel free to pause the video and check the math yourself and check that what I just claimed really is true. But it turns out that by constructing these two matrices, what you can therefore do is very quickly apply all 3 hypotheses to all 4 house sizes to get all 12 predicted prices output by your 3 hypotheses on your 4 houses. So with just one matrix multiplication step you managed to make 12 predictions.


House sizes:
2104
1416
1534
852
Have 3 competing hypotheses:
1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$
Matrix
Matrix
$\begin{bmatrix}1 & 2104\\1 & 1416\\1 & 1534\\1 & 852\end{bmatrix} \times \begin{bmatrix}-40 & 200 & -150\\0.25 & 0.1 & 0.4\end{bmatrix} = \begin{bmatrix}486 & 410 & 692\\314 & 342 & 416\\344 & 353 & 464\\173 & 285 & 191\end{bmatrix}$
Prediction of first hθ
Predictions of 2nd hθ
Andrew

And even better, it turns out that in order to do that matrix multiplication, there are lots of good linear algebra libraries in order to do this multiplication step for you. And so pretty much any reasonable programming language that you might be using. Certainly all the top ten most popular programming languages will have great linear algebra libraries. And there'll be good linear algebra libraries that are highly optimized in order to do that matrix-matrix multiplication very efficiently. Including taking advantage of any sort of parallel computation that your computer may be capable of, whether your computer has multiple cores or multiple processors. Or within a processor sometimes there's parallelism as well called SIMD parallelism that your computer can take care of. And there are very good free libraries that you can use to do this matrix-matrix multiplication very efficiently, so that you can very efficiently make lots of predictions with lots of hypotheses.

Matrix Multiplication properties

Matrix multiplication is really useful, since you can pack a lot of computation into just one matrix multiplication operation. But you should be careful of how you use them. In this video, I wanna tell you about a few properties of matrix multiplication. When working with just real numbers or when working with scalars, multiplication is commutative. And what I mean by that is that if you take 3 times 5, that is equal to 5 times 3. And the ordering of this multiplication doesn't matter. And this is called the commutative property of multiplication of real numbers. It turns out this property, they can reverse the order in which you multiply things. This is not true for matrix multiplication. So concretely, if A and B are matrices. Then in general, A times B is not equal to B times A. So, just be careful of that. Its not okay to arbitrarily reverse the order in which you multiply matrices. Matrix multiplication in not commutative, is the fancy way of saying it. As a concrete example, here are two matrices. This matrix 1 1 0 0 times 0 0 2 0 and if you multiply these two matrices you get this result on the right. Now let's swap around the order of these two matrices. So I'm gonna take this two matrices and just reverse them. It turns out if you multiply these two matrices, you get the second answer on the right. And well clearly, right, these two matrices are not equal to each other. So, in fact, in general if you have a matrix operation like A times B, if A is an m by n matrix, and B is an n by m matrix, just as an example. Then, it turns out that the matrix A times B, right, is going to be an m by m matrix. Whereas the matrix B times A is going to be an n by n matrix. So the dimensions don't even match, right? So if A x B and B x A may not even be the same dimension. In the example on the left, I have all two by two matrices. So the dimensions were the same, but in general, reversing the order of the matrices can even change the dimension of the outcome. So, matrix multiplication is not commutative.



$3 \times 5 = 5 \times 3$ "Commutative"

Let A and B be matrices. Then in general,

$A \times B \neq B \times A$. (not commutative.)

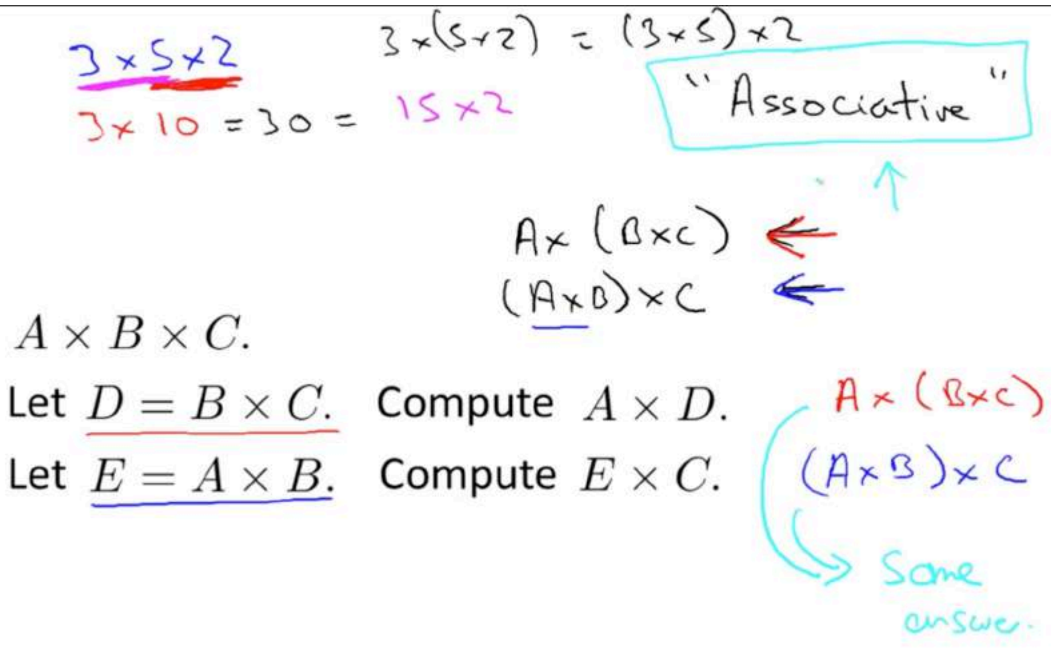
E.g.
$$\begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}\begin{bmatrix}0 & 0\\2 & 0\end{bmatrix} = \begin{bmatrix}2 & 0\\0 & 0\end{bmatrix} \neq$$
$$\begin{bmatrix}0 & 0\\2 & 0\end{bmatrix}\begin{bmatrix}1 & 1\\0 & 0\end{bmatrix} = \begin{bmatrix}0 & 0\\2 & 2\end{bmatrix}$$

$\underset{m\times n}{A} \times \underset{n\times m}{B}$

$A\times B$ is $m\times m$

$B\times A$ is $n\times n$

Here's the next property I want to talk about. So, when talking about real numbers or scalars, let's say I have 3 x 5 x 2. I can either multiply 5 x 2 first. Then I can compute this as 3 x 10. Or, I can multiply 3 x 5 first, and I can compute this as 15 x 2. And both of these give you the same answer, right? Both of these is equal to 30. So it doesn't matter whether I multiply 5 x 2 first or whether I multiply 3 x 5 first, because sort of, well, 3 x (5 x 2) = (3 x 5) x 2. And this is called the associative property of real number multiplication. It turns out that matrix multiplication is associative. So concretely, let's say I have a product of three matrices A x B x C. Then, I can compute this either as A x (B x C) or I can computer this as (A x B) x C, and these will actually give me the same answer. I'm not gonna prove this but you can just take my word for it I guess. So just be clear, what I mean by these two cases. Let's look at the first one, right. This first case. What I mean by that is if you actually wanna compute A x B x C. What you can do is you can first compute B x C. So that D = B x C then compute A x D. And so this here is really computing A x B x C. Or, for this second case, you can compute this as, you can set E = A x B, then compute E times C. And this is then the same as A x B x C, and it turns out that both of these options will give you this guarantee to give you the same answer. And so we say that matrix multiplication thus enjoy the associative property. Okay? And don't worry about the terminology associative and commutative. That's what it's called, but I'm not really going to use this terminology later in this class, so don't worry about memorizing those terms.

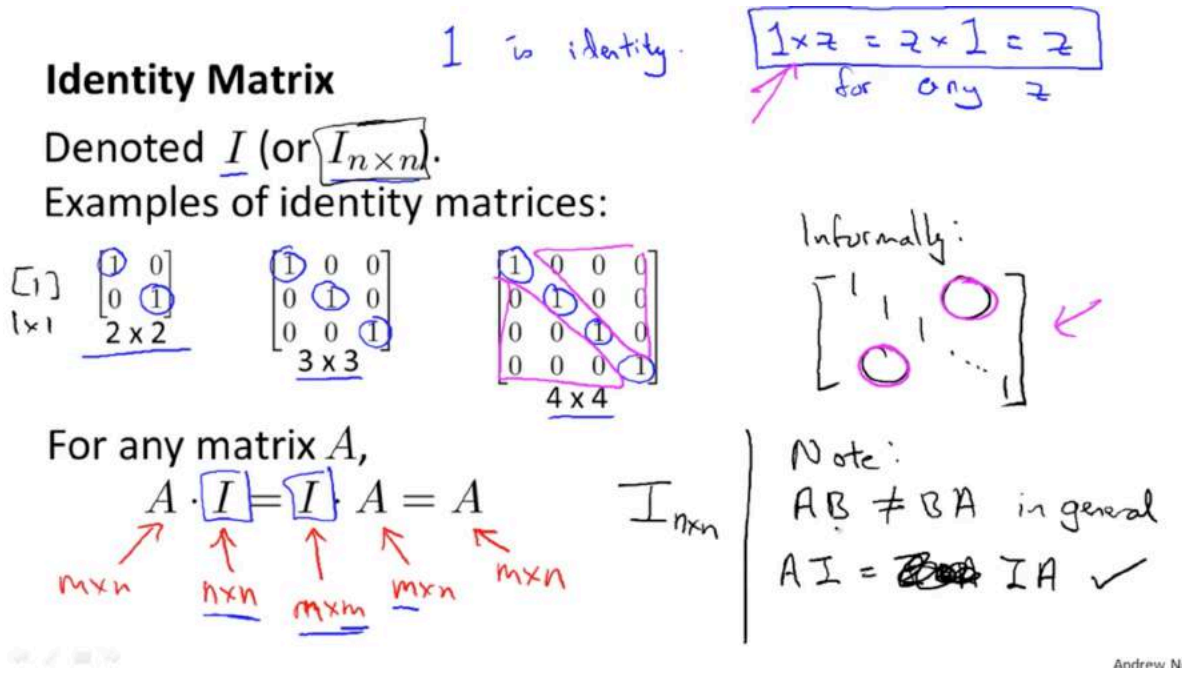


$A \times B \times C.$

Let $D = B \times C.$ Compute $A \times D.$

Let $E = A \times B.$ Compute $E \times C.$

Finally, I want to tell you about the Identity Matrix, which is a special matrix. So let's again make the analogy to what we know of real numbers. When dealing with real numbers or scalar numbers, the number 1, you can think of it as the identity of multiplication. And what I mean by that is that for any number z, 1 x z = z x 1. And that's just equal to the number z for any real number z. So 1 is the identity operation and so it satisfies this equation. So it turns out, that this in the space of matrices there's an identity matrix as well and it's usually denoted I or sometimes we write it as I of n x n if we want to make it explicit to dimensions. So I subscript n x n is the n x n identity matrix. And so that's a different identity matrix for each dimension n. And here are few examples. Here's the 2 x 2 identity matrix, here's the 3 x 3 identity matrix, here's the 4 x 4 matrix. So the identity matrix has the property that it has ones along the diagonals. All right, and so on. And 0 everywhere else. And so, by the way, the 1 x 1 identity matrix is just a number 1, and so the 1 x 1 matrix with just 1 in it. So it's not a very interesting identity matrix. And informally, when I or others are being sloppy, very often we'll write the identity matrices in fine notation. We'll draw square brackets, just write one one one dot dot dot dot one, and then we'll maybe somewhat sloppily write a bunch of zeros there. And these zeroes on the, this big zero and this big zero, that's meant to denote that this matrix is zero everywhere except for the diagonal. So this is just how I might swap you the right D identity matrix. And it turns out that the identity matrix has its property that for any matrix A, A times identity equals I times A equals A so that's a lot like this equation that we have up here. Right? So 1 times z equals z times 1 equals z itself. So I times A equals A times I equals A. Just to make sure we have the dimensions right. So if A is an m by n matrix, then this identity matrix here, that's an n by n identity matrix. And if is and by then, then this identity matrix, right? For matrix multiplication to make sense, that has to be an m by m matrix. Because this m has the match up that m, and in either case, the outcome of this process is you get back the matrix A which is m by n. So whenever we write the identity matrix I, you know, very often the dimension Mention, right, will be implicit from the content. So these two I's, they're actually different dimension matrices. One may be n by n, the other is n by m. But when we want to make the dimension of the matrix explicit, then sometimes we'll write to this I subscript n by n, kind of like we had up here. But very often, the dimension will be implicit. Finally, I just wanna point out that earlier I said that AB is not, in general, equal to BA. Right? For most matrices A and B, this is not true. But when B is the identity matrix, this does hold true, that A times the identity matrix does indeed equal to identity times A is just that you know this is not true for other matrices B in general.



Identity Matrix

1 is identity.

$\boxed{1 \times z = z \times 1 = z}$ for any z

Denoted I (or $\boxed{I_{n\times n}}$).

Examples of identity matrices:

$[1]$ 1×1 $\quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ 2×2 $\quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 3×3 $\quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ 4×4

Informally:

$$\begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & 1 & \\ 0 & & & \ddots \; 1 \end{bmatrix}$$

For any matrix A,

$$\underset{m\times n}{A} \cdot \underset{n\times n}{\boxed{I}} = \underset{m\times m}{\boxed{I}} \cdot \underset{m\times n}{A} = \underset{m\times n}{A}$$

$I_{n\times n}$

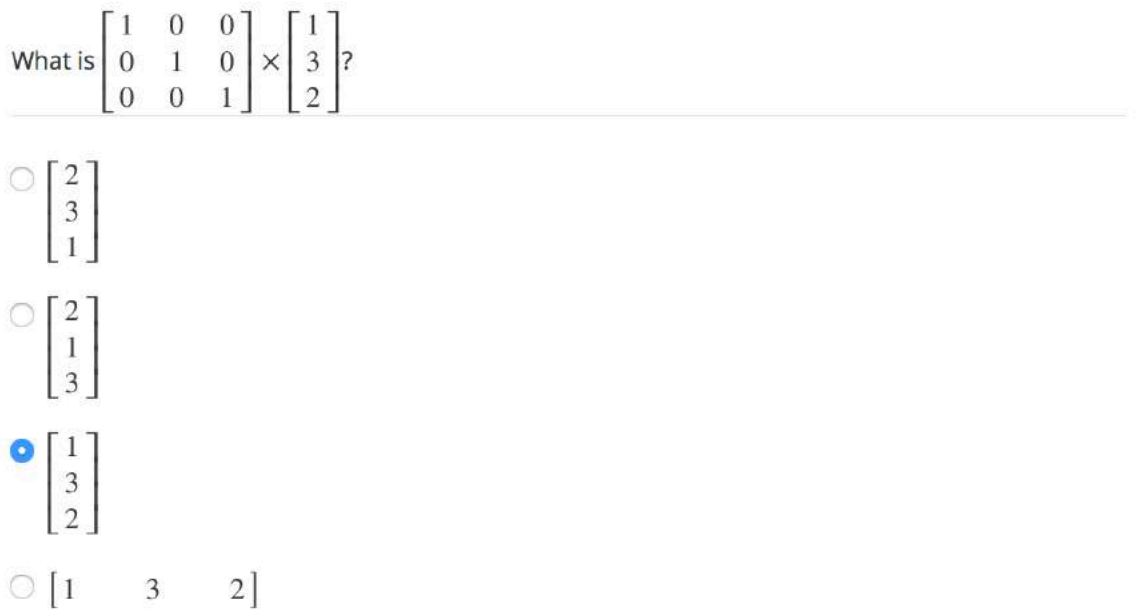
Note:

$AB \neq BA$ in general

$AI = IA$ ✓

Andrew N

Question



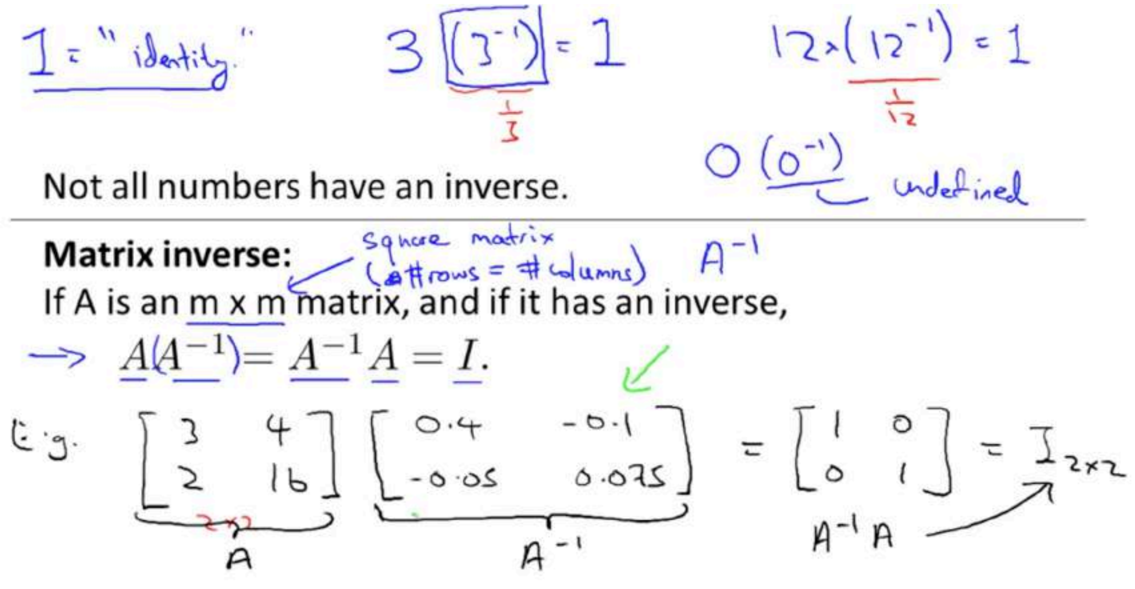
What is $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$?

- $\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$
- $\begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$
- [✓ $\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$]
- $\begin{bmatrix} 1 & 3 & 2 \end{bmatrix}$

So, that's it for the properties of matrix multiplication and special matrices like the identity matrix I want to tell you about. In the next and final video on our linear algebra review, I'm going to quickly tell you about a couple of special matrix operations and after that everything you need to know about linear algebra for this class.

Inverse and Transpose

In this video, I want to tell you about a couple of special matrix operations, called the matrix inverse and the matrix transpose operation. Let's start by talking about matrix inverse, and as usual we'll start by thinking about how it relates to real numbers. In the last video, I said that the number one plays the role of the identity in the space of real numbers because one times anything is equal to itself. It turns out that real numbers have this property that very number have an, that each number has an inverse, for example, given the number three, there exists some number, which happens to be three inverse so that that number times gives you back the identity element one. And so to me, inverse of course this is just one third. And given some other number, maybe twelve there is some number which is the inverse of twelve written as twelve to the minus one, or really this is just one twelve. So that when you multiply these two things together. the product is equal to the identity element one again. Now it turns out that in the space of real numbers, not everything has an inverse. For example the number zero does not have an inverse, right? Because zero's a zero inverse, one over zero that's undefined. Like this one over zero is not well defined. And what we want to do, in the rest of this slide, is figure out what does it mean to compute the inverse of a matrix. Here's the idea: If A is a n by n matrix, and it has an inverse, I will say a bit more about that later, then the inverse is going to be written A to the minus one and A times this inverse, A to the minus one, is going to equal to A inverse times A, is going to give us back the identity matrix. Okay? Only matrices that are m by m for some the idea of M having inverse. So, a matrix is M by M, this is also called a square matrix and it's called square because the number of rows is equal to the number of columns. Right and it turns out only square matrices have inverses, so A is a square matrix, is m by m, on inverse this equation over here. Let's look at a concrete example, so let's say I have a matrix, three, four, two, sixteen. So this is a two by two matrix, so it's a square matrix and so this may just could have an and it turns out that I happen to know the inverse of this matrix is zero point four, minus zero point one, minus zero point zero five, zero zero seven five. And if I take this matrix and multiply these together it turns out what I get is the two by two identity matrix, I, this is I two by two. Okay? And so on this slide, you know this matrix is the matrix A, and this matrix is the matrix A-inverse. And it turns out if that you are computing A times A-inverse, it turns out if you compute A-inverse times A you also get back the identity matrix. So how did I find this inverse or how did I come up with this inverse over here? It turns out that sometimes you can compute inverses by hand but almost no one does that these days. And it turns out there is very good numerical software for taking a matrix and computing its inverse. So again, this is one of those things where there are lots of open source libraries that you can link to from any of the popular programming languages to compute inverses of matrices.



$\underline{1 = \text{"identity"}}$ $\quad 3\,\underbrace{\boxed{(3^{-1})}}_{\frac{1}{3}} = 1 \quad 12 \times \underbrace{(12^{-1})}_{\frac{1}{12}} = 1$

Not all numbers have an inverse. $\quad 0\,\underbrace{(0^{-1})}$ undefined

Matrix inverse: square matrix (#rows = #columns) $\quad A^{-1}$

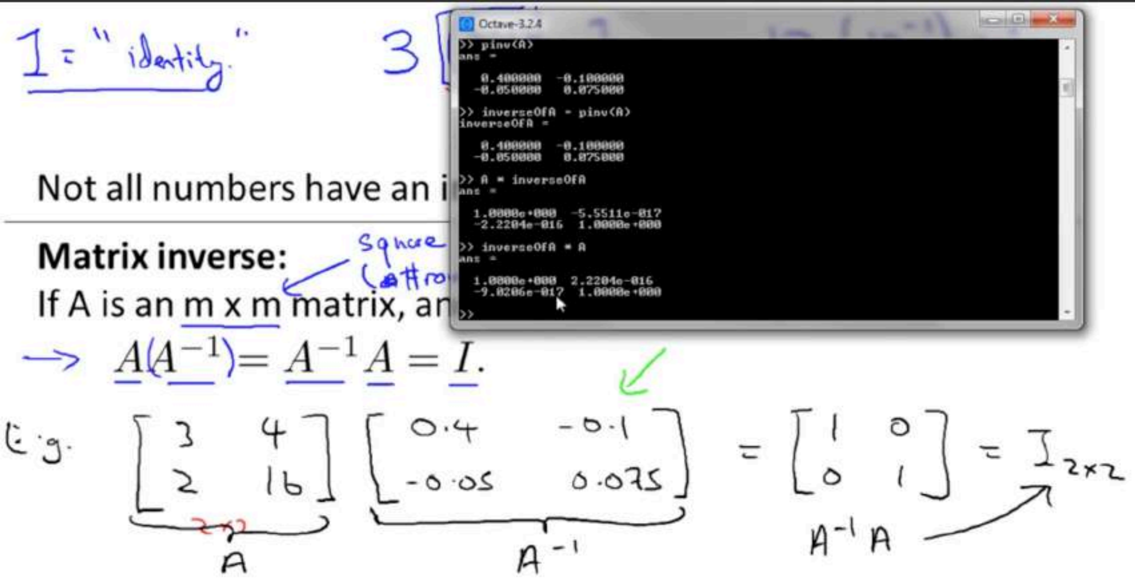
If A is an $\underline{m \times m}$ matrix, and if it has an inverse,

$$\rightarrow \underline{A}(\underline{A^{-1}}) = \underline{A^{-1}}\,\underline{A} = \underline{I}.$$

E.g. $$\underbrace{\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix}}_{\substack{2\times 2 \\ A}} \underbrace{\begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix}}_{A^{-1}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2\times 2}$$

$A^{-1}A \rightarrow I_{2\times 2}$

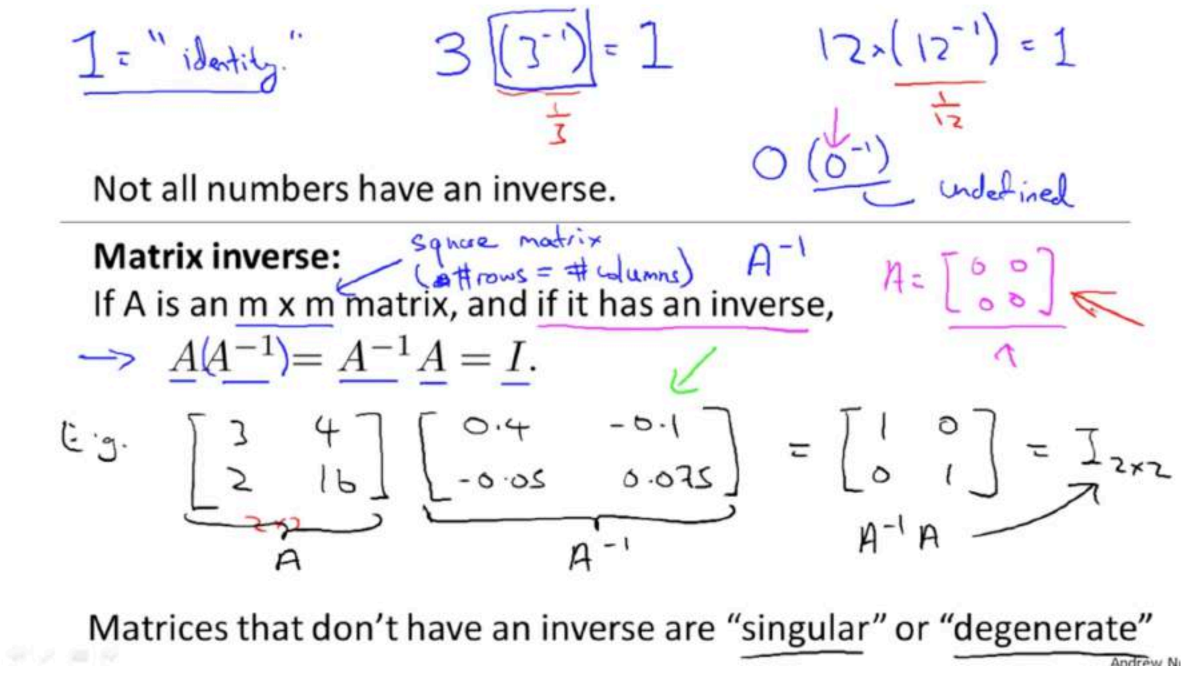
Let me show you a quick example. How I actually computed this inverse, and what I did was I used software called Octave. So let me bring that up. We will see a lot about Octave later. Let me just quickly show you an example. Set my matrix A to be equal to that matrix on the left, type three four two sixteen, so that's my matrix A right. This is matrix 34, 216 that I have down here on the left. And, the software lets me compute the inverse of A very easily. It's like P over A equals this. And so, this is right, this matrix here on my four minus, on my one, and so on. This given the numerical solution to what is the inverse of A. So let me just write, inverse of A equals P inverse of A over that I can now just verify that A times A inverse the identity is, type A times the inverse of A and the result of that is this matrix and this is one one on the diagonal and essentially ten to the minus seventeen, ten to the minus sixteen, so Up to numerical precision, up to a little bit of round off error that my computer had in finding optimal matrices and these numbers off the diagonals are essentially zero so A times the inverse is essentially the identity matrix. Can also verify the inverse of A times A is also equal to the identity, ones on the diagonals and values that are essentially zero except for a little bit of round dot error on the off diagonals.



I = "identity"
Not all numbers have an i
Matrix inverse:
If A is an m x m matrix, an
square
(# ro
$A(A^{-1}) = A^{-1}A = I.$
E.g.
$\underbrace{\begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix}}_{A}$ $\underbrace{\begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix}}_{A^{-1}}$ $= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2\times2}$
2x2
$A^{-1}A$
Octave-3.2.4

```
>> pinv(A)
ans =

   0.400000  -0.100000
  -0.050000   0.075000

>> inverseOfA = pinv(A)
inverseOfA =

   0.400000  -0.100000
  -0.050000   0.075000

>> A * inverseOfA
ans =

  1.0000e+000  -5.5511e-017
 -2.2204e-016   1.0000e+000

>> inverseOfA * A
ans =

  1.0000e+000  2.2204e-016
 -9.0206e-017  1.0000e+000

>>
```

In my definition of the inverse of a matrix is, I had this caveat first it must always be a square matrix, it had this caveat, that if A has an inverse, exactly what matrices have an inverse is beyond the scope of this linear algebra for review that one intuition you might take away that just as the number zero doesn't have an inverse, it turns out that if A is say the matrix of all zeros, then this matrix A also does not have an inverse because there's no matrix there's no A inverse matrix so that this matrix times some other matrix will give you the identity matrix so this matrix of all zeros, and there are a few other matrices with properties similar to this. That also don't have an inverse. But it turns out that in this review I don't want to go too deeply into what it means matrix have an inverse but it turns out for our machine learning application this shouldn't be an issue or more precisely for the learning algorithms where this may be an to namely whether or not an inverse matrix appears and I will tell when we get to those learning algorithms just what it means for an algorithm to have or not have an inverse and how to fix it in case. Working with matrices that don't have inverses. But the intuition if you want is that you can think of matrices as not have an inverse that is somehow too close to zero in some sense. So, just to wrap up the terminology, matrix that don't have an inverse Sometimes called a singular matrix or degenerate matrix and so this matrix over here is an example zero zero zero matrix. is an example of a matrix that is singular, or a matrix that is degenerate.



Not all numbers have an inverse.

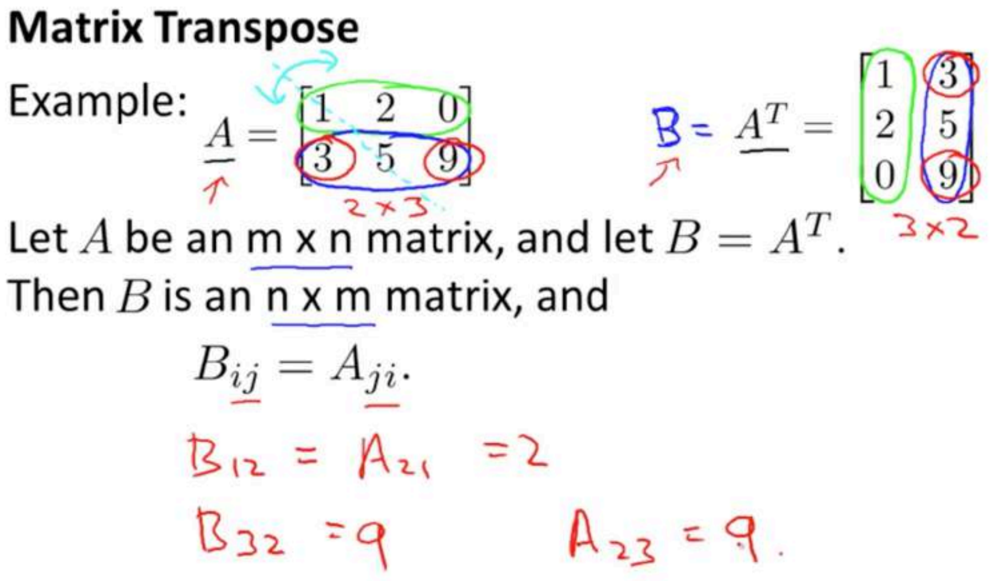
Matrix inverse:

If A is an m x m matrix, and if it has an inverse,

$$A(A^{-1}) = A^{-1}A = I.$$

Matrices that don't have an inverse are "singular" or "degenerate"

Finally, the last special matrix operation I want to tell you about is to do matrix transpose. So suppose I have matrix A, if I compute the transpose of A, that's what I get here on the right. This is a transpose which is written and A superscript T, and the way you compute the transpose of a matrix is as follows. To get a transpose I am going to first take the first row of A one to zero. That becomes this first column of this transpose. And then I'm going to take the second row of A, 3 5 9, and that becomes the second column. of the matrix A transpose. And another way of thinking about how the computer transposes is as if you're taking this sort of 45 degree axis and you are mirroring or you are flipping the matrix along that 45 degree axis. so here's the more formal definition of a matrix transpose. Let's say A is a m by n matrix. And let's let B equal A transpose and so BA transpose like so. Then B is going to be a n by m matrix with the dimensions reversed so here we have a 2x3 matrix. And so the transpose becomes a 3x2 matrix, and moreover, the BIJ is equal to AJI. So the IJ element of this matrix B is going to be the JI element of that earlier matrix A. So for example, B 1 2 is going to be equal to, look at this matrix, B 1 2 is going to be equal to this element 3 1st row, 2nd column. And that equal to this, which is a two one, second row first column, right, which is equal to two and some [It should be 3] of the example B 3 2, right, that's B 3 2 is this element 9, and that's equal to a two three which is this element up here, nine. And so that wraps up the definition of what it means to take the transpose of a matrix and that in fact concludes our linear algebra review.



Matrix Transpose

Example: $A = \underbrace{\begin{bmatrix}1 & 2 & 0\\ 3 & 5 & 9\end{bmatrix}}_{2\times 3}$ $\qquad B = A^T = \underbrace{\begin{bmatrix}1 & 3\\ 2 & 5\\ 0 & 9\end{bmatrix}}_{3\times 2}$

Let A be an m x n matrix, and let $B = A^T$.
Then B is an n x m matrix, and

$$B_{ij} = A_{ji}.$$

$B_{12} = A_{21} = 2$

$B_{32} = 9 \qquad A_{23} = 9.$

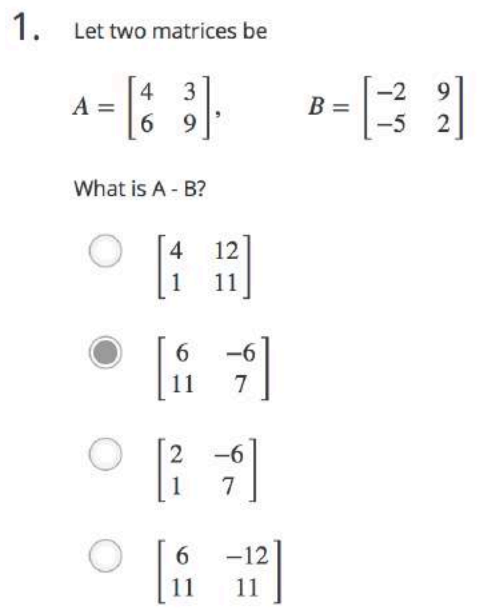
So by now hopefully you know how to add and subtract matrices as well as multiply them and you also know how, what are the definitions of the inverses and transposes of a matrix and these are the main operations used in linear algebra for this course. In case this is the first time you are seeing this material. I know this was a lot of linear algebra material all presented very quickly and it's a lot to absorb but if you there's no need to memorize all the definitions we just went through and if you download the copy of either these slides or of the lecture notes from the course website. and use either the slides or the lecture notes as a reference then you can always refer back to the definitions and to figure out what are these matrix multiplications, transposes and so on definitions. And the lecture notes on the course website also has pointers to additional resources linear algebra which you can use to learn more about linear algebra by yourself.

And next with these new tools. We'll be able in the next few videos to develop more powerful forms of linear regression that can view of a lot more data, a lot more features, a lot more training examples and later on after the new regression we'll actually continue using these linear algebra tools to derive more powerful learning algorithms as well.

Review

Practice Quiz: Linear Algebra

1.



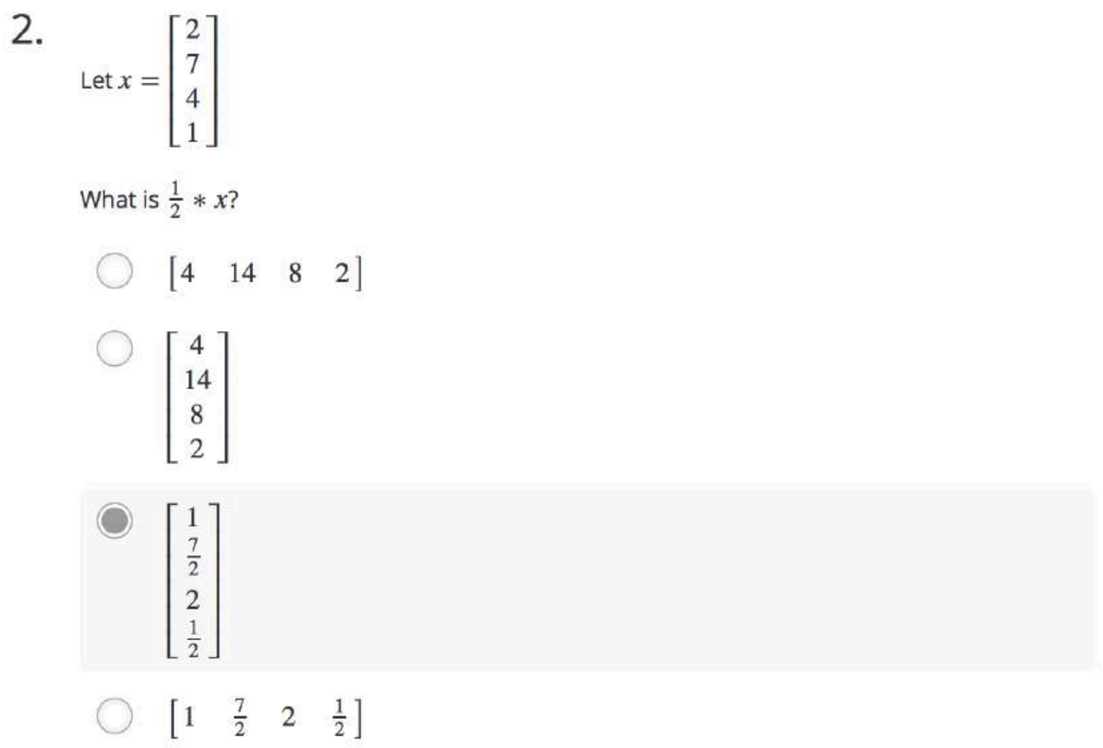
1. Let two matrices be

$$A = \begin{bmatrix} 4 & 3 \\ 6 & 9 \end{bmatrix}, \qquad B = \begin{bmatrix} -2 & 9 \\ -5 & 2 \end{bmatrix}$$

What is A - B?

- ○ $\begin{bmatrix} 4 & 12 \\ 1 & 11 \end{bmatrix}$
- [✓ $\begin{bmatrix} 6 & -6 \\ 11 & 7 \end{bmatrix}$]
- ○ $\begin{bmatrix} 2 & -6 \\ 1 & 7 \end{bmatrix}$
- ○ $\begin{bmatrix} 6 & -12 \\ 11 & 11 \end{bmatrix}$

2.

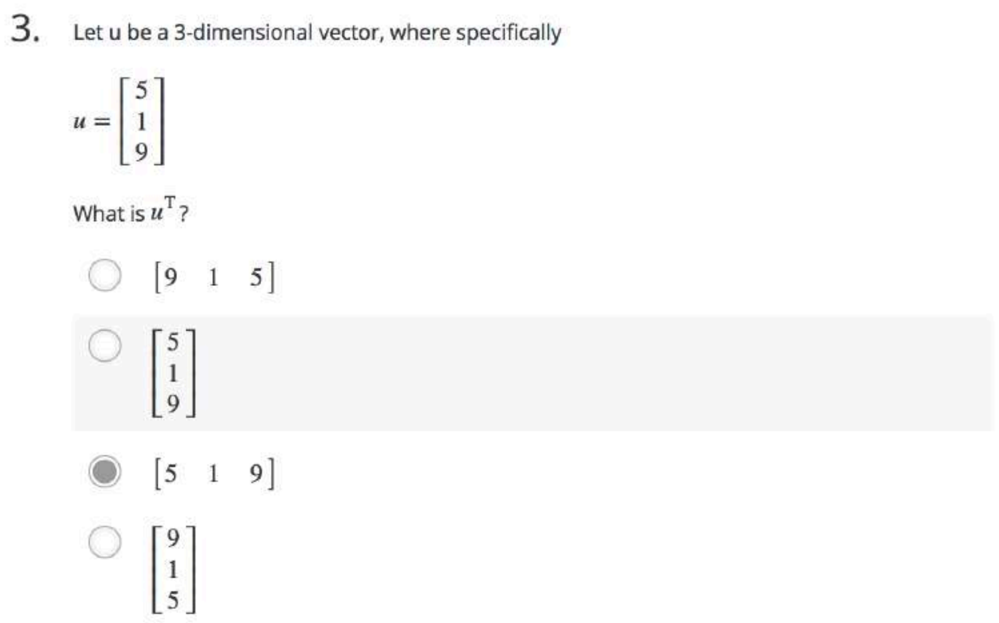


2. Let $x = \begin{bmatrix} 2 \\ 7 \\ 4 \\ 1 \end{bmatrix}$

What is $\frac{1}{2} * x$?

- $\begin{bmatrix} 4 & 14 & 8 & 2 \end{bmatrix}$
- $\begin{bmatrix} 4 \\ 14 \\ 8 \\ 2 \end{bmatrix}$
- [✓ $\begin{bmatrix} 1 \\ \frac{7}{2} \\ 2 \\ \frac{1}{2} \end{bmatrix}$]
- $\begin{bmatrix} 1 & \frac{7}{2} & 2 & \frac{1}{2} \end{bmatrix}$

3.



3. Let u be a 3-dimensional vector, where specifically

$u = \begin{bmatrix} 5 \\ 1 \\ 9 \end{bmatrix}$

What is u^{T}?

- $\begin{bmatrix} 9 & 1 & 5 \end{bmatrix}$
- $\begin{bmatrix} 5 \\ 1 \\ 9 \end{bmatrix}$
- [✓ $\begin{bmatrix} 5 & 1 & 9 \end{bmatrix}$]
- $\begin{bmatrix} 9 \\ 1 \\ 5 \end{bmatrix}$

4.

4. Let u and v be 3-dimensional vectors, where specifically

$$u = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix}$$

and

$$v = \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix}$$

What is $u^T v$?

(Hint: u^T is a

1x3 dimensional matrix, and v can also be seen as a 3x1

matrix. The answer you want can be obtained by taking

the matrix product of u^T and v.) Do not add brackets to your answer.

4

5.

5. Let A and B be 3x3 (square) matrices. Which of the following

must necessarily hold true? Check all that apply.

☐ $A * B * A = B * A * B$

☐ $A * B = B * A$

[✓ $A + B = B + A$]

[✓ If A is the 3x3 identity matrix, then $A * B = B * A$]

WEEK 2

Environment Setup Instructions

Setting Up Your Programming Assignment Environment

The Machine Learning course includes several programming assignments which you'll need to finish to complete the course. The assignments require the Octave or MATLAB scientific computing languages.

- Octave is a free, open-source application available for many platforms. It has a text interface and an experimental graphical one.
- MATLAB is proprietary software but a free, limited license is being offered for the completion of this course.

There are several subtle differences between the two software packages. MATLAB may offer a smoother experience (especially for Mac users), contains a larger number of functions, and can be more robust to failure. However, the functions used in this course are available in both packages, and many students have successfully completed the course using either.

Installing MATLAB

MathWorks is providing you access to MATLAB for use in your coursework.

When planning your activity, please note that access is valid for the length of the course (12 weeks).

Step 1: Enter your email to create a MathWorks account if you do not have one.

Step 2: Use this link again to download and install. You may need to log-in to your MathWorks account that you created in Step 1. After starting the installer, accept all defaults and log-in to your MathWorks account when prompted.

For additional resources, including an introduction to the MATLAB interface, please see "More Octave/MATLAB Resources."

Installing Octave on Mac OS X (10.10 Yosemite and 10.9 Mavericks)

Mac OS X has a feature called Gatekeeper that may only let you install applications from the Mac App Store. You may need to configure it to allow the Octave installer. Visit your System Preferences, click Security & Privacy, and check the setting to allow apps downloaded from Anywhere. You may need to enter your password to unlock the settings page.

2. Download the Octave 3.8.0 installer. The file is large so this may take some time.

3. Open the downloaded image, probably named GNU_Octave_3.8.0-6.dmg on your computer, and then open Octave-3.8.0-6.mpkg inside.

4. Follow the installer’s instructions. You may need to enter the administrator password for your computer.

5. After the installer completes, Octave should be installed on your computer. You can find Octave-cli in your Mac’s Applications, which is a text interface for Octave that you can use to complete Machine Learning’s programming assignments.

Octave also includes an experimental graphical interface which is called Octave-gui, also in your Mac’s Applications, but we recommend using Octave-cli because it’s more stable.

Note: If you use a package manager (like MacPorts or Homebrew), we recommend you follow the package manager installation instructions.

"Warning: Do not install Octave 4.0.0"; checkout the "Resources" menu's section of "Installation Issues".

More Octave/MATLAB resources

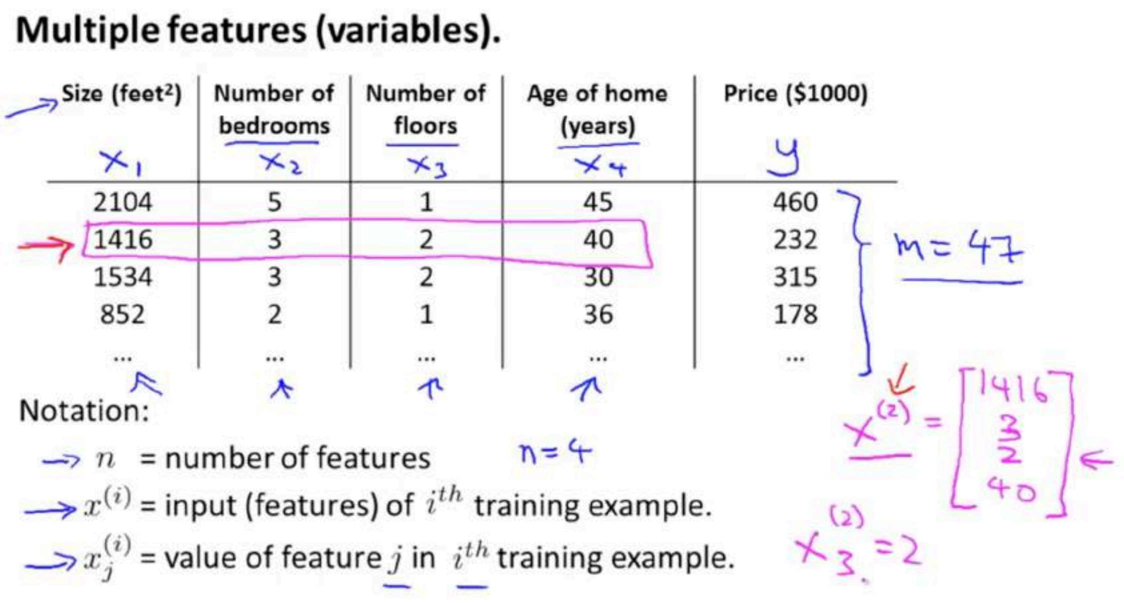
https://www.coursera.org/learn/machine-learning/supplement/Mlf3e/more-octave-matlab-resources

Multivariate Linear Regression

Multiple features (Video)

In this video we will start to talk about a new version of linear regression that's more powerful. One that works with multiple variables or with multiple features. Here's what I mean. In the original version of linear regression that we developed, we have a single feature x, the size of the house, and we wanted to use that to predict why the price of the house and this was our form of our hypothesis. But now imagine, what if we had not only the size of the house as a feature or as a variable of which to try to predict the price, but that we also knew the number of bedrooms, the number of house and the age of the home and years. It seems like this would give us a lot more information with which to predict the price. To introduce a little bit of notation, we sort of started to talk about this earlier, I'm going to use the variables X subscript 1 X subscript 2 and so on to denote my, in this case, four features and I'm going to continue to use

Y to denote the variable, the output variable price that we're trying to predict. Let's introduce a little bit more notation. Now that we have four features I'm going to use lowercase "n" to denote the number of features. So in this example we have n4 because we have, you know, one, two, three, four features. And "n" is different from our earlier notation where we were using "n" to denote the number of examples. So if you have 47 rows "M" is the number of rows on this table or the number of training examples. So I'm also going to use X superscript "I" to denote the input features of the "I" training example. As a concrete example let say X2 is going to be a vector of the features for my second training example. And so X2 here is going to be a vector 1416, 3, 2, 40 since those are my four features that I have to try to predict the price of the second house. So, in this notation, the superscript 2 here. That's an index into my training set. This is not X to the power of 2. Instead, this is, you know, an index that says look at the second row of this table. This refers to my second training example. With this notation X2 is a four dimensional vector. In fact, more generally, this is an in-dimensional feature back there. With this notation, X2 is now a vector and so, I'm going to use also Xi subscript J to denote the value of the J, of feature number J and the training example. So concretely X2 subscript 3, will refer to feature number three in the x factor which is equal to 2, right? That was a 3 over there, just fix my handwriting. So x2 subscript 3 is going to be equal to 2.



Multiple features (variables).

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

$m = 47$

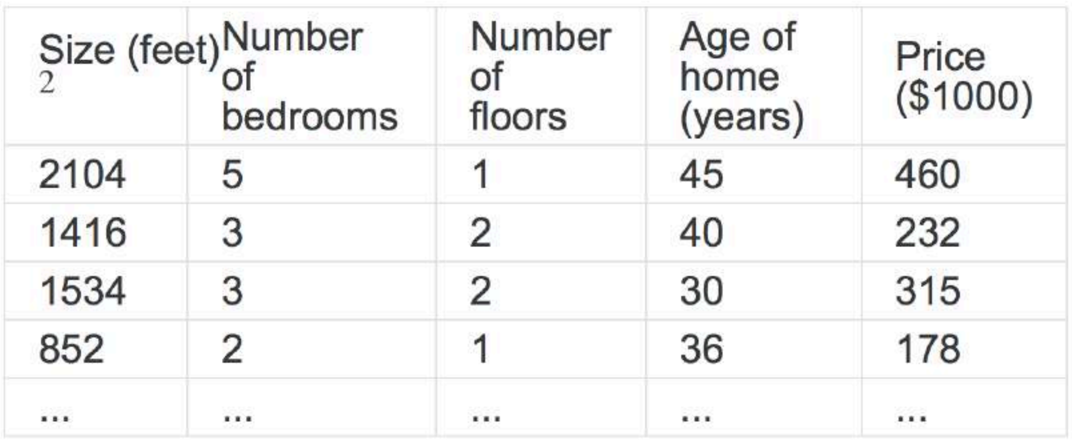
Notation:

- n = number of features $n = 4$
- $x^{(i)}$ = input (features) of i^{th} training example.
- $x_j^{(i)}$ = value of feature j in i^{th} training example.

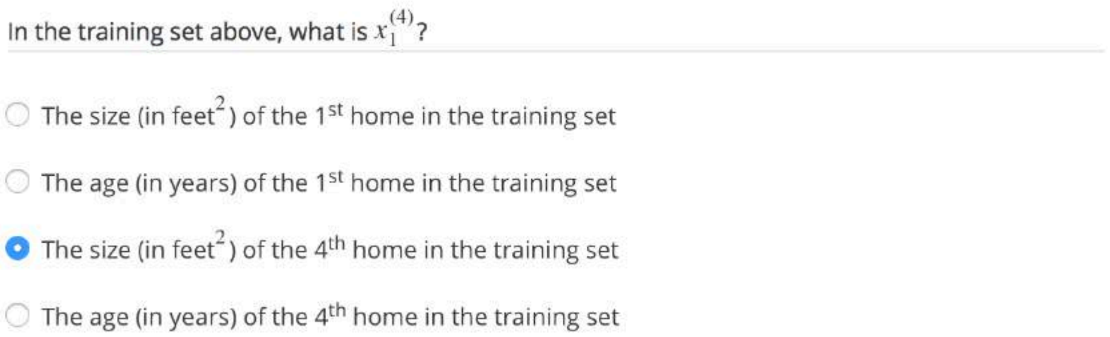
$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$

$$x_3^{(2)} = 2$$

Question



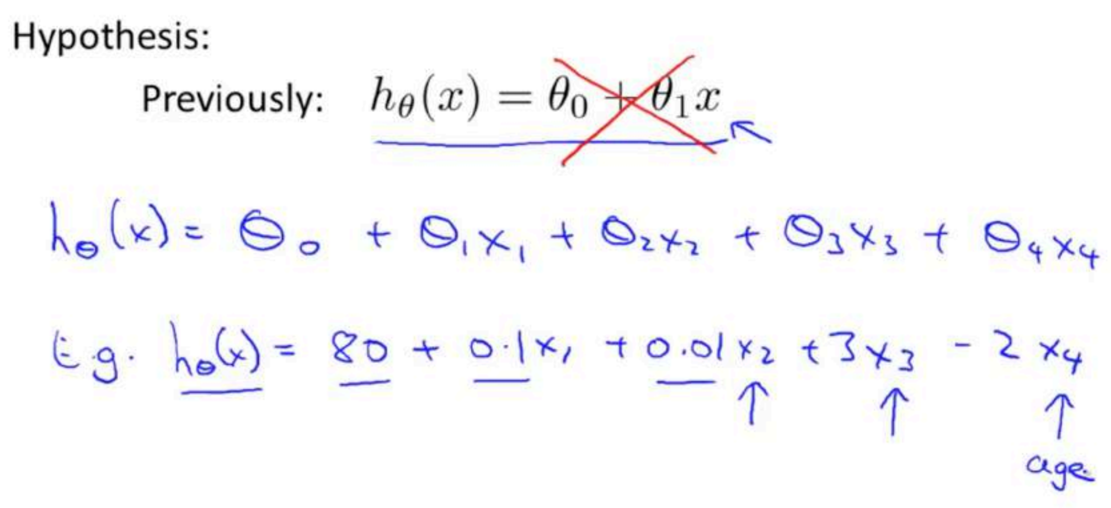
Size (feet) 2	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...



In the training set above, what is $x_1^{(4)}$?

- The size (in feet^2) of the 1st home in the training set
- The age (in years) of the 1st home in the training set
- [✓ The size (in feet^2) of the 4th home in the training set]
- The age (in years) of the 4th home in the training set

Now that we have multiple features, let's talk about what the form of our hypothesis should be. Previously this was the form of our hypothesis, where x was our single feature, but now that we have multiple features, we aren't going to use the simple representation any more. Instead, a form of the hypothesis in linear regression is going to be this, can be theta 0 plus theta 1 x1 plus theta 2 x2 plus theta 3 x3 plus theta 4 X4. And if we have N features then rather than summing up over our four features, we would have a sum over our N features. Concretely for a particular setting of our parameters we may have H of X 80 + 0.1 X1 + 0.01x2 + 3x3 - 2x4. This would be one example of a hypothesis and you remember a hypothesis is trying to predict the price of the house in thousands of dollars, just saying that, you know, the base price of a house is maybe 80,000 plus another open 1, so that's an extra, what, hundred dollars per square feet, yeah, plus the price goes up a little bit for each additional floor that the house has. X two is the number of floors, and it goes up further for each additional bedroom the house has, because X three was the number of bedrooms, and the price goes down a little bit with each additional age of the house. With each additional year of the age of the house.



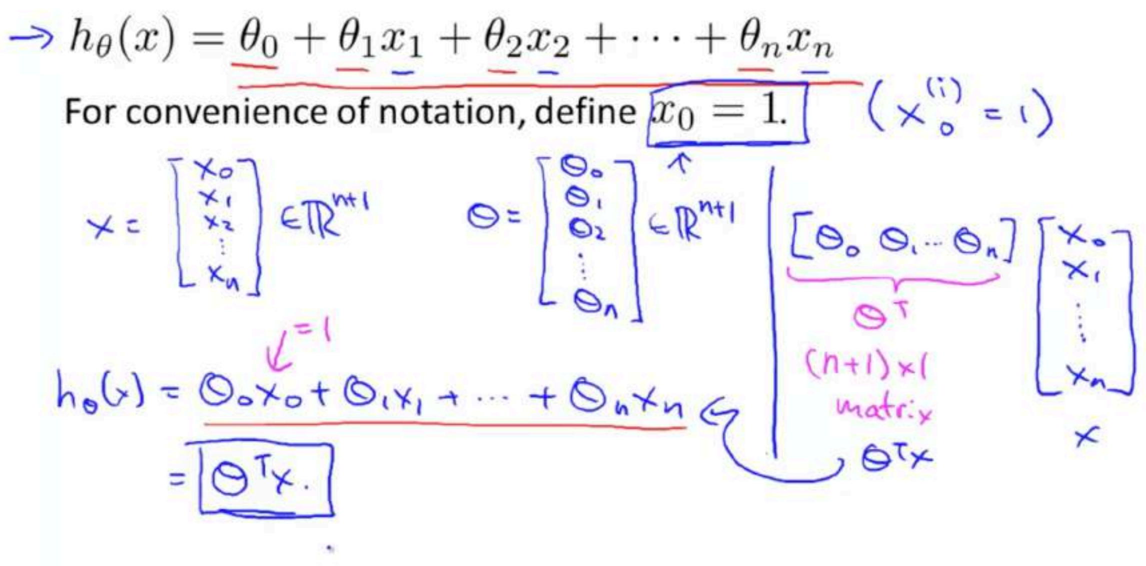
Hypothesis:

Previously: $\underline{h_\theta(x) = \theta_0 \cancel{+ \theta_1 x}}$

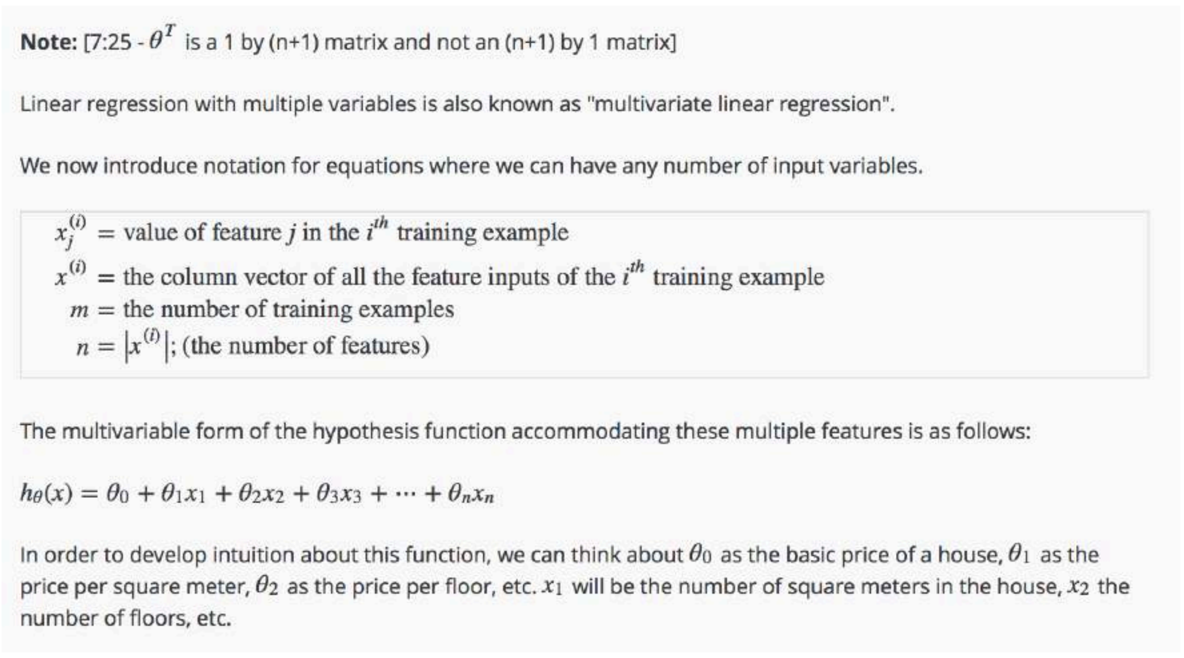
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$\text{E.g. } \underline{h_\theta(x)} = \underline{80} + \underline{0.1} x_1 + \underline{0.01} \underset{\uparrow}{x_2} + 3 \underset{\uparrow}{x_3} - 2 \underset{\underset{\text{age}}{\uparrow}}{x_4}$$

Here's the form of a hypothesis rewritten on the slide. And what I'm gonna do is introduce a little bit of notation to simplify this equation. For convenience of notation, let me define x subscript 0 to be equals one. Concretely, this means that for every example i I have a feature vector X superscript I and X superscript I subscript 0 is going to be equal to 1. You can think of this as defining an additional zero feature. So whereas previously I had n features because x1, x2 through xn, I'm now defining an additional sort of zero feature vector that always takes on the value of one. So now my feature vector X becomes this N+1 dimensional vector that is zero index. So this is now a n+1 dimensional feature vector, but I'm gonna index it from 0 and I'm also going to think of my parameters as a vector. So, our parameters here, right that would be our theta zero, theta one, theta two, and so on all the way up to theta n, we're going to gather them up into a parameter vector written theta 0, theta 1, theta 2, and so on, down to theta n. This is another zero index vector. It's of index signed from zero. That is another n plus 1 dimensional vector. So, my hypothesis cannot be written theta 0x0 plus theta 1x1+ up to theta n Xn. And this equation is the same as this on top because, you know, eight zero is equal to one. Underneath and I now take this form of the hypothesis and write this as either transpose x, depending on how familiar you are with inner products of vectors if you write what theta transfers x is what theta transfer and this is theta zero, theta one, up to theta N. So this thing here is theta transpose and this is actually a N plus one by one matrix. [It should be a 1 by (n+1) matrix] It's also called a row vector and you take that and multiply it with the vector X which is X zero, X one, and so on, down to X n. And so, the inner product that is theta transpose X is just equal to this. This gives us a convenient way to write the form of the hypothesis as just the inner product between our parameter vector theta and our theta vector X. And it is this little bit of notation, this little excerpt of the notation convention that let us write this in this compact form. So that's the form of a hypothesis when we have multiple features. And, just to give this another name, this is also called multivariate linear regression. And the term multivariable that's just maybe a fancy term for saying we have multiple features, or multivariables with which to try to predict the value Y.


$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
For convenience of notation, define $x_0 = 1$.
$(x_0^{(i)} = 1)$
$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$
$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$
$[\theta_0\ \theta_1 \cdots \theta_n]$
θ^T
$(n+1)\times 1$ matrix
x
$=1$
$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$
$= \theta^T x.$
$\theta^T x$

Multiple Features (Transcript)



Note: [7:25 - θ^T is a 1 by (n+1) matrix and not an (n+1) by 1 matrix]

Linear regression with multiple variables is also known as "multivariate linear regression".

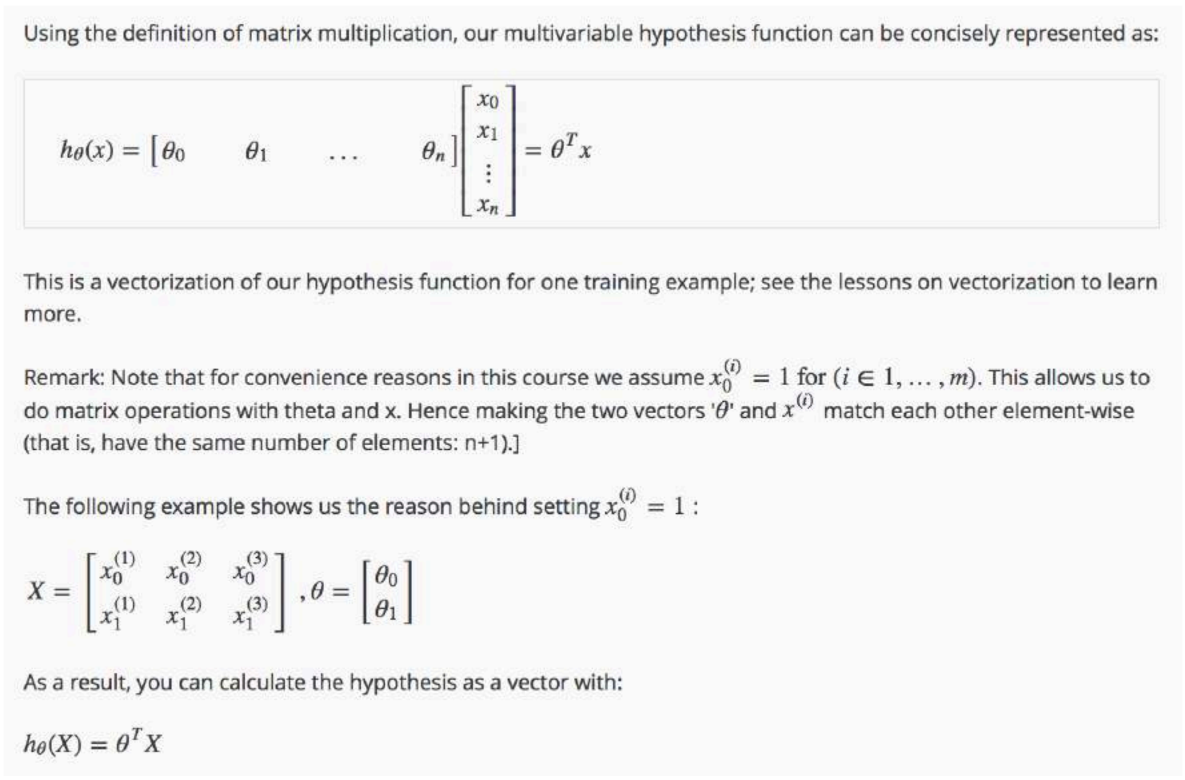
We now introduce notation for equations where we can have any number of input variables.

$$\begin{aligned} x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\ x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\ m &= \text{the number of training examples} \\ n &= \left|x^{(i)}\right|; \text{(the number of features)} \end{aligned}$$

The multivariable form of the hypothesis function accommodating these multiple features is as follows:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

In order to develop intuition about this function, we can think about θ_0 as the basic price of a house, θ_1 as the price per square meter, θ_2 as the price per floor, etc. x_1 will be the number of square meters in the house, x_2 the number of floors, etc.



Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

$$h_\theta(x) = \begin{bmatrix} \theta_0 & \theta_1 & \dots & \theta_n \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

This is a vectorization of our hypothesis function for one training example; see the lessons on vectorization to learn more.

Remark: Note that for convenience reasons in this course we assume $x_0^{(i)} = 1$ for $(i \in 1, \dots, m)$. This allows us to do matrix operations with theta and x. Hence making the two vectors 'θ' and $x^{(i)}$ match each other element-wise (that is, have the same number of elements: n+1).]

The following example shows us the reason behind setting $x_0^{(i)} = 1$:

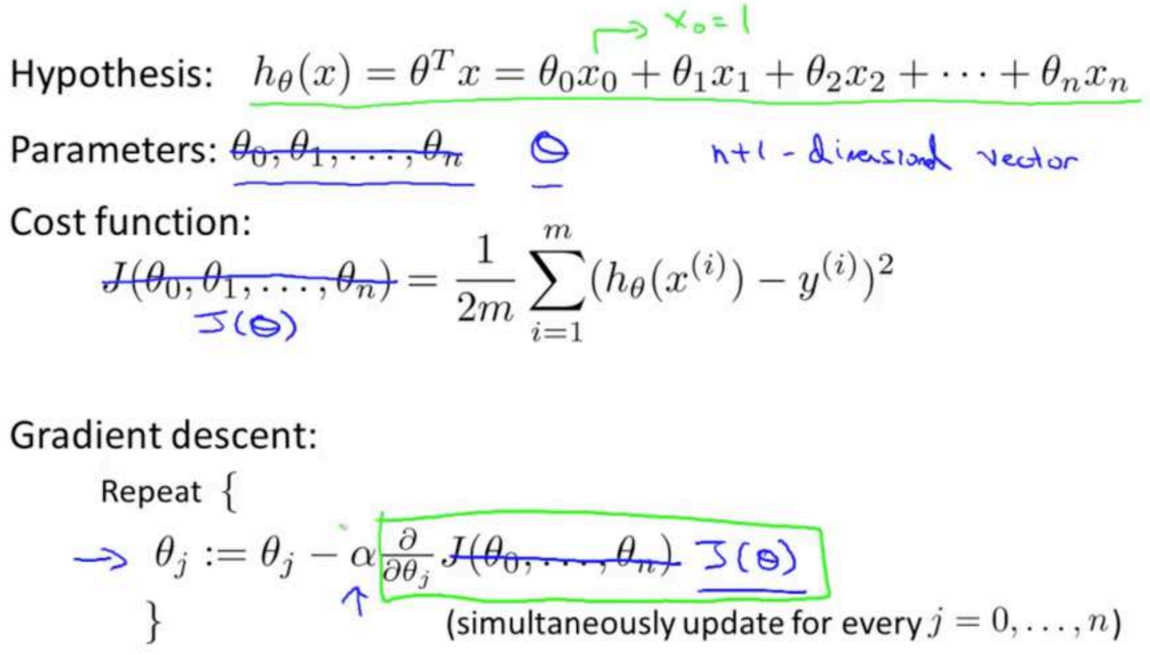
$$X = \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & x_0^{(3)} \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

As a result, you can calculate the hypothesis as a vector with:

$h_\theta(X) = \theta^T X$

Gradient Descent for Multiple variables (Video)

In the previous video, we talked about the form of the hypothesis for linear regression with multiple features or with multiple variables. In this video, let's talk about how to fit the parameters of that hypothesis. In particular let's talk about how to use gradient descent for linear regression with multiple features. To quickly summarise our notation, this is our formal hypothesis in multivariable linear regression where we've adopted the convention that x0=1. The parameters of this model are theta0 through theta n, but instead of thinking of this as n separate parameters, which is valid, I'm instead going to think of the parameters as theta where theta here is a n+1-dimensional vector. So I'm just going to think of the parameters of this model as itself being a vector. Our cost function is J of theta0 through theta n which is given by this usual sum of square of error term. But again instead of thinking of J as a function of these n+1 numbers, I'm going to more commonly write J as just a function of the parameter vector theta so that theta here is a vector. Here's what gradient descent looks like. We're going to repeatedly update each parameter theta j according to theta j minus alpha times this derivative term. And once again we just write this as J of theta, so theta j is updated as theta j minus the learning rate alpha times the derivative, a partial derivative of the cost function with respect to the parameter theta j. Let's see what this looks like when we implement gradient descent and, in particular, let's go see what that partial derivative term looks like.



Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 \overset{x_0 = 1}{x_0} + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: ~~$\theta_0, \theta_1, \ldots, \theta_n$~~ $\underline{\Theta}$ n+1 - dimensional vector

Cost function:

$$\text{~~}J(\theta_0, \theta_1, \ldots, \theta_n)\text{~~} = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$J(\Theta)$

Gradient descent:

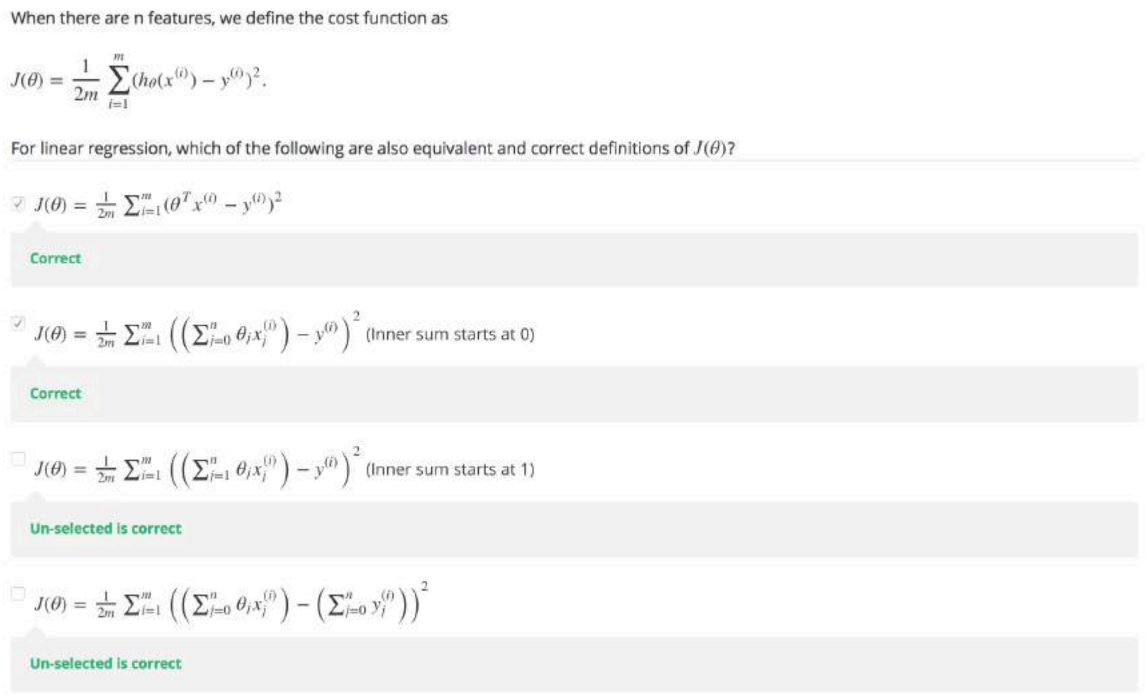
Repeat {

$\rightarrow \theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial \theta_j} \cancel{J(\theta_0, \ldots, \theta_n)}\ \underline{J(\Theta)}}$

}

(simultaneously update for every $j = 0, \ldots, n$)

Question



When there are n features, we define the cost function as

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

For linear regression, which of the following are also equivalent and correct definitions of $J(\theta)$?

✓ $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$

Correct

✓ $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=0}^{n}\theta_j x_j^{(i)}\right) - y^{(i)}\right)^2$ (Inner sum starts at 0)

Correct

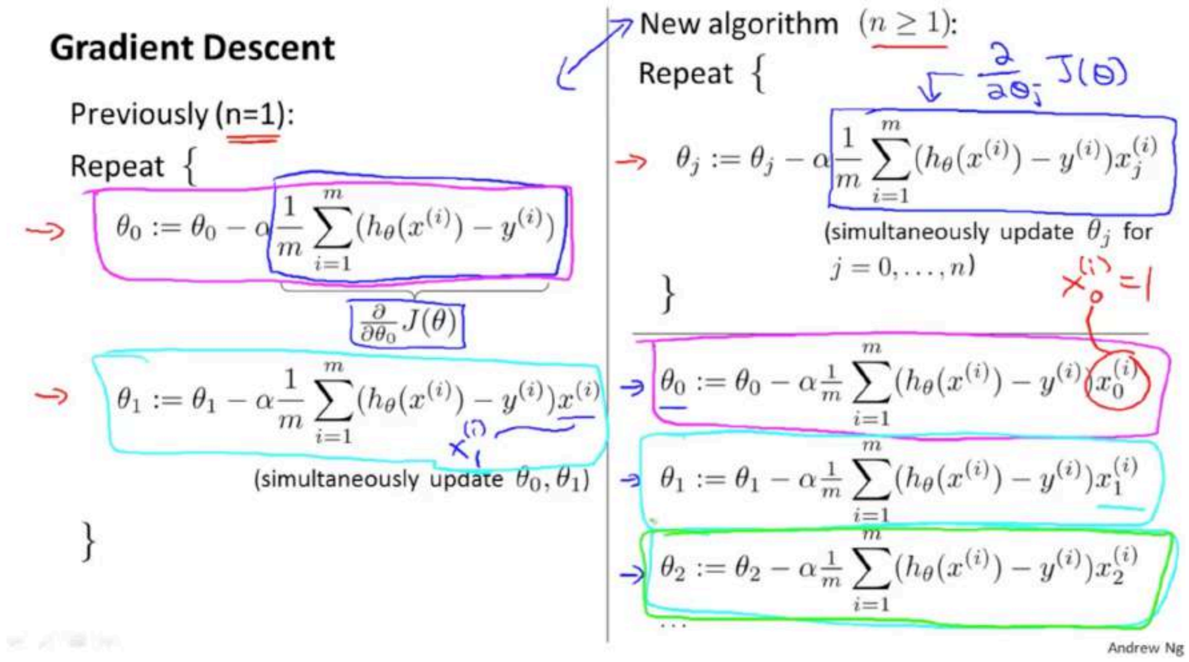
☐ $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=1}^{n}\theta_j x_j^{(i)}\right) - y^{(i)}\right)^2$ (Inner sum starts at 1)

Un-selected is correct

☐ $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=0}^{n}\theta_j x_j^{(i)}\right) - \left(\sum_{j=0}^{n} y_j^{(i)}\right)\right)^2$

Un-selected is correct

Here's what gradient descent looks like. We're going to repeatedly update each parameter theta j according to theta j minus alpha times this derivative term. And once again we just write this as J of theta, so theta j is updated as theta j minus the learning rate alpha times the derivative, a partial derivative of the cost function with respect to the parameter theta j. Let's see what this looks like when we implement gradient descent and, in particular, let's go see what that partial derivative term looks like. Here's what we have for gradient descent for the case of when we had N=1 feature. We had two separate update rules for the parameters theta0 and theta1, and hopefully these look familiar to you. And this term here was of course the partial derivative of the cost function with respect to the parameter of theta0, and similarly we had a different update rule for the parameter theta1. There's one little difference which is that when we previously had only one feature, we would call that feature x(i) but now in our new notation we would of course call this x(i)<u>1 to denote our one feature.</u> So that was for when we had only one feature. Let's look at the new algorithm for we have more than one feature, where the number of features n may be much larger than one. We get this update rule for gradient descent and, maybe for those of you that know calculus, if you take the definition of the cost function and take the partial derivative of the cost function J with respect to the parameter theta j, you'll find that that partial derivative is exactly that term that I've drawn the blue box around. And if you implement this you will get a working implementation of gradient descent for multivariate linear regression. The last thing I want to do on this slide is give you a sense of why these new and old algorithms are sort of the same thing or why they're both similar algorithms or why they're both gradient descent algorithms. Let's consider a case where we have two features or maybe more than two features, so we have three update rules for the parameters theta0, theta1, theta2 and maybe other values of theta as well. If you look at the update rule for theta0, what you find is that this update rule here is the same as the update rule that we had previously for the case of n = 1. And the reason that they are equivalent is, of course, because in our notational convention we had this x(i)<u>0 = 1 convention, which is</u> why these two term that I've drawn the magenta boxes around are equivalent. Similarly, if you look the update rule for theta1, you find that this term here is equivalent to the term we previously had, or the equation or the update rule we previously had for theta1, where of course we're just using this new notation x(i)<u>1 to denote</u> our first feature, and now that we have more than one feature we can have similar update rules for the other parameters like theta2 and so on. There's a lot going on on this slide so I definitely encourage you if you need to to pause the video and look at all the math on this slide slowly to make sure you understand everything that's going on here. But if you implement the algorithm written up here then you have a working implementation of linear regression with multiple features.



Gradient Descent

Previously (n=1):

Repeat {

$$\boxed{\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)}_{\frac{\partial}{\partial\theta_0}J(\theta)}}$$

$$\boxed{\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\underset{x_1^{(i)}}{x^{(i)}}}$$

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

$$\theta_j := \theta_j - \alpha \underset{}{\overset{\frac{\partial}{\partial\theta_j}J(\theta)}{\boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}}}}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

$x_0^{(i)} = 1$

$$\boxed{\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)}}$$

$$\boxed{\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_1^{(i)}}$$

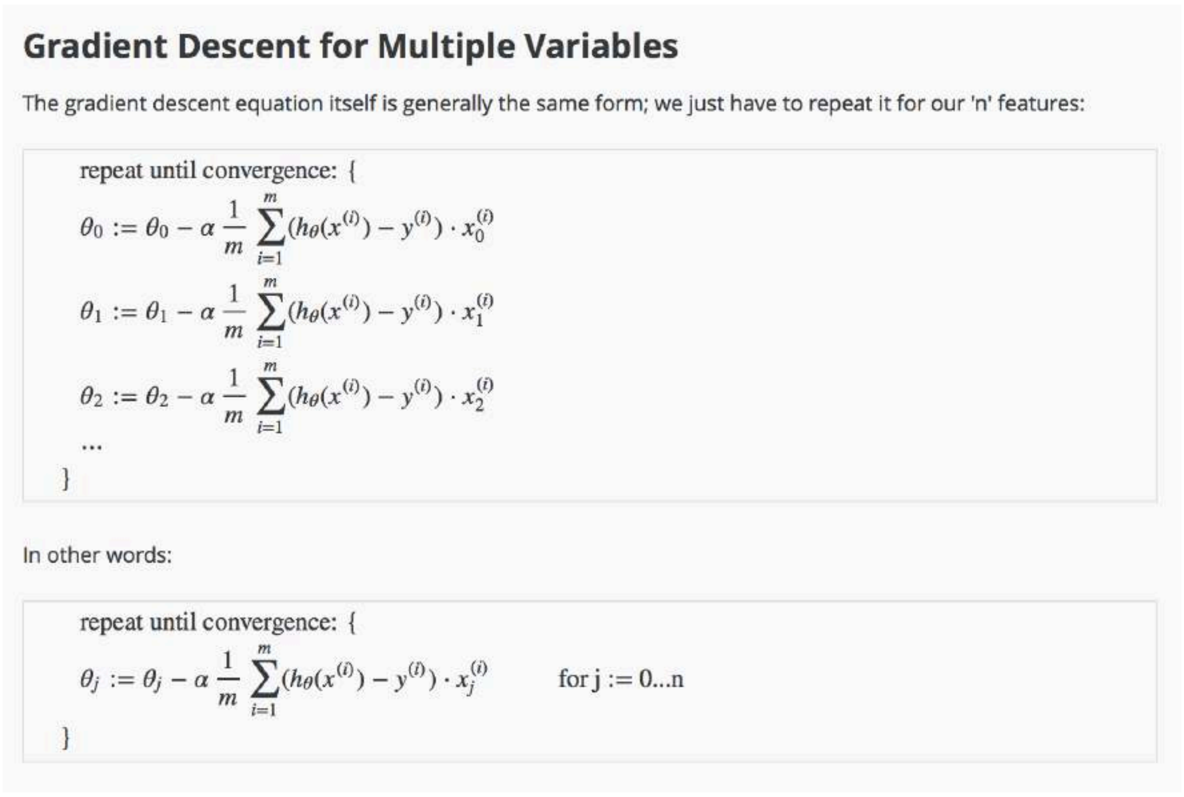
$$\boxed{\theta_2 := \theta_2 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_2^{(i)}}$$

...

Andrew Ng

Gradient Descent for Multiple

variables (Transcript)



Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

...

}

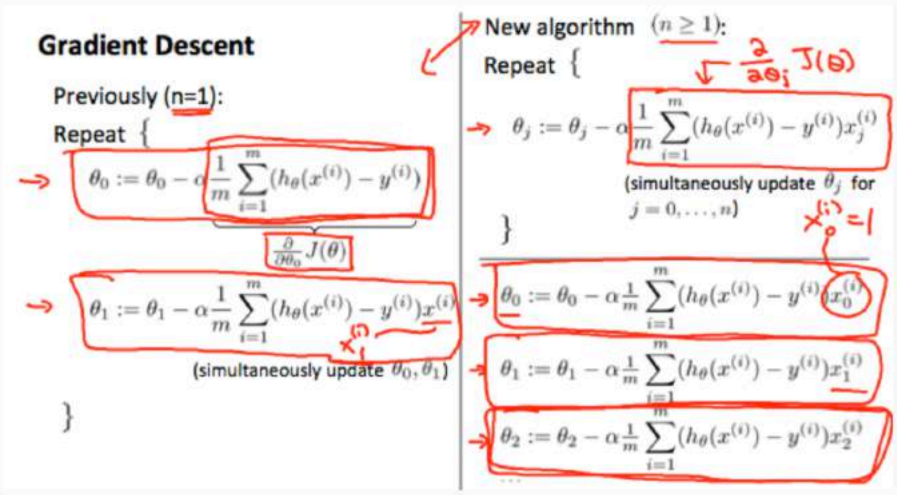
In other words:

repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n}$$

}

The following image compares gradient descent with one variable to gradient descent with multiple variables:



Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \underbrace{\boxed{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}}_{\boxed{\frac{\partial}{\partial \theta_0} J(\theta)}}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \underline{x^{(i)}}$$

$x^{(i)}$

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

$\frac{\partial}{\partial \theta_j} J(\theta)$

$$\theta_j := \theta_j - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

}

$x_0^{(i)} = 1$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

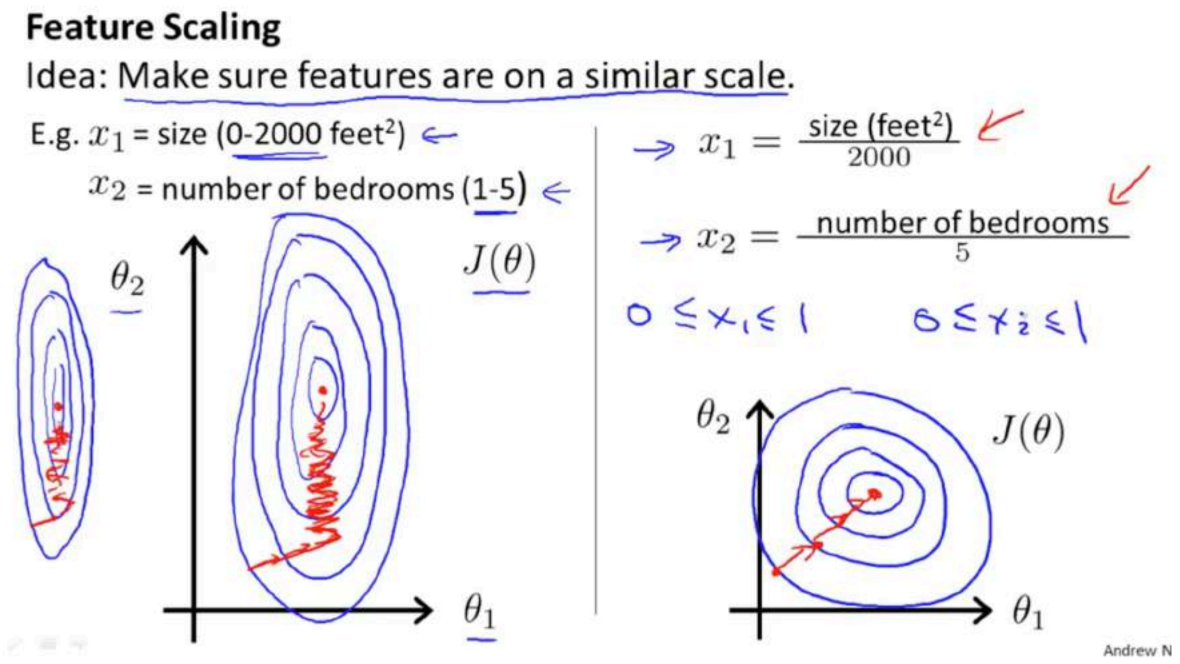
$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...

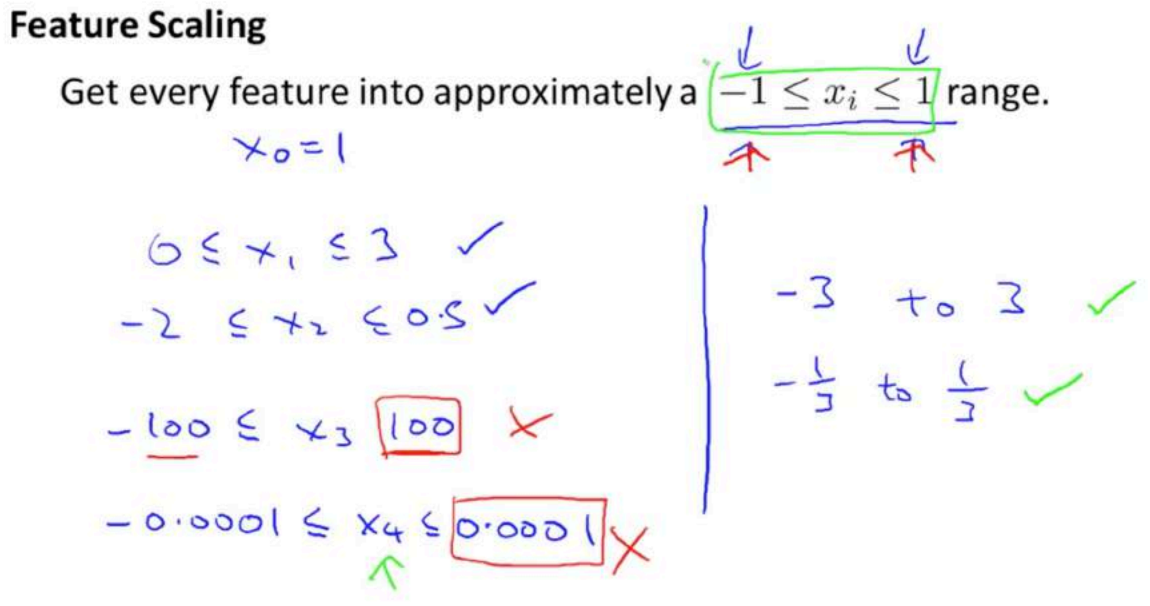
Gradient Descent in Practice 1 -

Feature scaling (Video)

In this video and in the video after this one, I wanna tell you about some of the practical tricks for making gradient descent work well. In this video, I want to tell you about an idea called feature skill. Here's the idea. If you have a problem where you have multiple features, if you make sure that the features are on a similar scale, by which I mean make sure that the different features take on similar ranges of values, then gradient descents can converge more quickly. Concretely let's say you have a problem with two features where X1 is the size of house and takes on values between say zero to two thousand and two is the number of bedrooms, and maybe that takes on values between one and five. If you plot the contours of the cos function J of theta, then the contours may look like this, where, let's see, J of theta is a function of parameters theta zero, theta one and theta two. I'm going to ignore theta zero, so let's about theta 0 and pretend as a function of only theta 1 and theta 2, but if x1 can take on them, you know, much larger range of values and x2 It turns out that the contours of the cause function J of theta can take on this very very skewed elliptical shape, except that with the so 2000 to 5 ratio, it can be even more secure. So, this is very, very tall and skinny ellipses, or these very tall skinny ovals, can form the contours of the cause function J of theta. And if you run gradient descents on this cos-function, your gradients may end up taking a long time and can oscillate back and forth and take a long time before it can finally find its way to the global minimum. In fact, you can imagine if these contours are exaggerated even more when you draw incredibly skinny, tall skinny contours, and it can be even more extreme than, then, gradient descent just have a much harder time taking it's way, meandering around, it can take a long time to find this way to the global minimum. In these settings, a useful thing to do is to scale the features. Concretely if you instead define the feature X one to be the size of the house divided by two thousand, and define X two to be maybe the number of bedrooms divided by five, then the count well as of the cost function J can become much more, much less skewed so the contours may look more like circles. And if you run gradient descent on a cost function like this, then gradient descent, you can show mathematically, you can find a much more direct path to the global minimum rather than taking a much more convoluted path where you're sort of trying to follow a much more complicated trajectory to get to the global minimum. So, by scaling the features so that there are, the consumer ranges of values. In this example, we end up with both features, X one and X two, between zero and one. You can wind up with an implementation of gradient descent. They can convert much faster.


Feature Scaling
Idea: Make sure features are on a similar scale.
E.g. x_1 = size (0-2000 feet^2)
x_2 = number of bedrooms (1-5)
θ_2
$J(\theta)$
θ_1
$x_1 = \frac{\text{size (feet}^2)}{2000}$
$x_2 = \frac{\text{number of bedrooms}}{5}$
$0 \leq x_1 \leq 1$
$0 \leq x_2 \leq 1$
θ_2
$J(\theta)$
θ_1
Andrew N

More generally, when we're performing feature scaling, what we often want to do is get every feature into approximately a -1 to +1 range and concretely, your feature x0 is always equal to 1. So, that's already in that range, but you may end up dividing other features by different numbers to get them to this range. The numbers -1 and +1 aren't too important. So, if you have a feature, x1 that winds up being between zero and three, that's not a problem. If you end up having a different feature that winds being between -2 and + 0.5, again, this is close enough to minus one and plus one that, you know, that's fine, and that's fine. It's only if you have a different feature, say X 3 that is between, that ranges from -100 to +100 , then, this is a very different values than minus 1 and plus 1. So, this might be a less well-skilled feature and similarly, if your features take on a very, very small range of values so if X 4 takes on values between minus 0.0001 and positive 0.0001, then again this takes on a much smaller range of values than the minus one to plus one range. And again I would consider this feature poorly scaled. So you want the range of values, you know, can be bigger than plus or smaller than plus one, but just not much bigger, like plus 100 here, or too much smaller like 0.00 one over there. Different people have different rules of thumb. But the one that I use is that if a feature takes on the range of values from say minus three the plus 3 how you should think that should be just fine, but maybe it takes on much larger values than plus 3 or minus 3 unless not to worry and if it takes on values from say minus one-third to one-third. You know, I think that's fine too or 0 to one-third or minus one-third to 0. I guess that's typical range of value sector 0 okay. But it will take on a much tinier range of values like x4 here than gain on mine not to worry. So, the take-home message is don't worry if your features are not exactly on the same scale or exactly in the same range of values. But so long as they're all close enough to this gradient descent it should work okay.



Feature Scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

$x_0 = 1$

$0 \leq x_1 \leq 3$ ✓

$-2 \leq x_2 \leq 0.5$ ✓

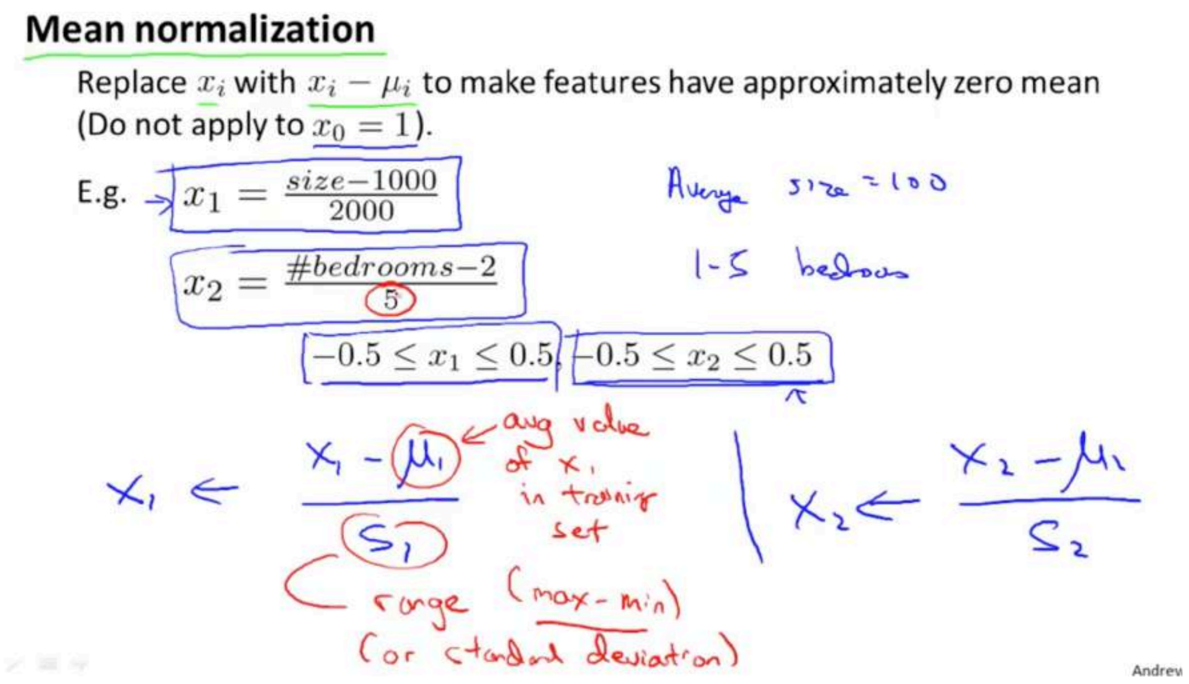
$-100 \leq x_3 \boxed{100}$ ✗

$-0.0001 \leq x_4 \leq \boxed{0.0001}$ ✗

-3 to 3 ✓

$-\frac{1}{3}$ to $\frac{1}{3}$ ✓

In addition to dividing by so that the maximum value when performing feature scaling sometimes people will also do what's called mean normalization. And what I mean by that is that you want to take a feature Xi and replace it with Xi minus new i to make your features have approximately 0 mean. And obviously we want to apply this to the future x zero, because the future x zero is always equal to one, so it cannot have an average value of zero. But it concretely for other features if the range of sizes of the house takes on values between 0 to 2000 and if you know, the average size of a house is equal to 1000 then you might use this formula. Size, set the feature X1 to the size minus the average value divided by 2000 and similarly, on average if your houses have one to five bedrooms and if on average a house has two bedrooms then you might use this formula to mean normalize your second feature x2. In both of these cases, you therefore wind up with features x1 and x2. They can take on values roughly between minus .5 and positive .5. Exactly not true - X2 can actually be slightly larger than .5 but, close enough. And the more general rule is that you might take a feature X1 and replace it with X1 minus mu1 over S1 where to define these terms mu1 is the average value of x1 in the training sets and S1 is the range of values of that feature and by range, I mean let's say the maximum value minus the minimum value or for those of you that understand the deviation of the variable is setting S1 to be the standard deviation of the variable would be fine, too. But taking, you know, this max minus min would be fine. And similarly for the second feature, x2, you replace x2 with this sort of subtract the mean of the feature and divide it by the range of values meaning the max minus min. And this sort of formula will get your features, you know, maybe not exactly, but maybe roughly into these sorts of ranges, and by the way, for those of you that are being super careful technically if we're taking the range as max minus min this five here will actually become a four. So if max is 5 minus 1 then the range of their own values is actually equal to 4, but all of these are approximate and any value that gets the features into anything close to these sorts of ranges will do fine. And the feature scaling doesn't have to be too exact, in order to get gradient descent to run quite a lot faster.


Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

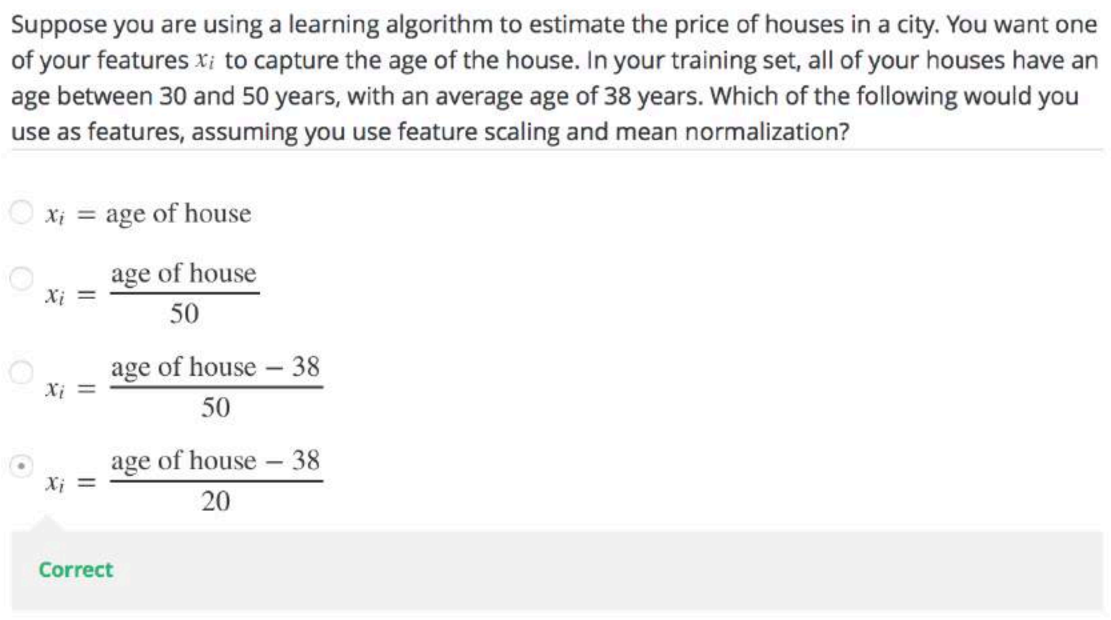
E.g. $x_1 = \frac{size-1000}{2000}$

$x_2 = \frac{\#bedrooms-2}{5}$

$-0.5 \le x_1 \le 0.5$, $-0.5 \le x_2 \le 0.5$

Average size = 100
1-5 bedrooms
$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$
avg value of x_1 in training set
range (max - min)
(or standard deviation)
$x_2 \leftarrow \frac{x_2 - \mu_1}{s_2}$
Andrew

Question

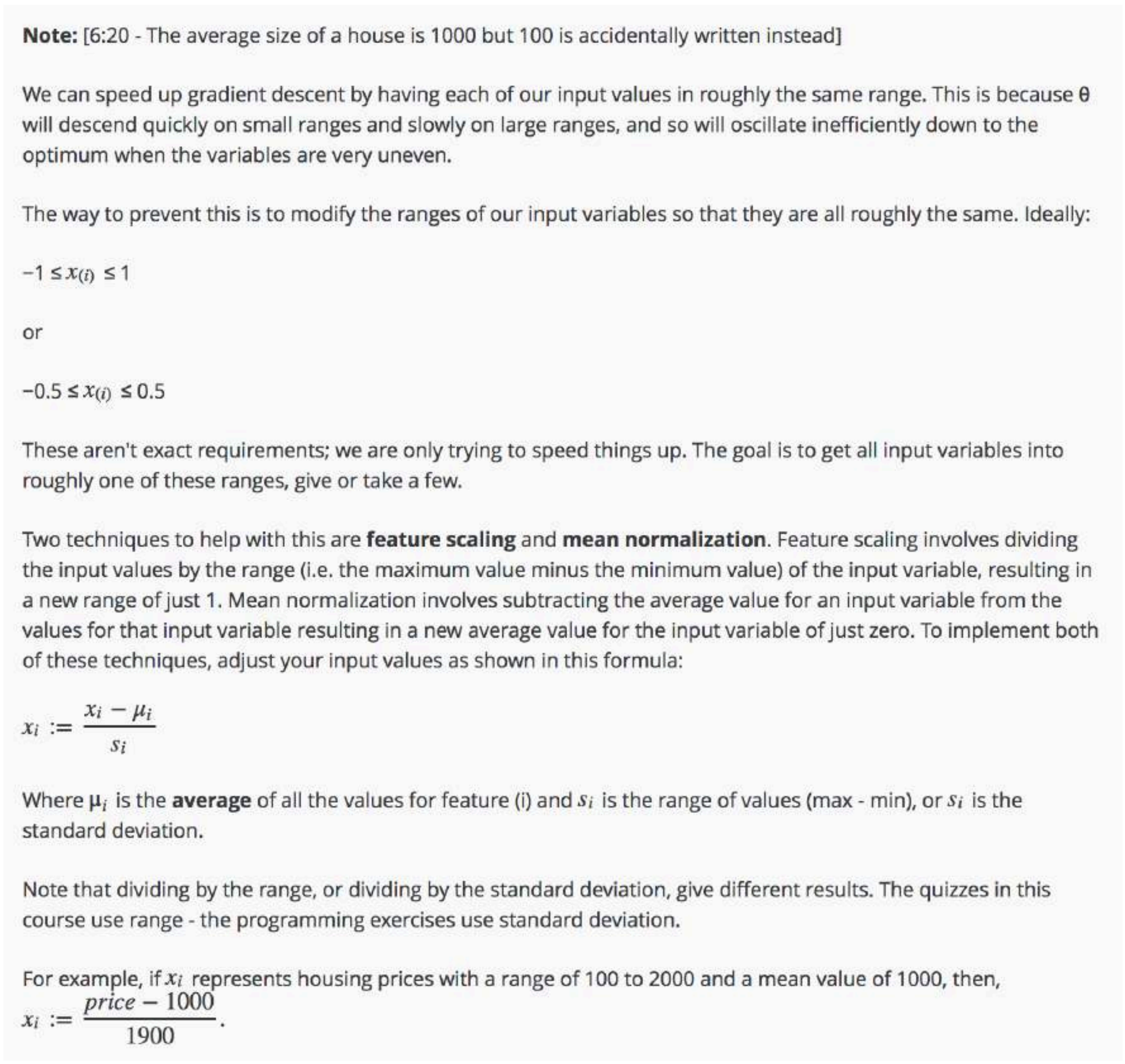

Suppose you are using a learning algorithm to estimate the price of houses in a city. You want one of your features x_i to capture the age of the house. In your training set, all of your houses have an age between 30 and 50 years, with an average age of 38 years. Which of the following would you use as features, assuming you use feature scaling and mean normalization?

- ○ $x_i = \text{age of house}$
- ○ $x_i = \frac{\text{age of house}}{50}$
- ○ $x_i = \frac{\text{age of house} - 38}{50}$
- [✓ $x_i = \frac{\text{age of house} - 38}{20}$]

Correct

So, now you know about feature scaling and if you apply this simple trick, it and make gradient descent run much faster and converge in a lot fewer other iterations. That was feature scaling. In the next video, I'll tell you about another trick to make gradient descent work well in practice.

Gradient Descent in Practice 1 - Feature scaling (Transcript)



Note: [6:20 - The average size of a house is 1000 but 100 is accidentally written instead]

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

$-1 \leq x_{(i)} \leq 1$

or

$-0.5 \leq x_{(i)} \leq 0.5$

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

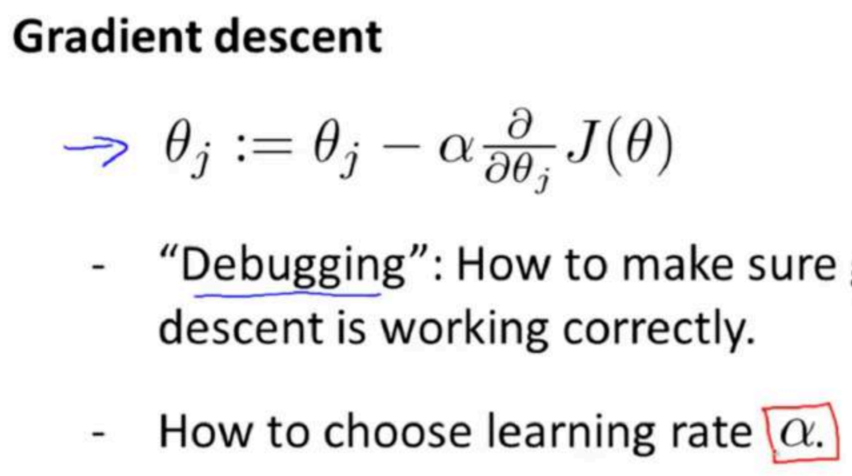
Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Note that dividing by the range, or dividing by the standard deviation, give different results. The quizzes in this course use range - the programming exercises use standard deviation.

For example, if x_i represents housing prices with a range of 100 to 2000 and a mean value of 1000, then, $x_i := \frac{price - 1000}{1900}$.

Gradient Descent in Practice 2 - Feature scaling (Video)

In this video, I want to give you more practical tips for getting gradient descent to work. The ideas in this video will center around the learning rate alpha. Concretely, here's the gradient descent update rule. And what I want to do in this video is tell you about what I think of as debugging, and some tips for making sure that gradient descent is working correctly. And second, I wanna tell you how to choose the learning rate alpha or at least how I go about choosing it.

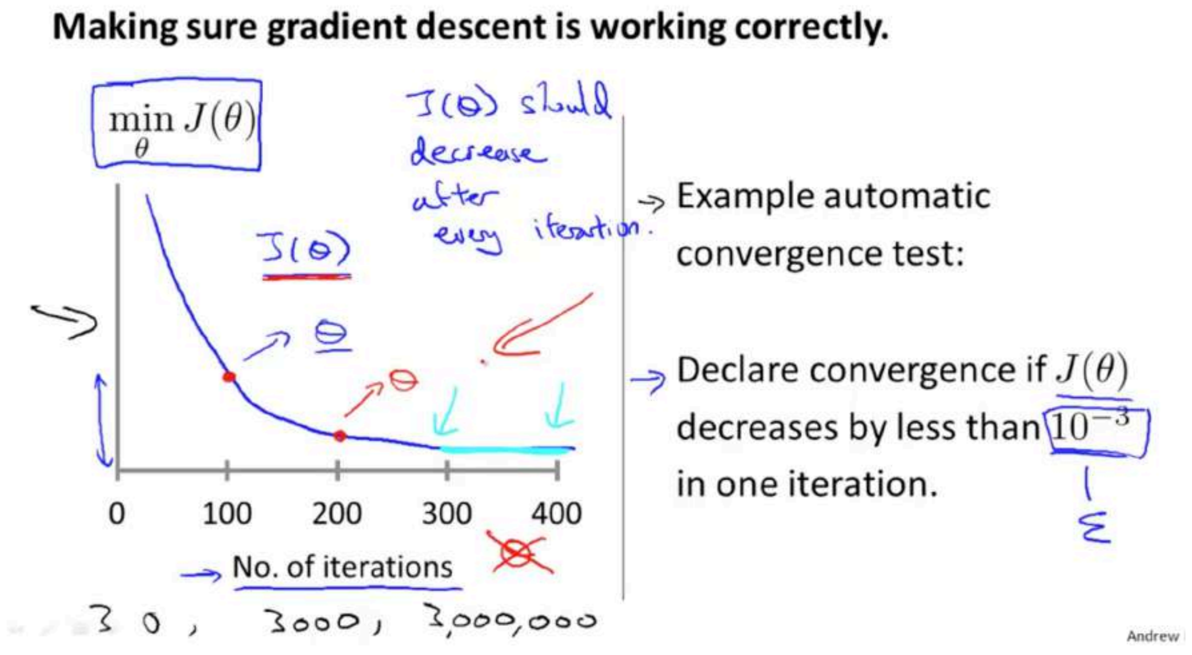


Gradient descent

$$\rightarrow \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- "Debugging": How to make sure gradient descent is working correctly.
- How to choose learning rate $\boxed{\alpha.}$

Here's something that I often do to make sure that gradient descent is working correctly. The job of gradient descent is to find the value of theta for you that hopefully minimizes the cost function J(theta). What I often do is therefore plot the cost function J(theta) as gradient descent runs. So the x axis here is a number of iterations of gradient descent and as gradient descent runs you hopefully get a plot that maybe looks like this. Notice that the x axis is number of iterations. Previously we where looking at plots of J(theta) where the x axis, where the horizontal axis, was the parameter vector theta but this is not what this is. Concretely, what this point is, is I'm going to run gradient descent for 100 iterations. And whatever value I get for theta after 100 iterations, I'm going to get some value of theta after 100 iterations. And I'm going to evaluate the cost function J(theta). For the value of theta I get after 100 iterations, and this vertical height is the value of J(theta). For the value of theta I got after 100 iterations of gradient descent. And this point here that corresponds to the value of J(theta) for the theta that I get after I've run gradient descent for 200 iterations. So what this plot is showing is, is it's showing the value of your cost function after each iteration of gradient decent. And if gradient is working properly then J(theta) should decrease after every iteration. And one useful thing that this sort of plot can tell you also is that if you look at the specific figure that I've drawn, it looks like by the time you've gotten out to maybe 300 iterations, between 300 and 400 iterations, in this segment it looks like J(theta) hasn't gone down much more. So by the time you get to 400 iterations, it looks like this curve has flattened out here. And so way out here 400 iterations, it looks like gradient descent has more or less converged because your cost function isn't going down much more. So looking at this figure can also help you judge whether or not gradient descent has converged. By the way, the number of iterations the gradient descent takes to converge for a physical application can vary a lot, so maybe for one application, gradient descent may converge after just thirty iterations. For a different application, gradient descent may take 3,000 iterations, for another learning algorithm, it may take 3 million iterations. It turns out to be very difficult to tell in advance how many iterations gradient descent needs to converge. And is usually by plotting this sort of plot, plotting the cost function as we increase in number in iterations, is usually by looking at these plots. But I try to tell if gradient descent has converged. It's also possible to come up with automatic convergence test, namely to have a algorithm try to tell you if gradient descent has converged. And here's maybe a pretty typical example of an automatic convergence test. And such a test may declare convergence if your cost function J(theta) decreases by less than some small value epsilon, some small value 10 to the minus 3 in one iteration. But I find that usually choosing what this threshold is is pretty difficult. And so in order to check your gradient descent's converge I actually tend to look at plots like these, like this figure on the left, rather than rely on an automatic convergence test. Looking at this sort of figure can also tell you, or give you an advance warning, if maybe gradient descent is not working correctly.

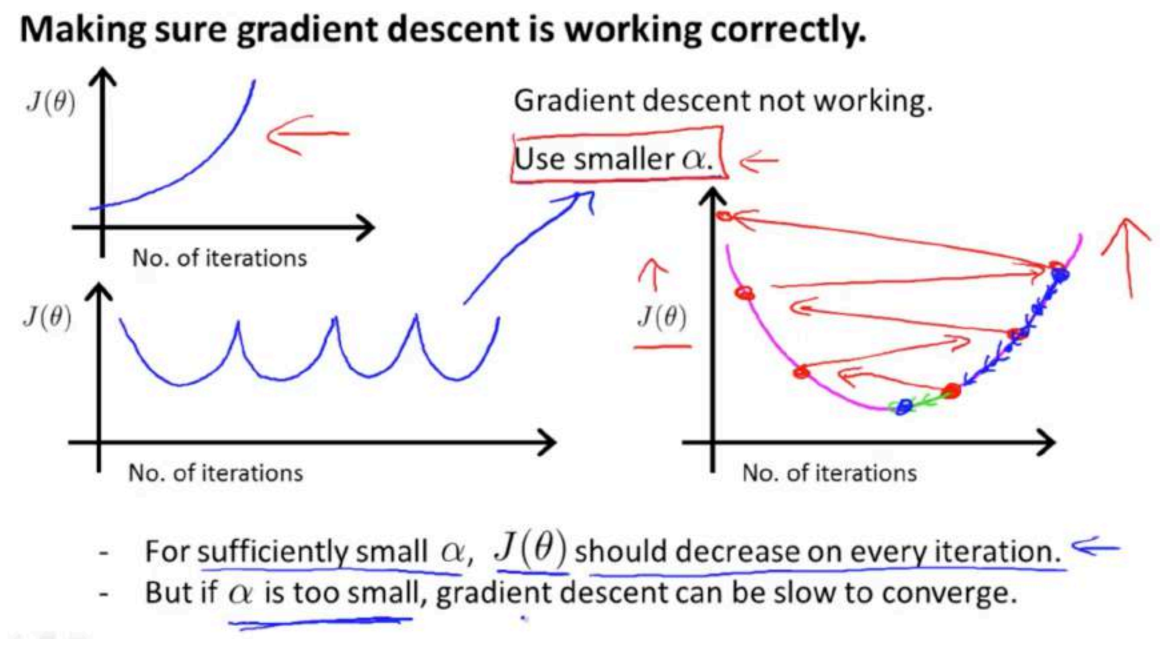


Making sure gradient descent is working correctly.

Example automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

Concretely, if you plot J(theta) as a function of the number of iterations. Then if you see a figure like this where J(theta) is actually increasing, then that gives you a clear sign that gradient descent is not working. And a theta like this usually means that you should be using learning rate alpha. If J(theta) is actually increasing, the most common cause for that is if you're trying to minimize a function, that maybe looks like this. But if your learning rate is too big then if you start off there, gradient descent may overshoot the minimum and send you there. And if the learning rate is too big, you may overshoot again and it sends you there, and so on. So that, what you really wanted was for it to start here and for it to slowly go downhill, right? But if the learning rate is too big, then gradient descent can instead keep on overshooting the minimum. So that you actually end up getting worse and worse instead of getting to higher values of the cost function J(theta). So you end up with a plot like this and if you see a plot like this, the fix is usually just to use a smaller value of alpha. Oh, and also, of course, make sure your code doesn't have a bug of it. But usually too large a value of alpha could be a common problem. Similarly sometimes you may also see J(theta) do something like this, it may go down for a while then go up then go down for a while then go up go down for a while go up and so on. And a fix for something like this is also to use a smaller value of alpha. I'm not going to prove it here, but under other assumptions about the cost function J, that does hold true for linear regression, mathematicians have shown that if your learning rate alpha is small enough, then J(theta) should decrease on every iteration. So if this doesn't happen probably means the alpha's too big, you should set it smaller. But of course, you also don't want your learning rate to be too small because if you do that then the gradient descent can be slow to converge.And if alpha were too small, you might end up starting out here, say, and end up taking just minuscule baby steps. And just taking a lot of iterations before you finally get to the minimum, and so if alpha is too small, gradient descent can make very slow progress and be slow to converge.



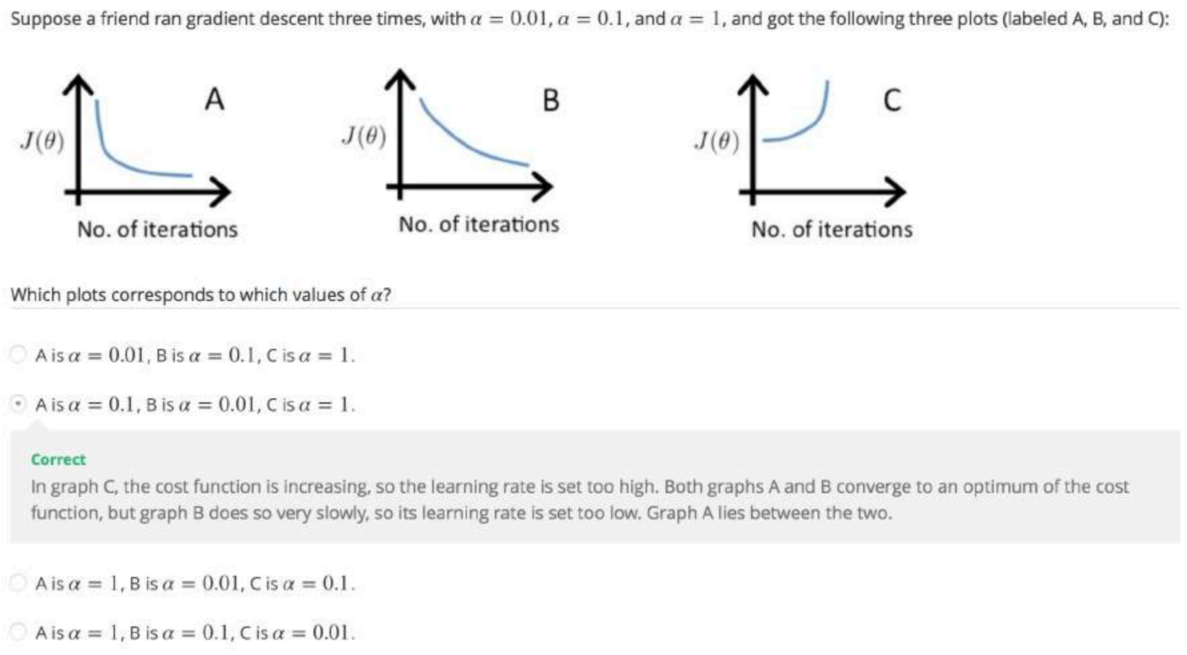
Making sure gradient descent is working correctly.

Gradient descent not working.

Use smaller α.

- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Question



Suppose a friend ran gradient descent three times, with $\alpha = 0.01$, $\alpha = 0.1$, and $\alpha = 1$, and got the following three plots (labeled A, B, and C):

Which plots corresponds to which values of α?

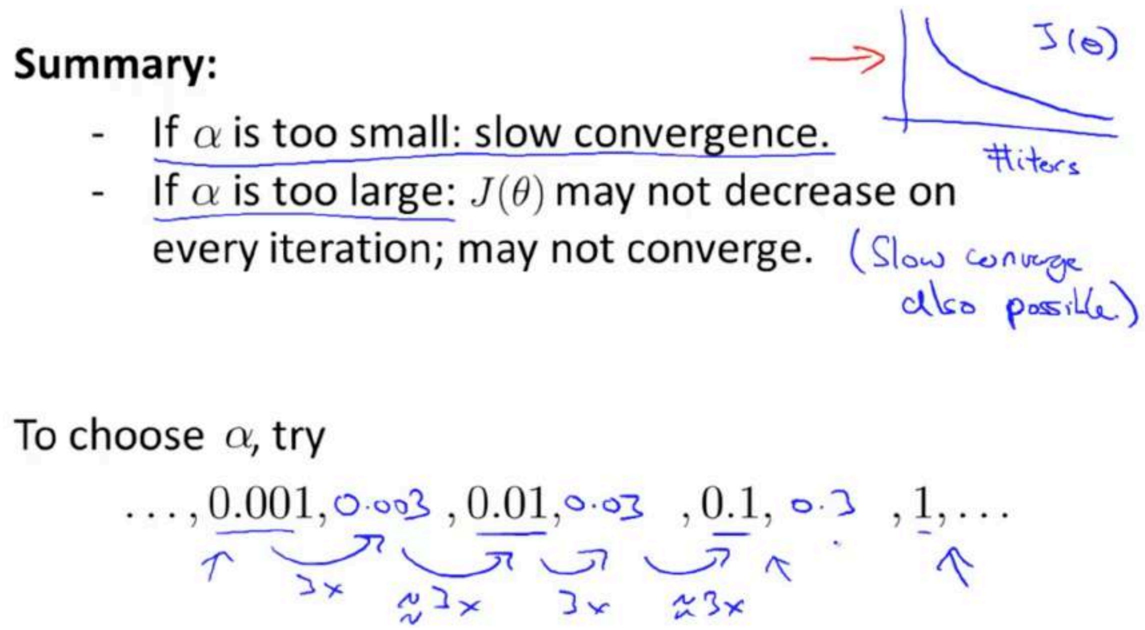
- A is $\alpha = 0.01$, B is $\alpha = 0.1$, C is $\alpha = 1$.
- [✓ A is $\alpha = 0.1$, B is $\alpha = 0.01$, C is $\alpha = 1$.]

Correct

In graph C, the cost function is increasing, so the learning rate is set too high. Both graphs A and B converge to an optimum of the cost function, but graph B does so very slowly, so its learning rate is set too low. Graph A lies between the two.

- A is $\alpha = 1$, B is $\alpha = 0.01$, C is $\alpha = 0.1$.
- A is $\alpha = 1$, B is $\alpha = 0.1$, C is $\alpha = 0.01$.

To summarize, if the learning rate is too small, you can have a slow convergence problem, and if the learning rate is too large, J(theta) may not decrease on every iteration and it may not even converge. In some cases if the learning rate is too large, slow convergence is also possible. But the more common problem you see is just that J(theta) may not decrease on every iteration. And in order to debug all of these things, often plotting that J(theta) as a function of the number of iterations can help you figure out what's going on. Concretely, what I actually do when I run gradient descent is I would try a range of values. So just try running gradient descent with a range of values for alpha, like 0.001 and 0.01. So these are factor of ten differences. And for these different values of alpha are just plot J(theta) as a function of number of iterations, and then pick the value of alpha that seems to be causing J(theta) to decrease rapidly. In fact, what I do actually isn't these steps of ten. So this is a scale factor of ten of each step up. What I actually do is try this range of values. And so on, where this is 0.001. I'll then increase the learning rate threefold to get 0.003. And then this step up, this is another roughly threefold increase from 0.003 to 0.01. And so these are, roughly, trying out gradient descents with each value I try being about 3x bigger than the previous value. So what I'll do is try a range of values until I've found one value that's too small and made sure that I've found one value that's too large. And then I'll sort of try to pick the largest possible value, or just something slightly smaller than the largest reasonable value that I found. And when I do that usually it just gives me a good learning rate for my problem. And if you do this too, maybe you'll be able to choose a good learning rate for your implementation of gradient descent.



Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge. (Slow converge also possible)

To choose α, try

$$\dots, \underline{0.001}, 0.003, \underline{0.01}, 0.03, \underline{0.1}, 0.3, \underline{1}, \dots$$

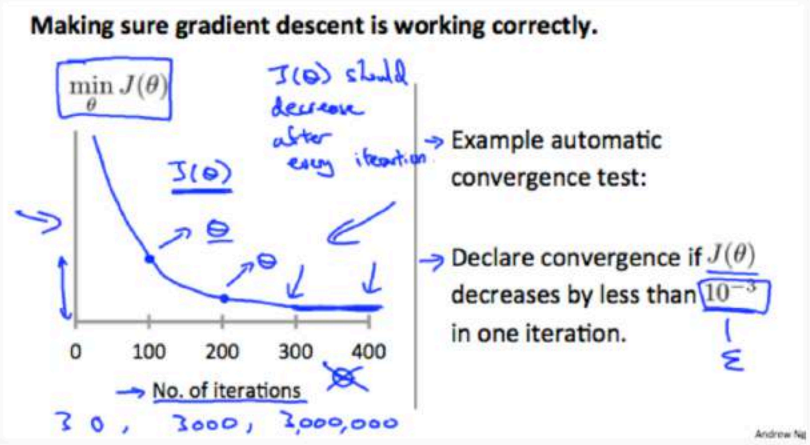
3x ≈3x 3x ≈3x

Gradient Descent in Practice 2 - Feature scaling (Transcript)

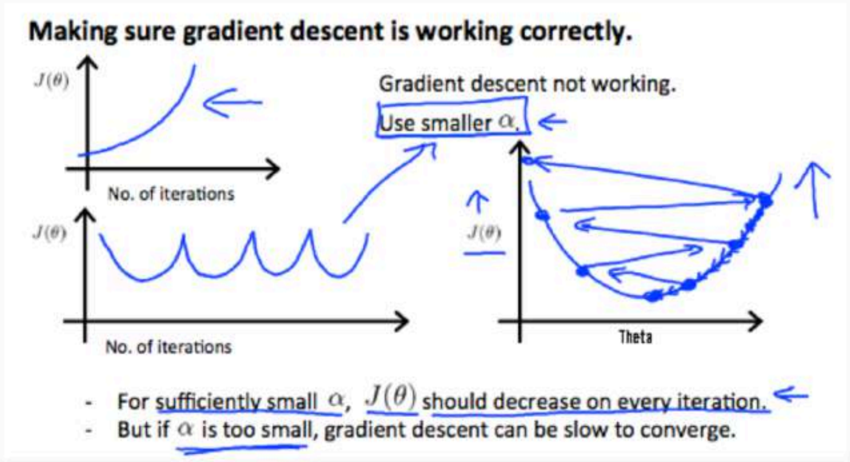
Note: [5:20 - the x -axis label in the right graph should be θ rather than No. of iterations]

Debugging gradient descent. Make a plot with *number of iterations* on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease α.

Automatic convergence test. Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10^{-3}. However in practice it's difficult to choose this threshold value.



It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.



To summarize:

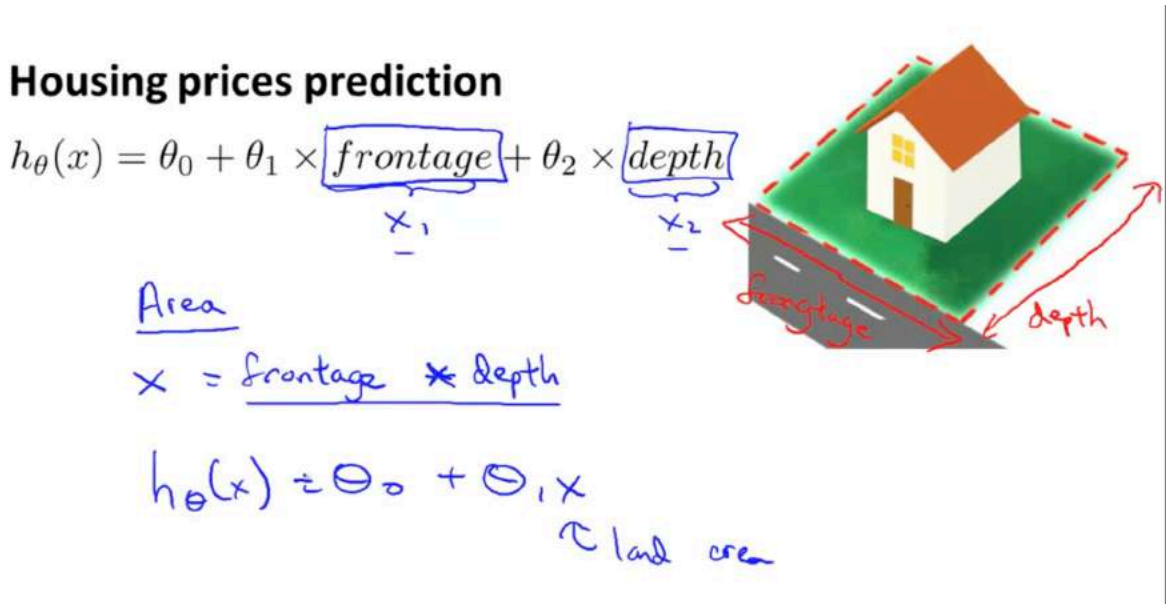
If α is too small: slow convergence.

If α is too large: may not decrease on every iteration and thus may not converge.

Features and Polynomial Regression (Video)

You now know about linear regression with multiple variables. In this video, I wanna tell you a bit about the choice of features that you have and how you can get different learning algorithm, sometimes very powerful ones by choosing appropriate features. And in particular I also want to tell you about polynomial regression allows you to use the machinery of linear regression to fit very complicated, even very non-linear functions.

Let's take the example of predicting the price of the house. Suppose you have two features, the frontage of house and the depth of the house. So, here's the picture of the house we're trying to sell. So, the frontage is defined as this distance is basically the width or the length of how wide your lot is if this that you own, and the depth of the house is how deep your property is, so there's a frontage, there's a depth. called frontage and depth. You might build a linear regression model like this where frontage is your first feature x1 and and depth is your second feature x2, but when you're applying linear regression, you don't necessarily have to use just the features x1 and x2 that you're given. What you can do is actually create new features by yourself. So, if I want to predict the price of a house, what I might do instead is decide that what really determines the size of the house is the area or the land area that I own. So, I might create a new feature. I'm just gonna call this feature x which is frontage, times depth. This is a multiplication symbol. It's a frontage x depth because this is the land area that I own and I might then select my hypothesis as that using just one feature which is my land area, right? Because the area of a rectangle is you know, the product of the length of the size So, depending on what insight you might have into a particular problem, rather than just taking the features [xx] that we happen to have started off with, sometimes by defining new features you might actually get a better model.



Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{\boxed{frontage}}_{\underline{x_1}} + \theta_2 \times \underbrace{\boxed{depth}}_{\underline{x_2}}$$

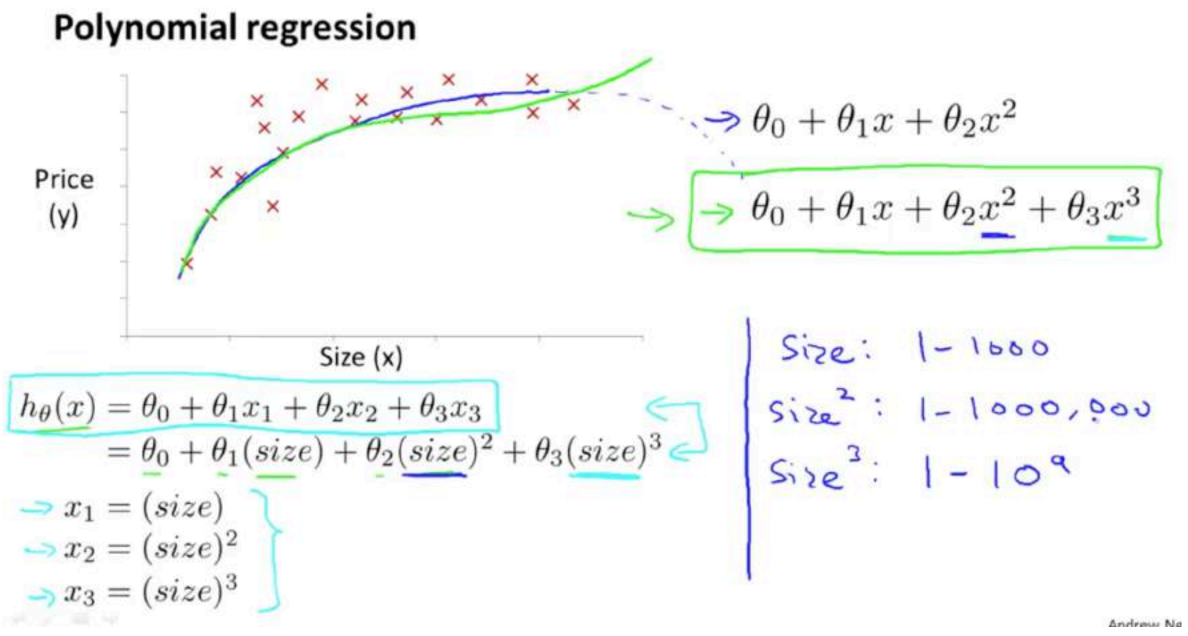
Area

x = frontage * depth

$h_\theta(x) \doteq \theta_0 + \theta_1 x$

land area

Closely related to the idea of choosing your features is this idea called polynomial regression. Let's say you have a housing price data set that looks like this. Then there are a few different models you might fit to this. One thing you could do is fit a quadratic model like this. It doesn't look like a straight line fits this data very well. So maybe you want to fit a quadratic model like this where you think the size, where you think the price is a quadratic function and maybe that'll give you, you know, a fit to the data that looks like that. But then you may decide that your quadratic model doesn't make sense because of a quadratic function, eventually this function comes back down and well, we don't think housing prices should go down when the size goes up too high. So then maybe we might choose a different polynomial model and choose to use instead a cubic function, and where we have now a third-order term and we fit that, maybe we get this sort of model, and maybe the green line is a somewhat better fit to the data cause it doesn't eventually come back down. So how do we actually fit a model like this to our data? Using the machinery of multivariant linear regression, we can do this with a pretty simple modification to our algorithm. The form of the hypothesis we, we know how the fit looks like this, where we say H of x is theta zero plus theta one x one plus x two theta X3. And if we want to fit this cubic model that I have boxed in green, what we're saying is that to predict the price of a house, it's theta 0 plus theta 1 times the size of the house plus theta 2 times the square size of the house. So this term is equal to that term. And then plus theta 3 times the cube of the size of the house raises that third term. In order to map these two definitions to each other, well, the natural way to do that is to set the first feature x one to be the size of the house, and set the second feature x two to be the square of the size of the house, and set the third feature x three to be the cube of the size of the house. And, just by choosing my three features this way and applying the machinery of linear regression, I can fit this model and end up with a cubic fit to my data. I just want to point out one more thing, which is that if you choose your features like this, then feature scaling becomes increasingly important. So if the size of the house ranges from one to a thousand, so, you know, from one to a thousand square feet, say, then the size squared of the house will range from one to one million, the square of a thousand, and your third feature x cubed, excuse me you, your third feature x three, which is the size cubed of the house, will range from one two ten to the nine, and so these three features take on very different ranges of values, and it's important to apply feature scaling if you're using gradient descent to get them into comparable ranges of values.



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

$$\rightarrow \boxed{\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3}$$

$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

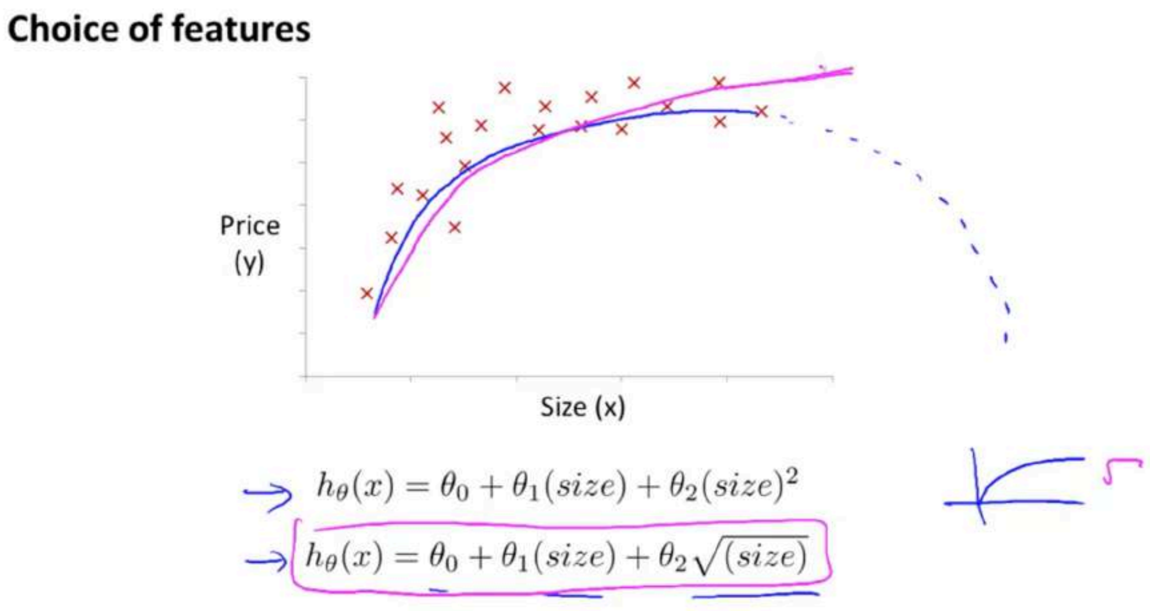
$$\rightarrow x_1 = (size)$$
$$\rightarrow x_2 = (size)^2$$
$$\rightarrow x_3 = (size)^3$$

Size: 1 - 1000

Size^2: 1 - 1000,000

Size^3: 1 - 10^9

Finally, here's one last example of how you really have broad choices in the features you use. Earlier we talked about how a quadratic model like this might not be ideal because, you know, maybe a quadratic model fits the data okay, but the quadratic function goes back down and we really don't want, right, housing prices that go down, to predict that, as the size of housing freezes. But rather than going to a cubic model there, you have, maybe, other choices of features and there are many possible choices. But just to give you another example of a reasonable choice, another reasonable choice might be to say that the price of a house is theta zero plus theta one times the size, and then plus theta two times the square root of the size, right? So the square root function is this sort of function, and maybe there will be some value of theta one, theta two, theta three, that will let you take this model and, for the curve that looks like that, and, you know, goes up, but sort of flattens out a bit and doesn't ever come back down. And, so, by having insight into, in this case, the shape of a square root function, and, into the shape of the data, by choosing different features, you can sometimes get better models. In this video, we talked about polynomial regression.

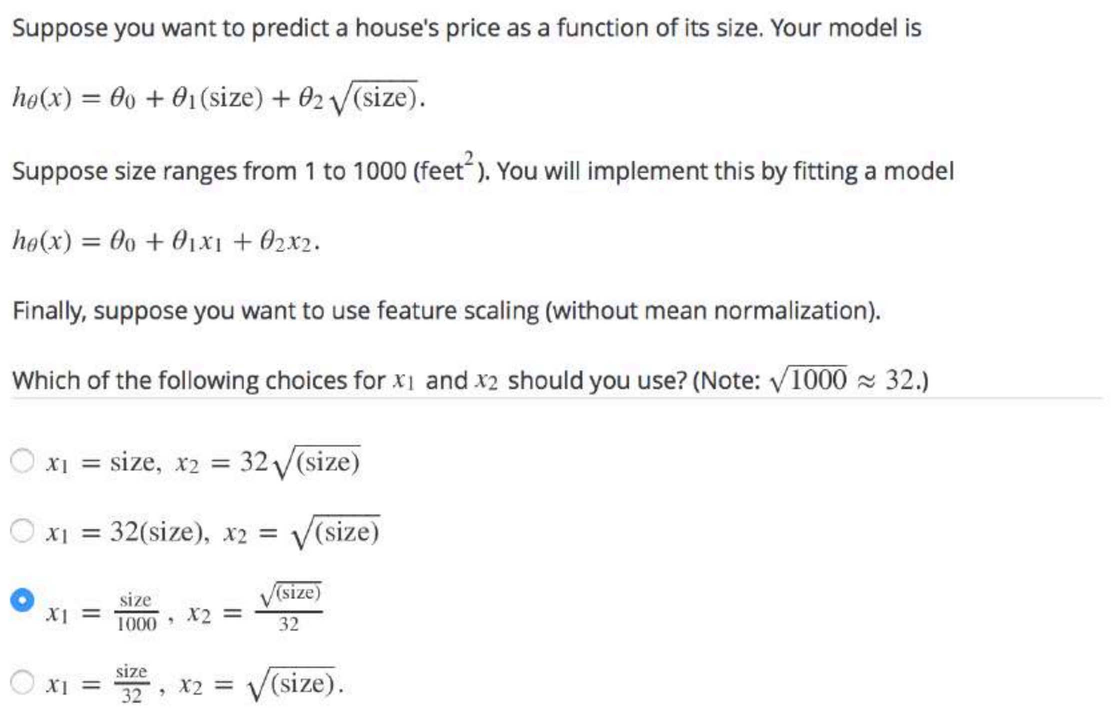


Choice of features

$$\rightarrow \quad h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$\rightarrow \quad \boxed{h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}}$$

Question



Suppose you want to predict a house's price as a function of its size. Your model is

$h_\theta(x) = \theta_0 + \theta_1(\text{size}) + \theta_2\sqrt{(\text{size})}$.

Suppose size ranges from 1 to 1000 (feet^2). You will implement this by fitting a model

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$.

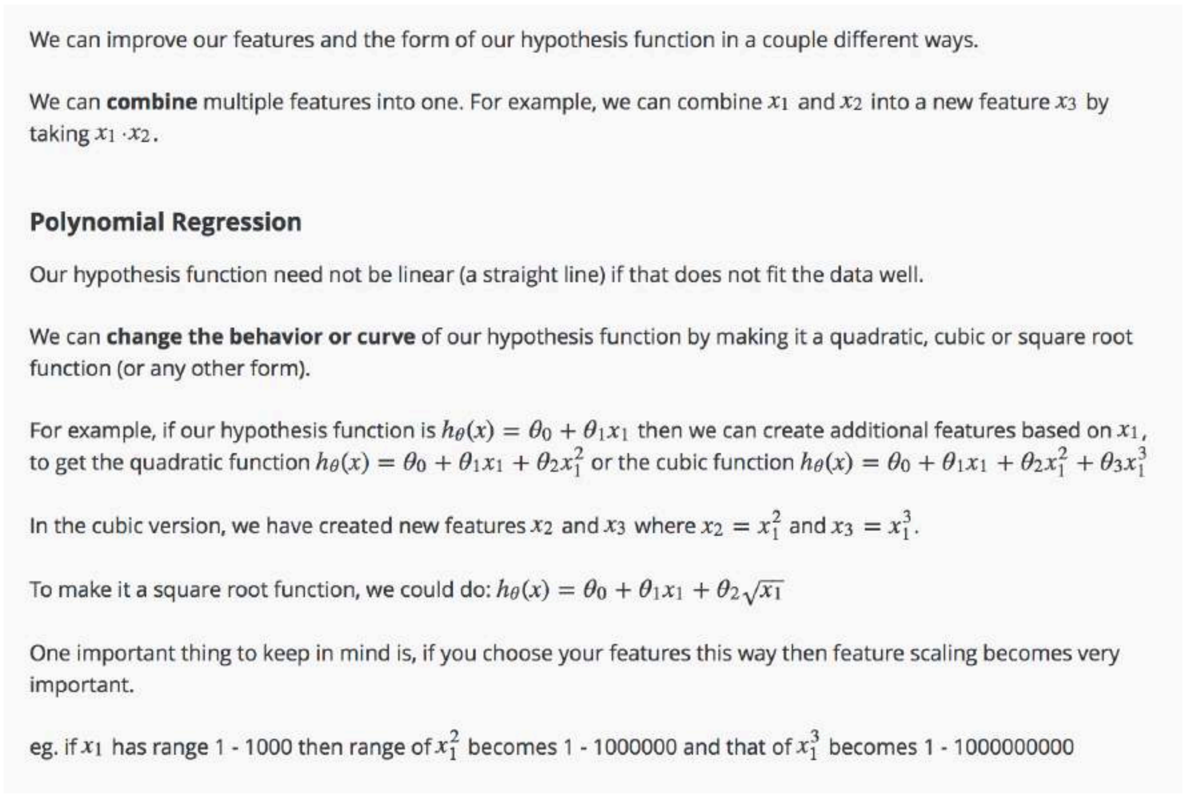
Finally, suppose you want to use feature scaling (without mean normalization).

Which of the following choices for x_1 and x_2 should you use? (Note: $\sqrt{1000} \approx 32$.)

- $x_1 = \text{size},\ x_2 = 32\sqrt{(\text{size})}$
- $x_1 = 32(\text{size}),\ x_2 = \sqrt{(\text{size})}$
- [✓ $x_1 = \frac{\text{size}}{1000},\ x_2 = \frac{\sqrt{(\text{size})}}{32}$]
- $x_1 = \frac{\text{size}}{32},\ x_2 = \sqrt{(\text{size})}$.

That is, how to fit a polynomial, like a quadratic function, or a cubic function, to your data. Was also throw out this idea, that you have a choice in what features to use, such as that instead of using the frontish and the depth of the house, maybe, you can multiply them together to get a feature that captures the land area of a house. In case this seems a little bit bewildering, that with all these different feature choices, so how do I decide what features to use. Later in this class, we'll talk about some algorithms were automatically choosing what features are used, so you can have an algorithm look at the data and automatically choose for you whether you want to fit a quadratic function, or a cubic function, or something else. But, until we get to those algorithms now I just want you to be aware that you have a choice in what features to use, and by designing different features you can fit more complex functions your data then just fitting a straight line to the data and in particular you can put polynomial functions as well and sometimes by appropriate insight into the feature simply get a much better model for your data.

Features and Polynomial Regression (Transcript)



We can improve our features and the form of our hypothesis function in a couple different ways.

We can **combine** multiple features into one. For example, we can combine x_1 and x_2 into a new feature x_3 by taking $x_1 \cdot x_2$.

Polynomial Regression

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can **change the behavior or curve** of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1, to get the quadratic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

In the cubic version, we have created new features x_2 and x_3 where $x_2 = x_1^2$ and $x_3 = x_1^3$.

To make it a square root function, we could do: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

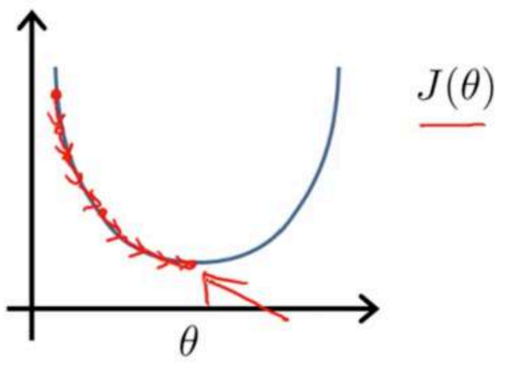
eg. if x_1 has range 1 - 1000 then range of x_1^2 becomes 1 - 1000000 and that of x_1^3 becomes 1 - 1000000000

Computing Parameters Analytically

Normal Equation (Video)

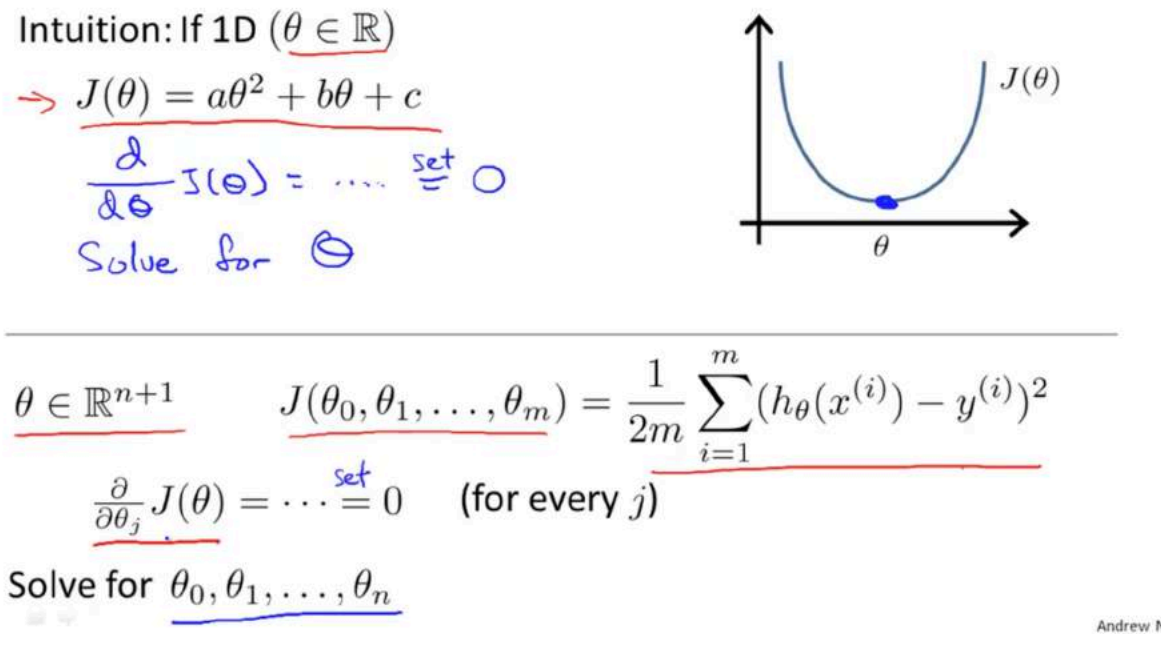
In this video, we'll talk about the normal equation, which for some linear regression problems, will give us a much better way to solve for the optimal value of the parameters theta. Concretely, so far the algorithm that we've been using for linear regression is gradient descent where in order to minimize the cost function J of Theta, we would take this iterative algorithm that takes many steps, multiple iterations of gradient descent to converge to the global minimum. In contrast, the normal equation would give us a method to solve for theta analytically, so that rather than needing to run this iterative algorithm, we can instead just solve for the optimal value for theta all at one go, so that in basically one step you get to the optimal value right there.

Gradient Descent



Normal equation: Method to solve for θ analytically.

It turns out the normal equation that has some advantages and some disadvantages, but before we get to that and talk about when you should use it, let's get some intuition about what this method does. For this week's planetary example, let's imagine, let's take a very simplified cost function J of Theta, that's just the function of a real number Theta. So, for now, imagine that Theta is just a scalar value or that Theta is just a row value. It's just a number, rather than a vector. Imagine that we have a cost function J that's a quadratic function of this real value parameter Theta, so J of Theta looks like that. Well, how do you minimize a quadratic function? For those of you that know a little bit of calculus, you may know that the way to minimize a function is to take derivatives and to set derivatives equal to zero. So, you take the derivative of J with respect to the parameter of Theta. You get some formula which I am not going to derive, you set that derivative equal to zero, and this allows you to solve for the value of Theda that minimizes J of Theta. That was a simpler case of when data was just real number. In the problem that we are interested in, Theta is no longer just a real number, but, instead, is this n+1-dimensional parameter vector, and, a cost function J is a function of this vector value or Theta 0 through Theta m. And, a cost function looks like this, some square cost function on the right. How do we minimize this cost function J? Calculus actually tells us that, if you, that one way to do so, is to take the partial derivative of J, with respect to every parameter of Theta J in turn, and then, to set all of these to 0. If you do that, and you solve for the values of Theta 0, Theta 1, up to Theta N, then, this would give you that values of Theta to minimize the cost function J. Where, if you actually work through the calculus and work through the solution to the parameters Theta 0 through Theta N, the derivation ends up being somewhat involved. And, what I am going to do in this video, is actually to not go through the derivation, which is kind of long and kind of involved, but what I want to do is just tell you what you need to know in order to implement this process so you can solve for the values of the thetas that corresponds to where the partial derivatives is equal to zero. Or alternatively, or equivalently, the values of Theta is that minimize the cost function J of Theta. I realize that some of the comments I made that made more sense only to those of you that are normally familiar with calculus. So, but if you don't know, if you're less familiar with calculus, don't worry about it. I'm just going to tell you what you need to know in order to implement this algorithm and get it to work.



Intuition: If 1D ($\theta \in \mathbb{R}$)

$\rightarrow$ $J(\theta) = a\theta^2 + b\theta + c$

$\frac{\partial}{\partial \theta} J(\theta) = \dots \overset{\text{set}}{=} 0$

Solve for θ

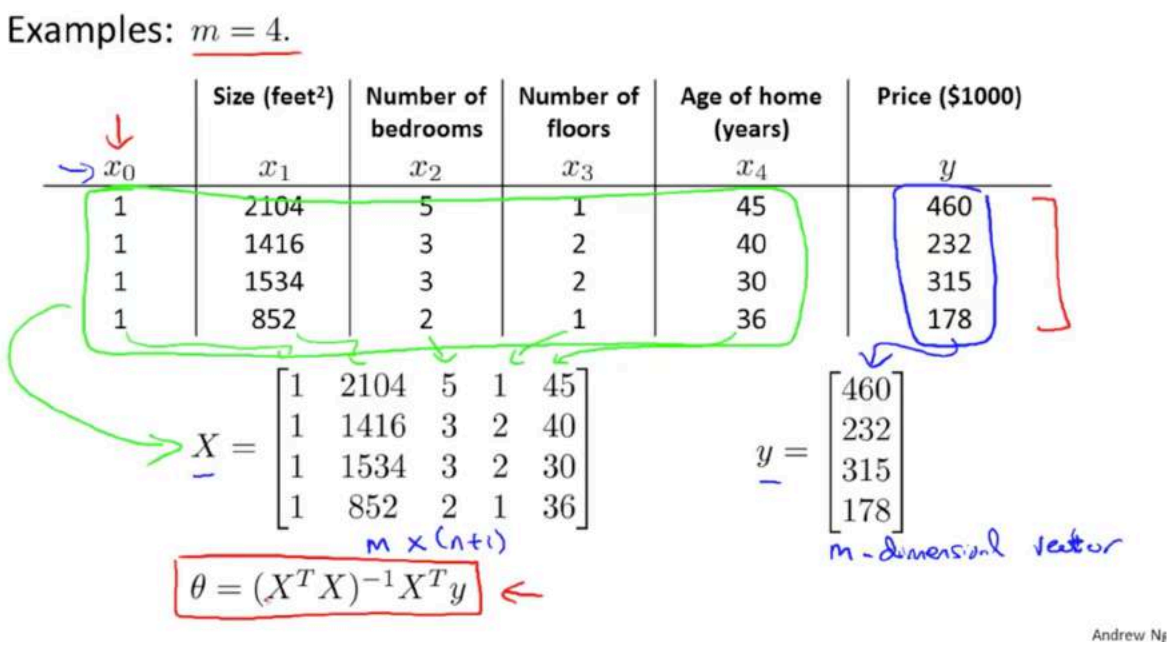
$\theta \in \mathbb{R}^{n+1}$ $\quad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots \overset{\text{set}}{=} 0$ (for every j)

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Andrew N

For the example that I want to use as a running example let's say that I have m = 4 training examples. In order to implement this normal equation at big, what I'm going to do is the following. I'm going to take my data set, so here are my four training examples. In this case let's assume that, you know, these four examples is all the data I have. What I am going to do is take my data set and add an extra column that corresponds to my extra feature, x0, that is always takes on this value of 1. What I'm going to do is I'm then going to construct a matrix called X that's a matrix are basically contains all of the features from my training data, so completely here is my here are all my features and we're going to take all those numbers and put them into this matrix "X", okay? So just, you know, copy the data over one column at a time and then I am going to do something similar for y's. I am going to take the values that I'm trying to predict and construct now a vector, like so and call that a vector y. So X is going to be a m by (n+1) - dimensional matrix, and Y is going to be a m-dimensional vector where m is the number of training examples and n is, n is a number of features, n+1, because of this extra feature X0 that I had. Finally if you take your matrix X and you take your vector Y, and if you just compute this, and set theta to be equal to X transpose X inverse times X transpose Y, this would give you the value of theta that minimizes your cost function. There was a lot that happened on the slides and I work through it using one specific example of one dataset. Let me just write this out in a slightly more general form and then let me just, and later on in this video let me explain this equation a little bit more. It is not yet entirely clear how to do this.



Examples: $m = 4$.

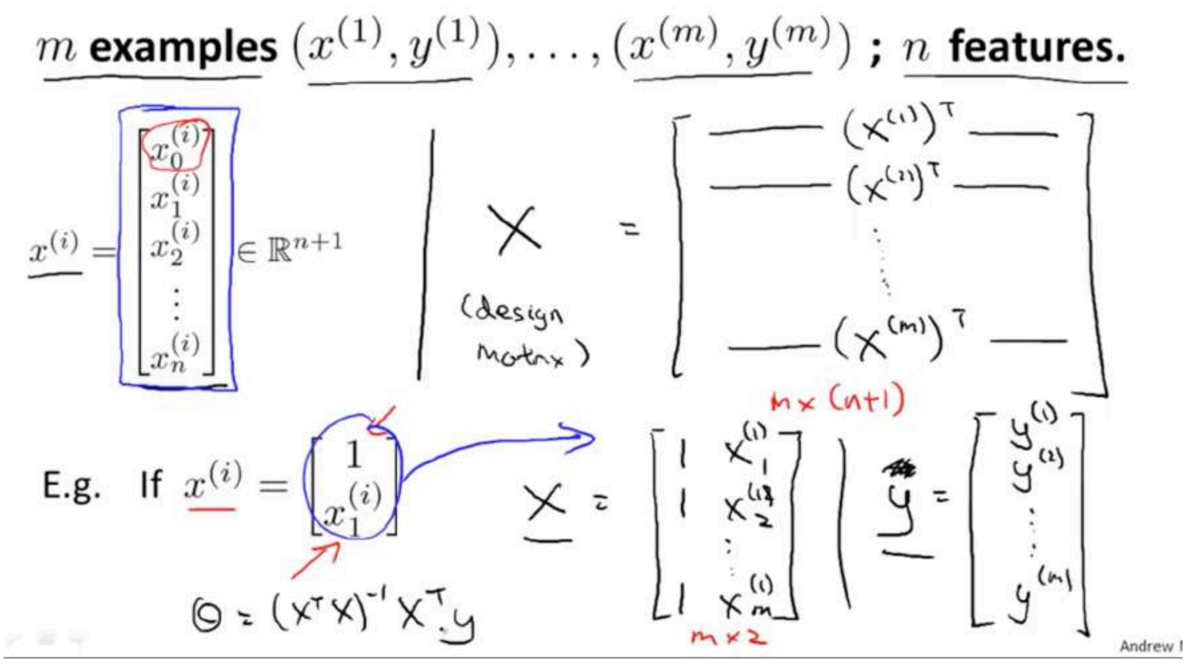
	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

$$\boxed{\theta = (X^T X)^{-1} X^T y} \leftarrow$$

Andrew Ng

In a general case, let us say we have M training examples so X1, Y1 up to Xn, Yn and n features. So, each of the training example x(i) may looks like a vector like this, that is a n+1 dimensional feature vector. The way I'm going to construct the matrix "X", this is also called the design matrix is as follows. Each training example gives me a feature vector like this. say, sort of n+1 dimensional vector. The way I am going to construct my design matrix x is only construct the matrix like this. and what I'm going to do is take the first training example, so that's a vector, take its transpose so it ends up being this, you know, long flat thing and make x1 transpose the first row of my design matrix. Then I am going to take my second training example, x2, take the transpose of that and put that as the second row of x and so on, down until my last training example. Take the transpose of that, and that's my last row of my matrix X. And, so, that makes my matrix X, an M by N +1 dimensional matrix. As a concrete example, let's say I have only one feature, really, only one feature other than X zero, which is always equal to 1. So if my feature vectors X-i are equal to this 1, which is X-0, then some real feature, like maybe the size of the house, then my design matrix, X, would be equal to this. For the first row, I'm going to basically take this and take its transpose. So, I'm going to end up with 1, and then X-1-1. For the second row, we're going to end up with 1 and then X-1-2 and so on down to 1, and then X-1-M. And thus, this will be a m by 2-dimensional matrix. So, that's how to construct the matrix X. And, the vector Y--sometimes I might write an arrow on top to denote that it is a vector, but very often I'll just write this as Y, either way. The vector Y is obtained by taking all all the labels, all the correct prices of houses in my training set, and just stacking them up into an M-dimensional vector, and that's Y. Finally, having constructed the matrix X and the vector Y, we then just compute theta as X'(1/X) x X'Y.



m **examples** $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$; n **features.**

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

(design matrix)

$m \times (n+1)$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(1)} \\ \vdots & \vdots \\ 1 & x_m^{(1)} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$m \times 2$

$$\Theta = (X^T X)^{-1} X^T y$$

Andrew I

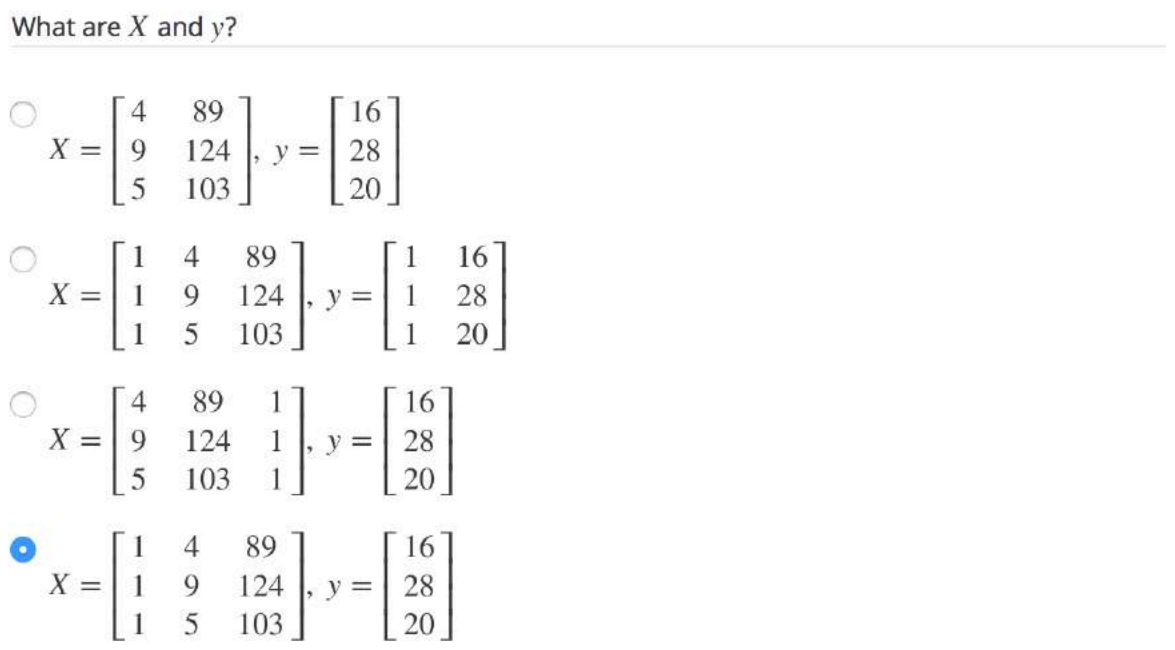
Question

Suppose you have the training in the table below:

age (x_1)	height in cm (x_2)	weight in kg (y)
4	89	16
9	124	28
5	103	20

You would like to predict a child's weight as a function of his age and height with the model

$\text{weight} = \theta_0 + \theta_1 \text{age} + \theta_2 \text{height}.$



What are X and y?

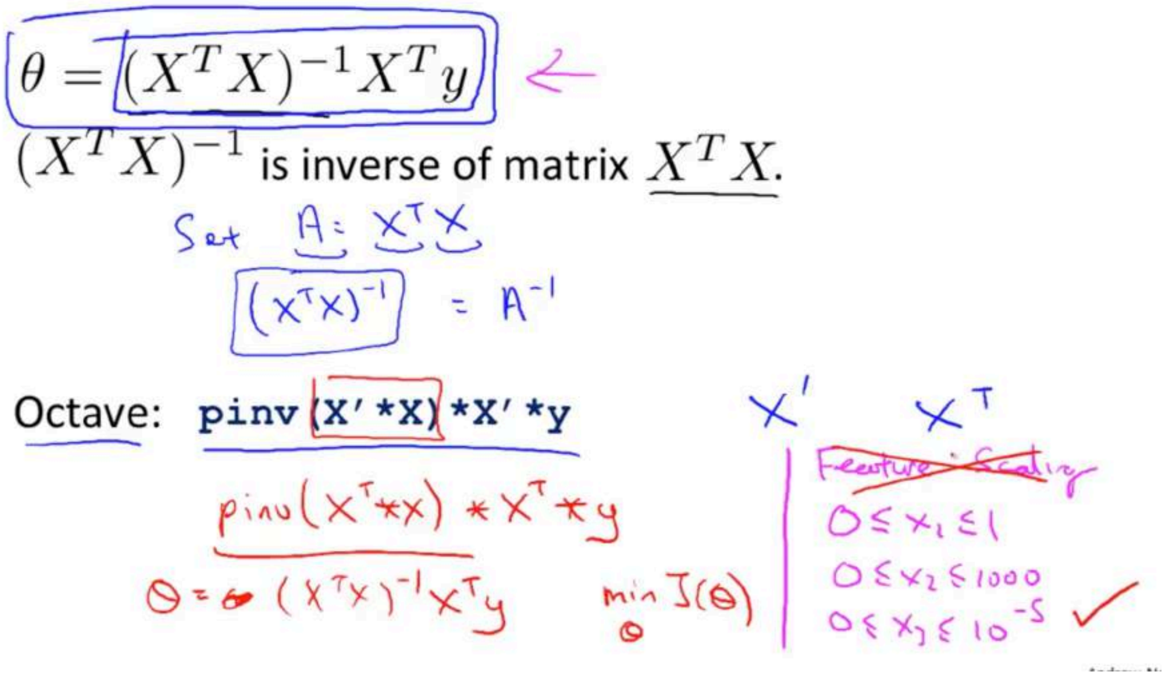
○ $X = \begin{bmatrix} 4 & 89 \\ 9 & 124 \\ 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$

○ $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 1 & 16 \\ 1 & 28 \\ 1 & 20 \end{bmatrix}$

○ $X = \begin{bmatrix} 4 & 89 & 1 \\ 9 & 124 & 1 \\ 5 & 103 & 1 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$

[✓ $X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}, y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$]

I just want to make sure that this equation makes sense to you and that you know how to implement it. So, you know, concretely, what is this X'(1/X)? Well, X'(1/X) is the inverse of the matrix X'X. Concretely, if you were to say set A to be equal to X' x X, so X' is a matrix, X' x X gives you another matrix, and we call that matrix A. Then, you know, X'(1/X) is just you take this matrix A and you invert it, right! This gives, let's say 1/A. And so that's how you compute this thing. You compute X'X and then you compute its inverse. We haven't yet talked about Octave. We'll do so in the later set of videos, but in the Octave programming language or a similar view, and also the matlab programming language is very similar. The command to compute this quantity, X transpose X inverse times X transpose Y, is as follows. In Octave X prime is the notation that you use to denote X transpose. And so, this expression that's boxed in red, that's computing X transpose times X. pinv is a function for computing the inverse of a matrix, so this computes X transpose X inverse, and then you multiply that by X transpose, and you multiply that by Y. So you end computing that formula which I didn't prove, but it is possible to show mathematically even though I'm not going to do so here, that this formula gives you the optimal value of theta in the sense that if you set theta equal to this, that's the value of theta that minimizes the cost function J of theta for the new regression. One last detail in the earlier video. I talked about the feature skill and the idea of getting features to be on similar ranges of Scales of similar ranges of values of each other. If you are using this normal equation method then feature scaling isn't actually necessary and is actually okay if, say, some feature X one is between zero and one, and some feature X two is between ranges from zero to one thousand and some feature x three ranges from zero to ten to the minus five and if you are using the normal equation method this is okay and there is no need to do features scaling, although of course if you are using gradient descent, then, features scaling is still important.



$$\boxed{\theta = \boxed{(X^TX)^{-1}X^Ty}}$$

$(X^TX)^{-1}$ is inverse of matrix $\underline{X^TX}$.

Set $A = X^TX$

$\boxed{(X^TX)^{-1}} = A^{-1}$

Octave: `pinv(X'*X)*X'*y`

$\mathrm{pinv}(X^T * X) * X^T * y$

$\theta = (X^TX)^{-1}X^Ty$ $\quad \min_{\theta} J(\theta)$

X' $\quad X^T$

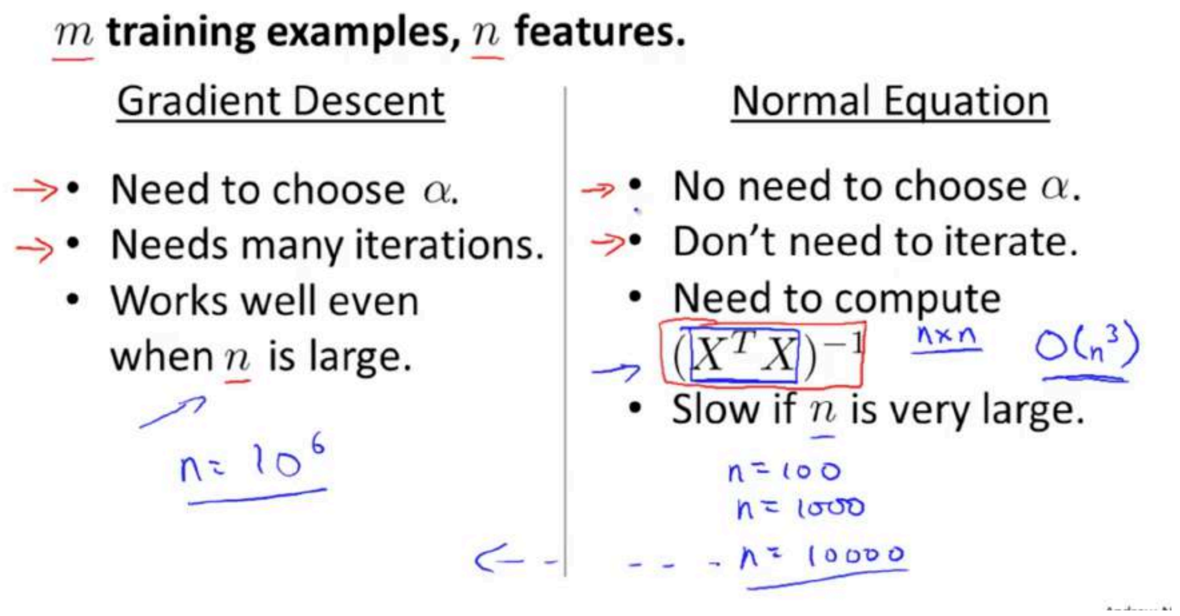
~~Feature Scaling~~

$0 \le x_1 \le 1$

$0 \le x_2 \le 1000$

$0 \le x_3 \le 10^{-5}$ ✓

Finally, where should you use the gradient descent and when should you use the normal equation method. Here are some of the their advantages and disadvantages. Let's say you have m training examples and n features. One disadvantage of gradient descent is that, you need to choose the learning rate Alpha. And, often, this means running it few times with different learning rate alphas and then seeing what works best. And so that is sort of extra work and extra hassle. Another disadvantage with gradient descent is it needs many more iterations. So, depending on the details, that could make it slower, although there's more to the story as we'll see in a second. As for the normal equation, you don't need to choose any learning rate alpha. So that, you know, makes it really convenient, makes it simple to implement. You just run it and it usually just works. And you don't need to iterate, so, you don't need to plot J of Theta or check the convergence or take all those extra steps. So far, the balance seems to favour normal the normal equation. Here are some disadvantages of the normal equation, and some advantages of gradient descent. Gradient descent works pretty well, even when you have a very large number of features. So, even if you have millions of features you can run gradient descent and it will be reasonably efficient. It will do something reasonable. In contrast to normal equation, In, in order to solve for the parameters data, we need to solve for this term. We need to compute this term, X transpose, X inverse. This matrix X transpose X. That's an n by n matrix, if you have n features. Because, if you look at the dimensions of X transpose the dimension of X, you multiply, figure out what the dimension of the product is, the matrix X transpose X is an n by n matrix where n is the number of features, and for almost computed implementations the cost of inverting the matrix, rose roughly as the cube of the dimension of the matrix. So, computing this inverse costs, roughly order, and cube time. Sometimes, it's slightly faster than N cube but, it's, you know, close enough for our purposes. So if n the number of features is very large, then computing this quantity can be slow and the normal equation method can actually be much slower. So if n is large then I might usually use gradient descent because we don't want to pay this all in q time. But, if n is relatively small, then the normal equation might give you a better way to solve the parameters. What does small and large mean? Well, if n is on the order of a hundred, then inverting a hundred-by-hundred matrix is no problem by modern computing standards. If n is a thousand, I would still use the normal equation method. Inverting a thousand-by-thousand matrix is actually really fast on a modern computer. If n is ten thousand, then I might start to wonder. Inverting a ten-thousand- by-ten-thousand matrix starts to get kind of slow, and I might then start to maybe lean in the direction of gradient descent, but maybe not quite. n equals ten thousand, you can sort of convert a ten-thousand-by-ten-thousand matrix. But if it gets much bigger than that, then, I would probably use gradient descent. So, if n equals ten to the sixth with a million features, then inverting a million-by-million matrix is going to be very expensive, and I would definitely favour gradient descent if you have that many features. So exactly how large set of features has to be before you convert a gradient descent, it's hard to give a strict number. But, for me, it is usually around ten thousand that I might start to consider switching over to gradient descents or maybe, some other algorithms that we'll talk about later in this class. To summarize, so long as the number of features is not too large, the normal equation gives us a great alternative method to solve for the parameter theta. Concretely, so long as the number of features is less than 1000, you know, I would use, I would usually is used in normal equation method rather than, gradient descent.



m **training examples,** n **features.**

Gradient Descent	Normal Equation
• Need to choose α.	• No need to choose α.
• Needs many iterations.	• Don't need to iterate.
• Works well even when n is large.	• Need to compute $\boxed{(\boxed{X^T X})^{-1}}$ $n \times n$ $O(n^3)$
$n = 10^6$	• Slow if n is very large.
	$n = 100$
	$n = 1000$
	$n = 10000$

To preview some ideas that we'll talk about later in this course, as we get to the more complex learning algorithm, for example, when we talk about classification algorithm, like a logistic regression algorithm, We'll see that those algorithm actually... The normal equation method actually do not work for those more sophisticated learning algorithms, and, we will have to resort to gradient descent for those algorithms. So, gradient descent is a very useful algorithm to know. The linear regression will have a large number of features and for some of the other algorithms that we'll see in this course, because, for them, the normal equation method just doesn't apply and doesn't work. But for this specific model of linear regression, the normal equation can give you a alternative that can be much faster, than gradient descent. So, depending on the detail of your algorithm, depending of the detail of the problems and how many features that you have, both of these algorithms are well worth knowing about.

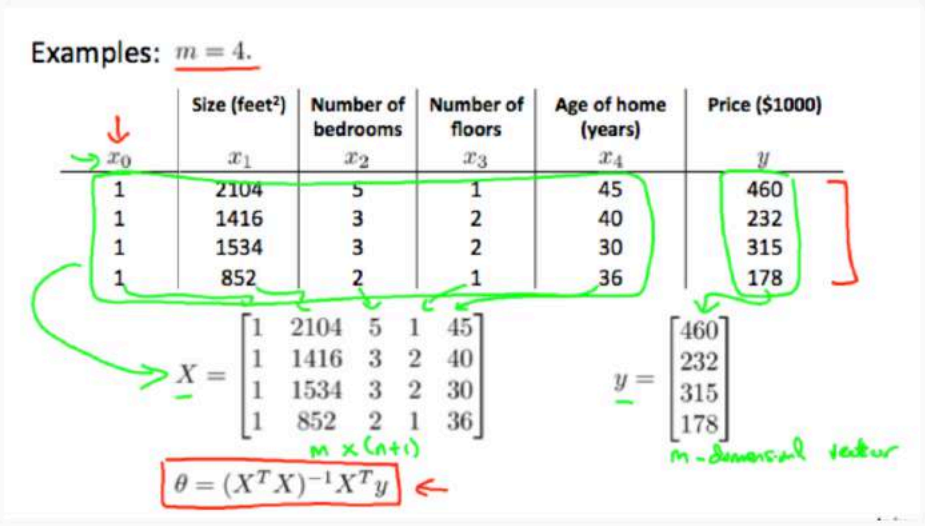
Normal Equation (Transcript)

Normal Equation

Note: [8:00 to 8:44 - The design matrix X (in the bottom right side of the slide) given in the example should have elements x with subscript 1 and superscripts varying from 1 to m because for all m training sets there are only 2 features x_0 and x_1. 12:56 - The X matrix is m by (n+1) and NOT n by n.]

Gradient descent gives one way of minimizing J. Let's discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj 's, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

$\theta = (X^T X)^{-1} X^T y$



Examples: $m = 4.$

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$\underline{X} = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad \underline{y} = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

$$\boxed{\theta = (X^T X)^{-1} X^T y} \leftarrow$$

There is **no need** to do feature scaling with the normal equation.

The following is a comparison of gradient descent and the normal equation:

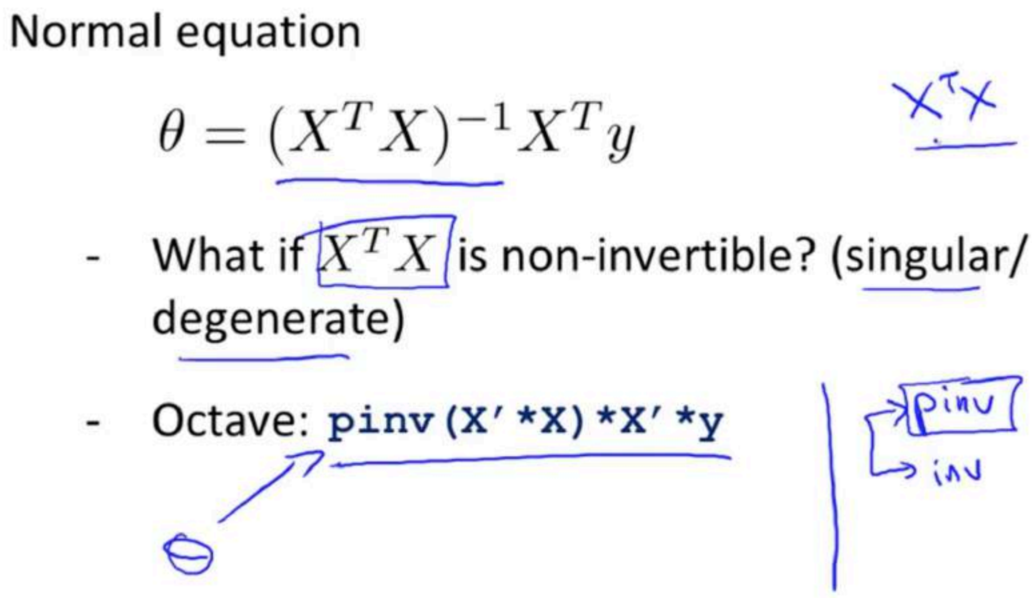
Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.

Normal Equation

Noninvertibility (Video)

In this video I want to talk about the Normal equation and non-invertibility. This is a somewhat more advanced concept, but it's something that I've often been asked about. And so I want to talk it here and address it here. But this is a somewhat more advanced concept, so feel free to consider this optional material. And there's a phenomenon that you may run into that may be somewhat useful to understand, but even if you don't understand the normal equation and linear progression, you should really get that to work okay. Here's the issue. For those of you there are, maybe some are more familiar with linear algebra, what some students have asked me is, when computing this Theta equals X transpose X inverse X transpose Y. What if the matrix X transpose X is non-invertible? So for those of you that know a bit more linear algebra you may know that only some matrices are invertible and some matrices do not have an inverse we call those non-invertible matrices. Singular or degenerate matrices. The issue or the problem of x transpose x being non invertible should happen pretty rarely. And in Octave if you implement this to compute theta, it turns out that this will actually do the right thing. I'm getting a little technical now, and I don't want to go into the details, but Octave hast two functions for inverting matrices. One is called pinv, and the other is called inv. And the differences between these two are somewhat technical. One's called the pseudo-inverse, one's called the inverse. But you can show mathematically that so long as you use the pinv function then this will actually compute the value of data that you want even if X transpose X is non-invertible. The specific details between inv. What is the difference between pinv? What is inv? That's somewhat advanced numerical computing concepts, I don't really want to get into. But I thought in this optional video, I'll try to give you little bit of intuition about what it means for X transpose X to be non-invertible. For those of you that know a bit more linear Algebra might be interested.



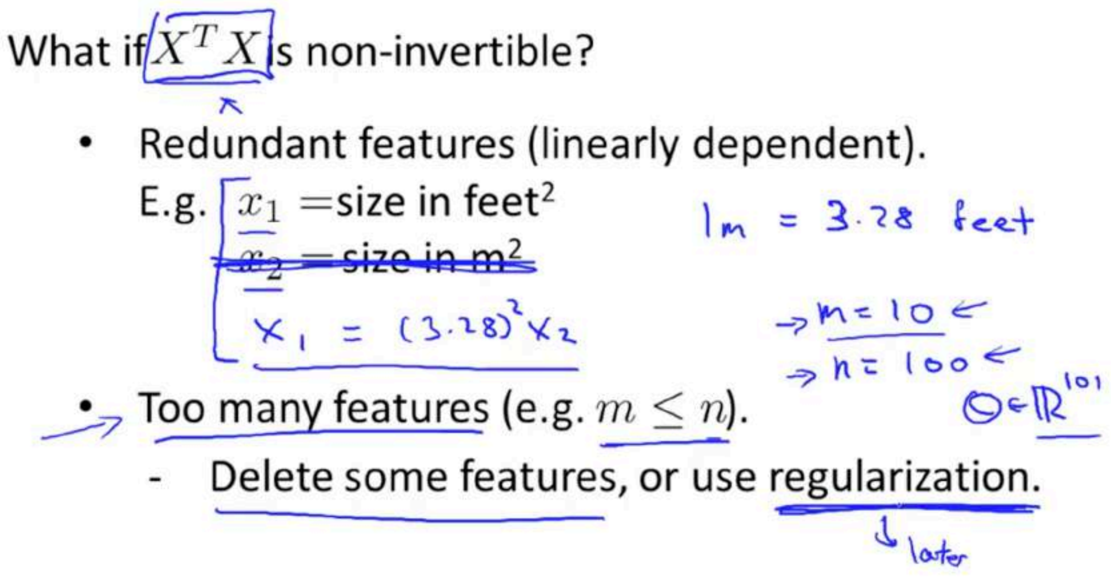
Normal equation

$$\theta = \underline{(X^T X)^{-1}} X^T y$$

$X^T X$

- What if $\boxed{X^T X}$ is non-invertible? (singular/ degenerate)
- Octave: `pinv(X'*X)*X'*y`

I'm not gonna prove this mathematically but if X transpose X is non-invertible, there usually two most common causes for this. The first cause is if somehow in your learning problem you have redundant features. Concretely, if you're trying to predict housing prices and if x1 is the size of the house in feet, in square feet and x2 is the size of the house in square meters, then you know 1 meter is equal to 3.28 feet Rounded to two decimals. And so your two features will always satisfy the constraint x1 equals 3.28 squared times x2. And you can show for those of you that are somewhat advanced in linear Algebra, but if you're explaining the algebra you can actually show that if your two features are related, are a linear equation like this. Then matrix X transpose X would be non-invertable. The second thing that can cause X transpose X to be non-invertable is if you are training, if you are trying to run the learning algorithm with a lot of features. Concretely, if m is less than or equal to n. For example, if you imagine that you have m = 10 training examples that you have n equals 100 features then you're trying to fit a parameter back to theta which is, you know, n plus one dimensional. So this is 101 dimensional, you're trying to fit 101 parameters from just 10 training examples. This turns out to sometimes work but not always be a good idea. Because as we'll see later, you might not have enough data if you only have 10 examples to fit you know, 100 or 101 parameters. We'll see later in this course why this might be too little data to fit this many parameters. But commonly what we do then if m is less than n, is to see if we can either delete some features or to use a technique called regularization which is something that we'll talk about later in this class as well, that will kind of let you fit a lot of parameters, use a lot features, even if you have a relatively small training set. But this regularization will be a later topic in this course. But to summarize if ever you find that x transpose x is singular or alternatively you find it non-invertable, what I would recommend you do is first look at your features and see if you have redundant features like this x1, x2. You're being linearly dependent or being a linear function of each other like so. And if you do have redundant features and if you just delete one of these features, you really don't need both of these features. If you just delete one of these features, that would solve your non-invertibility problem. And so I would first think through my features and check if any are redundant. And if so then keep deleting redundant features until they're no longer redundant. And if your features are not redundant, I would check if I may have too many features. And if that's the case, I would either delete some features if I can bear to use fewer features or else I would consider using regularization. Which is this topic that we'll talk about later.So that's it for the normal equation and what it means for if the matrix X transpose X is non-invertable but this is a problem that you should run that hopefully you run into pretty rarely and if you just implement it in octave using P and using the P n function which is called a pseudo inverse function so you could use a different linear out your alive in Is called a pseudo-inverse but that implementation should just do the right thing, even if X transpose X is non-invertable, which should happen pretty rarely anyways, so this should not be a problem for most implementations of linear regression.

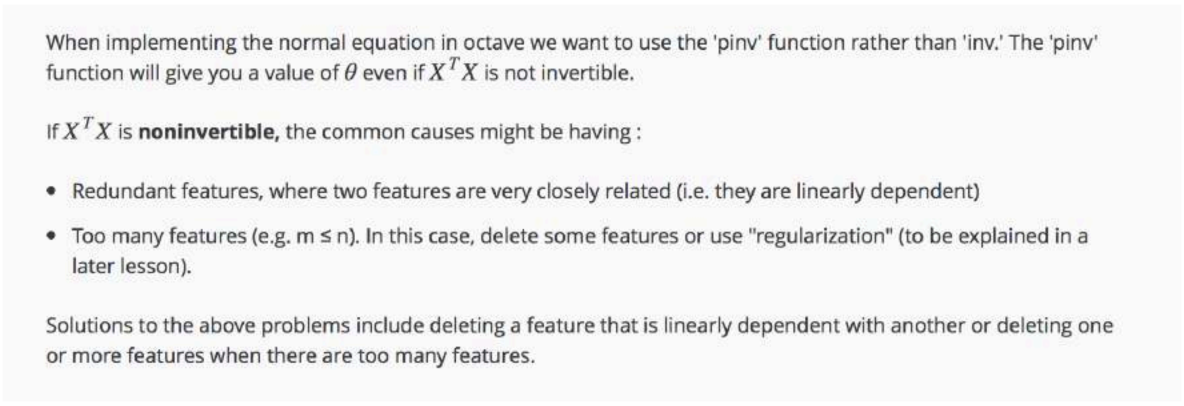


What if $\boxed{X^T X}$ is non-invertible?

- Redundant features (linearly dependent).
  E.g. x_1 =size in feet^2
  ~~x_2 =size in m^2~~
  $x_1 = (3.28)^2 x_2$
- Too many features (e.g. $m \leq n$).
  - Delete some features, or use regularization.

$1m = 3.28$ feet
$\rightarrow m = 10 \leftarrow$
$\rightarrow n = 100 \leftarrow$
$\theta \in \mathbb{R}^{101}$
later

Normal Equation Noninvertibility (Transcript)



When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.' The 'pinv' function will give you a value of θ even if X^TX is not invertible.

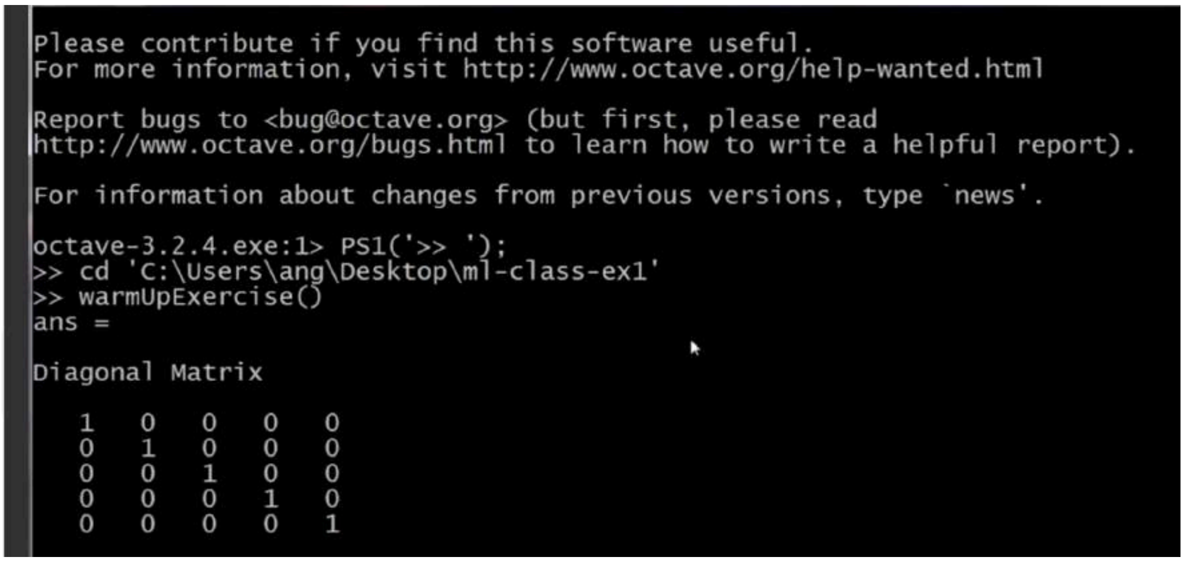
If X^TX is **noninvertible,** the common causes might be having :

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization" (to be explained in a later lesson).

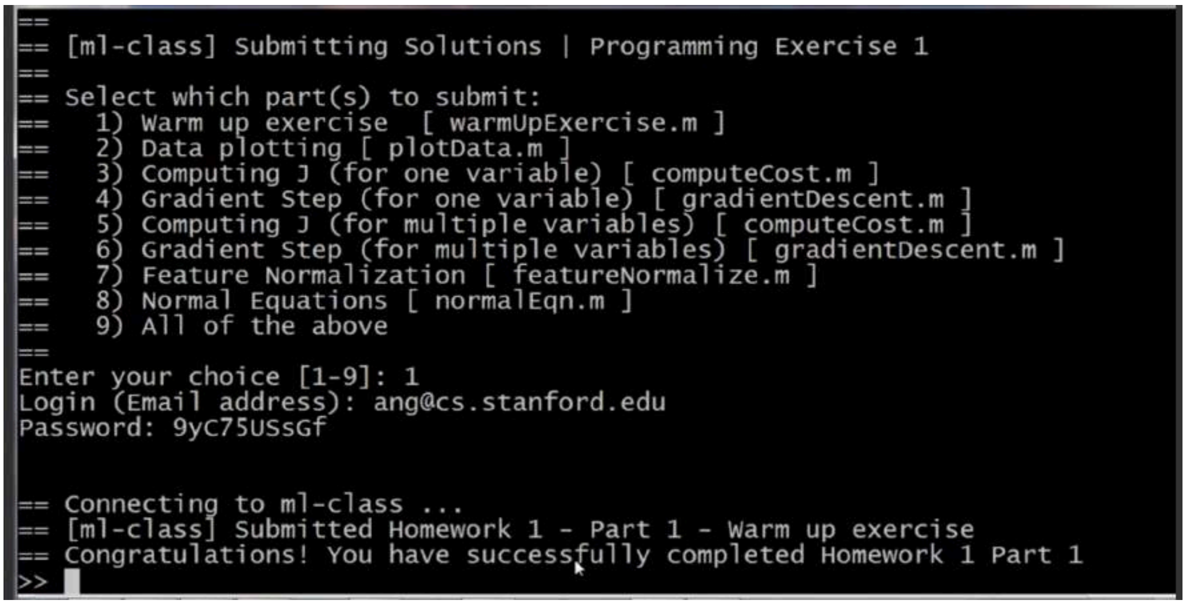
Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

Submitting Programming Assignments

Working on and Submitting Programming Assignments



```
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted.html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a helpful report).

For information about changes from previous versions, type `news'.

octave-3.2.4.exe:1> PS1('>> ');
>> cd 'C:\Users\ang\Desktop\ml-class-ex1'
>> warmUpExercise()
ans =

Diagonal Matrix

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   1
```



```
==
== [ml-class] Submitting Solutions | Programming Exercise 1
==
== Select which part(s) to submit:
==   1) Warm up exercise  [ warmUpExercise.m ]
==   2) Data plotting [ plotData.m ]
==   3) Computing J (for one variable) [ computeCost.m ]
==   4) Gradient Step (for one variable) [ gradientDescent.m ]
==   5) Computing J (for multiple variables) [ computeCost.m ]
==   6) Gradient Step (for multiple variables) [ gradientDescent.m ]
==   7) Feature Normalization [ featureNormalize.m ]
==   8) Normal Equations [ normalEqn.m ]
==   9) All of the above
==
Enter your choice [1-9]: 1
Login (Email address): ang@cs.stanford.edu
Password: 9yC75USsGf


== Connecting to ml-class ...
== [ml-class] Submitted Homework 1 - Part 1 - Warm up exercise
== Congratulations! You have successfully completed Homework 1 Part 1
>>
```

Programming tips from mentors

Thank you to Machine Learning Mentor, Tom Mosher, for compiling this list

Subject: Confused about "h(x) = theta' * x" vs. "h(x) = X * theta?"

Text:

The lectures and exercise PDF files are based on Prof. Ng's feeling that novice programmers will adapt to for-loop techniques more readily than vectorized methods. So the videos (and PDF files) are organized toward processing one training example at a time. The course uses column vectors (in most cases), so h (a scalar for one training example) is theta' * x.

Lower-case x typically indicates a single training example.

The more efficient vectorized techniques always use X as a matrix of all training examples, with each example as a row, and the features as columns. That makes X have dimensions of (m x n). where m is the number of training examples. This leaves us with h (a vector of all the hypothesis values for the entire training set) as X * theta, with dimensions of (m x 1).

X (as a matrix of all training examples) is denoted as upper-case X.

Throughout this course, dimensional analysis is your friend.

Subject: Tips from the Mentors: submit problems and fixing program errors

Text:

This post contains some frequently-used tips about the course, and to help get your programs working correctly.

The Most Important Tip:

Search the forum before posting a new question. If you've got a question, the chances are that someone else has already posted it, and received an answer. Save time for yourself and the Forum users by searching for topics before posting a new one.

Running your scripts:

At the Octave/Matlab command line, you do not need to include the ".m" portion of the script file name. If you include the ".m", you'll get an error message about an invalid indexing operation. So, run the Exercise 1 script by typing just "ex1" at the command line.

You also do not need to include parenthesis () when using the submit script. Just type "submit".

You cannot execute your functions by simply typing the name. All of the functions you will work on require a set of parameter values, enter between a set of parenthesis. Your three methods of testing your code are:

1 - use an exercise script, such as "ex1"

2 - use a Unit Test (see below) where you type-in the entire command line including the parameters.

3 - use the submit script.

Making the grader happy:

The submit grader uses a different test case than what is in the PDF file. These test cases use a different size of data set and are more sensitive to small errors than the ex test cases. Your code must work correctly with any size of data set.

Your functions must handle the general case. This means:

- You should avoid using hard-coded array indexes.

- You should avoid having fixed-length arrays and matrices.

It is very common for students to think that getting the same answer as listed in the PDF file means they should get full credit from the grader. This is a false hope. The PDF file is just one test case. The grader uses a different test case.

Also, the grader does not like your code to send any additional outputs to the workspace. So, every line of code should end with a semicolon.

Getting Help:

When you want help from the Forum community, please use this two-step procedure:

1 - Search the Forum for keywords that relate to your problem. Searching by the function name is a good start.

2 - If you don't find a suitable thread, then do this:

2a - Find the unit tests for that exercise (see below), and run the appropriate test. Attempt to debug your code.

2b - Take a screen capture of your whole console workspace (including the command line), and post it to the forum, along with any other useful information (computer type, Octave/Matlab version, other tests you've tried, etc).

Debugging:

If your code runs but gives the wrong answers, you can insert a "keyboard" command in your script, just before the function ends. This will cause the program to exit to the debugger, so you can inspect all your variables from the command line. This often is very helpful in analysing math errors, or trying out what commands to use to implement your function.

There are additional test cases and tutorials listed in pinned threads under "All Course Discussions". The test cases are especially helpful in debugging in situations where you get the expected output in ex but get no points or an error when submitting.

Unit Tests:

Each programming assignment has a "Discussions" area in the Forum. In this section you can often find "unit tests". These are additional test cases, which give you a command to type, and provides the expected results. It is always a good idea to test your functions using the unit tests before submitting to the grader.

If you run a unit test and do not get the correct results, you can most easily get help on the forums by posting a screen capture of your workspace - including the command line you entered, and the results.

Having trouble submitting your work to the grader?:

- This section will need to be supplemented with info appropriate to the new submission system. If you run the submit script and get a message that your identity can't be verified, be sure that you have logged-in using your Coursera account email and your Programming Assignment submission password.

- If you get the message "submit undefined", first check that you are in the working directory where you extracted the files from the ZIP archive. Use "cd" to get there if necessary.

- If the "submit undefined" error persists, or any other "function undefined" messages appear, try using the "addpath(pwd)" command to add your present working directory (pwd) to the Octave execution path.

-If the submit script crashes with an error message, please see the thread "Mentor tips for submitting your work" under "All Course Discussions".

-The submit script does not ask for what part of the exercise you want to submit. It automatically grades any function you have modified.

Found some errata in the course materials?

This course material has been used for many previous sessions. Most likely all of the errata has been discovered, and it's all documented in the 'Errata' section under 'Supplementary Materials'. Please check there before posting errata to the Forum.

Error messages with fmincg()

The "short-circuit" warnings are due to use a change in the syntax for conditional expressions (| and & vs || and &&) in the newer versions of Matlab. You can edit the fmincg.m file and the warnings may be resolved.

Warning messages about "automatic broadcasting"?

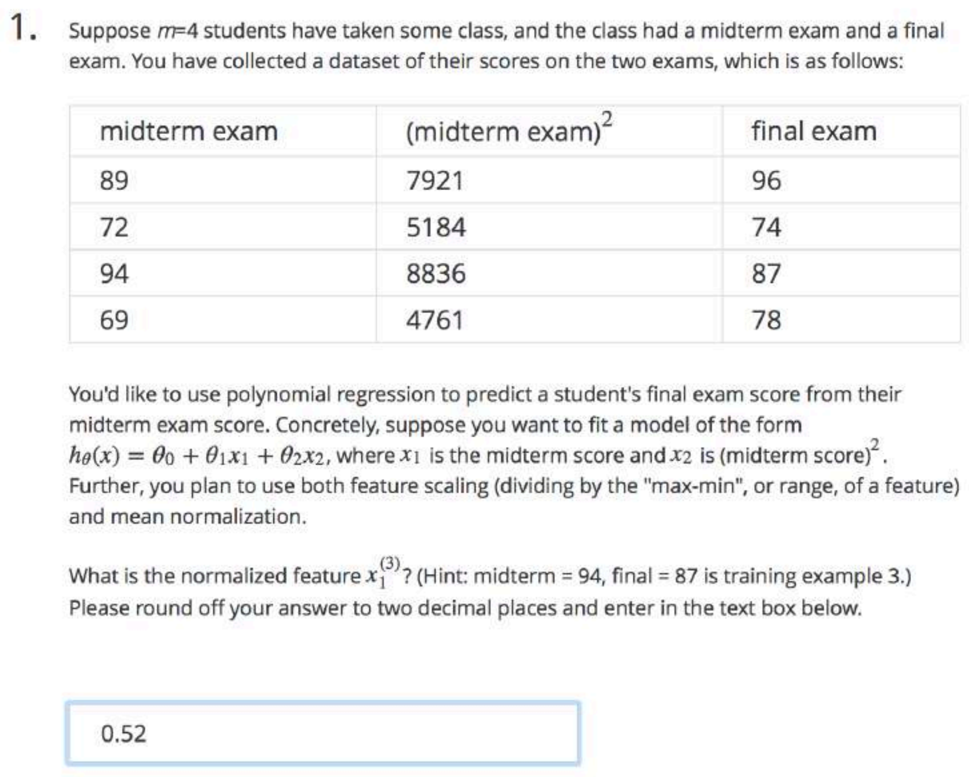
See this link for info.

Warnings about "divide by zero"

These are normal in some of the exercises, and do not represent a problem in your function. You can ignore them - Octave senses the issue and substitutes a +Inf or -Inf value so your program continues to execute.

Review

Quiz: Linear Regression with multiple variables



1. Suppose m=4 students have taken some class, and the class had a midterm exam and a final exam. You have collected a dataset of their scores on the two exams, which is as follows:

midterm exam	(midterm exam)^2	final exam
89	7921	96
72	5184	74
94	8836	87
69	4761	78

You'd like to use polynomial regression to predict a student's final exam score from their midterm exam score. Concretely, suppose you want to fit a model of the form $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$, where x_1 is the midterm score and x_2 is (midterm score)^2. Further, you plan to use both feature scaling (dividing by the "max-min", or range, of a feature) and mean normalization.

What is the normalized feature $x_1^{(3)}$? (Hint: midterm = 94, final = 87 is training example 3.) Please round off your answer to two decimal places and enter in the text box below.

0.52

2. You run gradient descent for 15 iterations

with $\alpha = 0.3$ and compute

$J(\theta)$ after each iteration. You find that the

value of $J(\theta)$ **decreases** quickly then levels

off. Based on this, which of the following conclusions seems

most plausible?

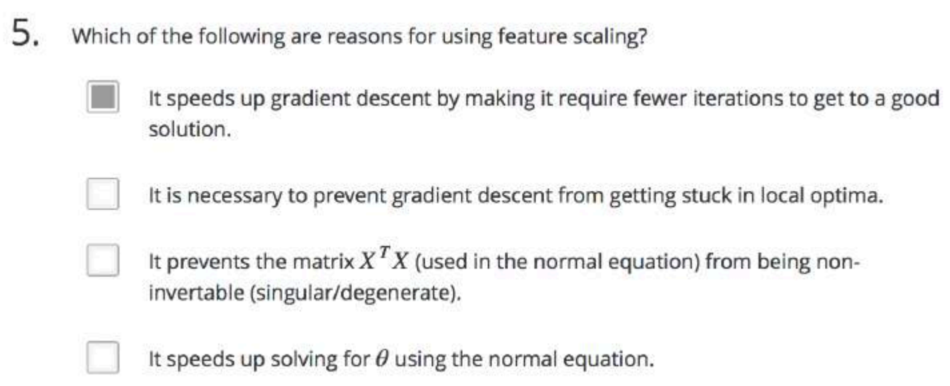
- [✓ $\alpha = 0.3$ is an effective choice of learning rate.]
- Rather than use the current value of α, it'd be more promising to try a smaller value of α (say $\alpha = 0.1$).
- Rather than use the current value of α, it'd be more promising to try a larger value of α (say $\alpha = 1.0$).

3. Suppose you have $m = 14$ training examples with $n = 3$ features (excluding the additional all-ones feature for the intercept term, which you should add). The normal equation is $\theta = (X^TX)^{-1}X^Ty$. For the given values of m and n, what are the dimensions of θ, X, and y in this equation?

- X is 14×4, y is 14×4, θ is 4×4
- X is 14×3, y is 14×1, θ is 3×1
- X is 14×3, y is 14×1, θ is 3×3
- [✓ X is 14×4, y is 14×1, θ is 4×1]

4. Suppose you have a dataset with $m = 1000000$ examples and $n = 200000$ features for each example. You want to use multivariate linear regression to fit the parameters θ to our data. Should you prefer gradient descent or the normal equation?

- [✓ Gradient descent, since $(X^TX)^{-1}$ will be very slow to compute in the normal equation.]
- Gradient descent, since it will always converge to the optimal θ.
- The normal equation, since it provides an efficient way to directly find the solution.
- The normal equation, since gradient descent might be unable to find the optimal θ.



5. Which of the following are reasons for using feature scaling?

[✓ It speeds up gradient descent by making it require fewer iterations to get to a good solution.]

☐ It is necessary to prevent gradient descent from getting stuck in local optima.

☐ It prevents the matrix X^TX (used in the normal equation) from being non-invertable (singular/degenerate).

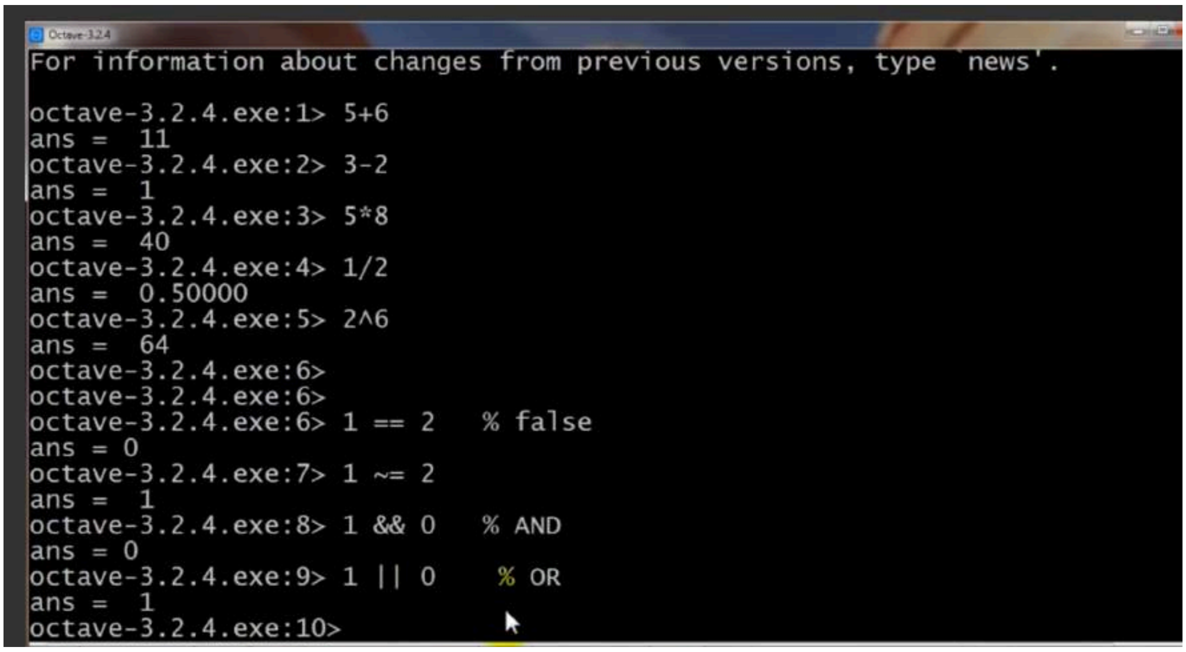
☐ It speeds up solving for θ using the normal equation.

Octave/Matlab Tutorial

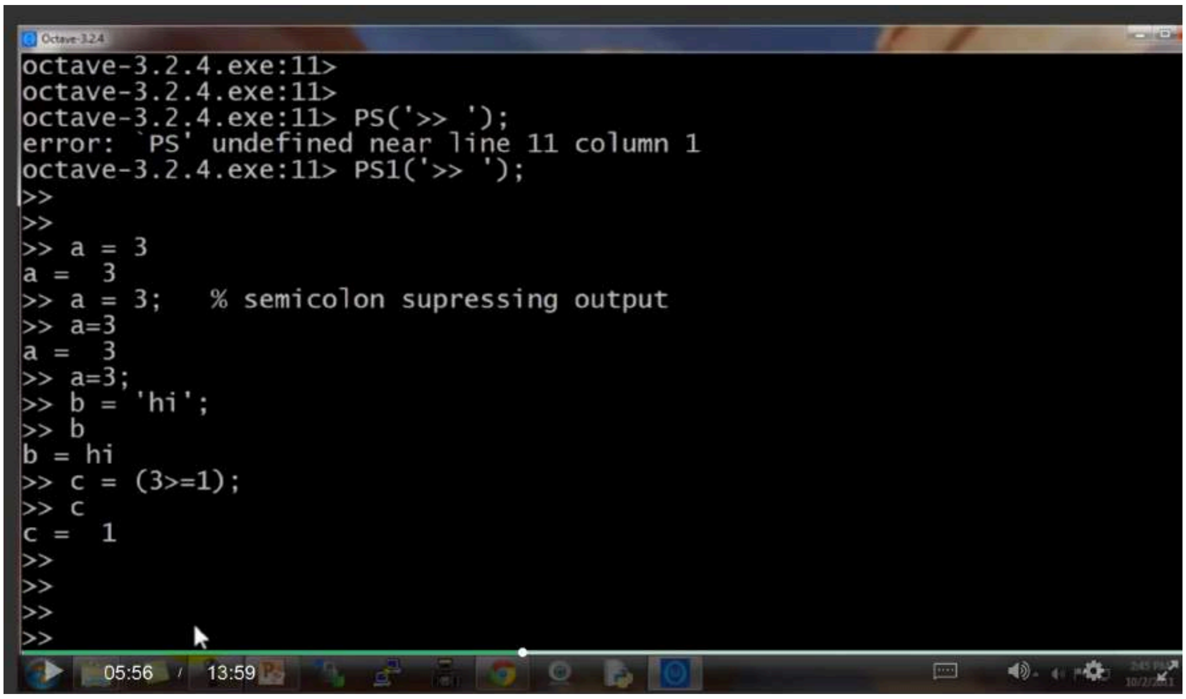
Basic Operations

You now know a bunch about machine learning. In this video, I like to teach you a programming language, Octave, in which you'll be able to very quickly implement the the learning algorithms we've seen already, and the learning algorithms we'll see later in this course. In the past, I've tried to teach machine learning using a large variety of different programming languages including C++ Java, Python, NumPy, and also Octave, and what I found was that students were able to learn the most productively learn the most quickly and prototype your algorithms most quickly using a relatively high level language like octave. In fact, what I often see in Silicon Valley is that if even if you need to build. If you want to build a large scale deployment of a learning algorithm, what people will often do is prototype and the language is Octave. Which is a great prototyping language. So you can sort of get your learning algorithms working quickly. And then only if you need to a very large scale deployment of it. Only then spend your time re-implementing the algorithm to C++ Java or some of the language like that. Because all the lessons we've learned is that a time or develop a time. That is your time. The machine learning's time is incredibly valuable. And if you can get your learning algorithms to work more quickly in Octave. Then overall you have a huge time savings by first developing the algorithms in Octave, and then implementing and maybe C++ Java, only after we have the ideas working. The most common prototyping language I see people use for machine learning are: Octave, MATLAB, Python, NumPy, and R. Octave is nice because open sourced. And MATLAB works well too, but it is expensive for to many people. But if you have access to a copy of MATLAB. You can also use MATLAB with this class. If you know Python, NumPy, or if you know R. I do see some people use it. But, what I see is that people usually end up developing somewhat more slowly, and you know, these languages. Because the Python, NumPy syntax is just slightly clunkier than the Octave syntax. And so because of that, and because we are releasing starter code in Octave. I strongly recommend that you not try to do the following exercises in this class in NumPy and R. But that I do recommend that you instead do the programming exercises for this class in octave instead. What I'm going to do in this video is go through a list of commands very, very quickly, and its goal is to quickly show you the range of commands and the range of things you can do in Octave. The course website will have a transcript of everything I do, and so after watching this video you can refer to the transcript posted on the course website when you want find a command. Concretely, what I recommend you do is first watch the tutorial videos. And after watching to the end, then install Octave on your computer. And finally, it goes to the course website, download the transcripts of the things you see in the session, and type in whatever commands seem interesting to you into Octave, so that it's running on your own computer, so you can see it run for yourself.

And with that let's get started. Here's my Windows desktop, and I'm going to start up Octave. And I'm now in Octave. And that's my Octave prompt. Let me first show the elementary operations you can do in Octave. So you type in 5 + 6. That gives you the answer of 11. 3 - 2. 5 x 8, 1/2, 2^6 is 64. So those are the elementary math operations. You can also do logical operations. So one equals two. This evaluates to false. The percent command here means a comment. So, one equals two, evaluates to false. Which is represents by zero. One not equals to two. This is true. So that returns one. Note that a not equal sign is this tilde equals symbol. And not bang equals. Which is what some other programming languages use. Lets see logical operations one and zero use a double ampersand sign to the logical AND. And that evaluates false. One or zero is the OR operation. And that evaluates to true. And I can XOR one and zero, and that evaluates to one. This thing over on the left, this Octave 324.x equals 11, this is the default Octave prompt. It shows the, what, the version in Octave and so on. If you don't want that prompt, there's a somewhat cryptic command PF quote, greater than, greater than and so on, that you can use to change the prompt. And I guess this quote a string in the middle. Your quote, greater than, greater than, space. That's what I prefer my Octave prompt to look like. So if I hit enter. Oops, excuse me. Like so. PS1 like so. Now my Octave prompt has changed to the greater than, greater than sign.Which, you know, looks quite a bit better. Next let's talk about Octave variables. I can take the variable A and assign it to 3. And hit enter. And now A is equal to 3. You want to assign a variable, but you don't want to print out the result. If you put a semicolon, the semicolon suppresses the print output. So to do that, enter, it doesn't print anything. Whereas A equals 3. mix it, print it out, where A equals, 3 semicolon doesn't print anything. I can do string assignment. B equals hi Now if I just enter B it prints out the variable B. So B is the string hi C equals 3 greater than colon 1. So, now C evaluates the true.



```
For information about changes from previous versions, type `news'.

octave-3.2.4.exe:1> 5+6
ans =  11
octave-3.2.4.exe:2> 3-2
ans =  1
octave-3.2.4.exe:3> 5*8
ans =  40
octave-3.2.4.exe:4> 1/2
ans =  0.50000
octave-3.2.4.exe:5> 2^6
ans =  64
octave-3.2.4.exe:6>
octave-3.2.4.exe:6>
octave-3.2.4.exe:6> 1 == 2   % false
ans = 0
octave-3.2.4.exe:7> 1 ~= 2
ans =  1
octave-3.2.4.exe:8> 1 && 0   % AND
ans = 0
octave-3.2.4.exe:9> 1 || 0    % OR
ans =  1
octave-3.2.4.exe:10>
```

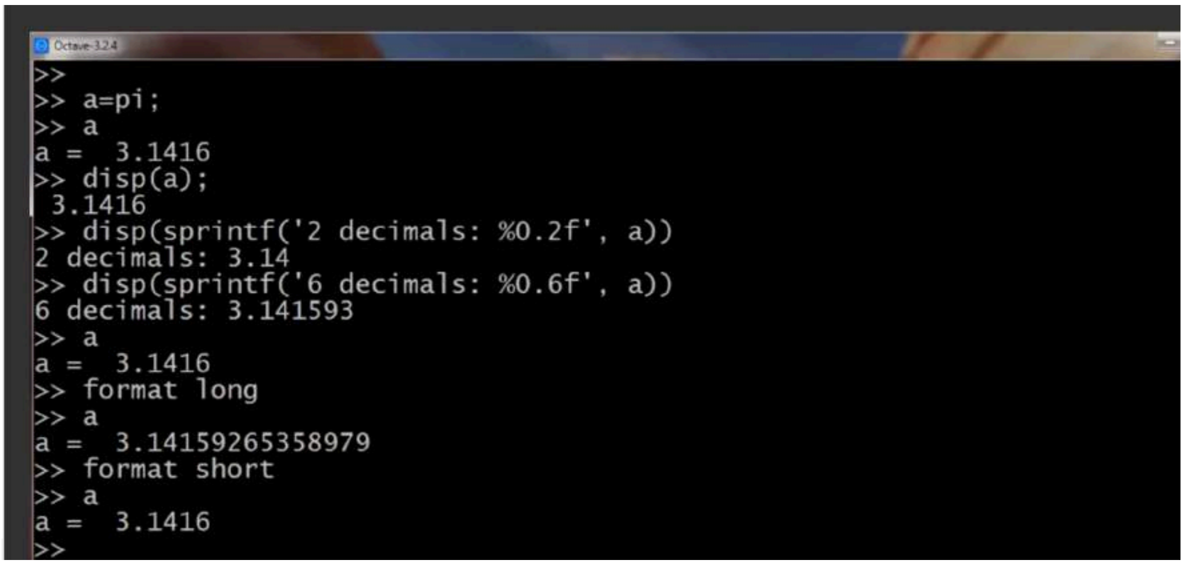


```
octave-3.2.4.exe:11>
octave-3.2.4.exe:11>
octave-3.2.4.exe:11> PS('>> ');
error: `PS' undefined near line 11 column 1
octave-3.2.4.exe:11> PS1('>> ');
>>
>>
>> a = 3
a =  3
>> a = 3;   % semicolon supressing output
>> a=3
a =  3
>> a=3;
>> b = 'hi';
>> b
b = hi
>> c = (3>=1);
>> c
c =  1
>>
>>
>>
>>
```

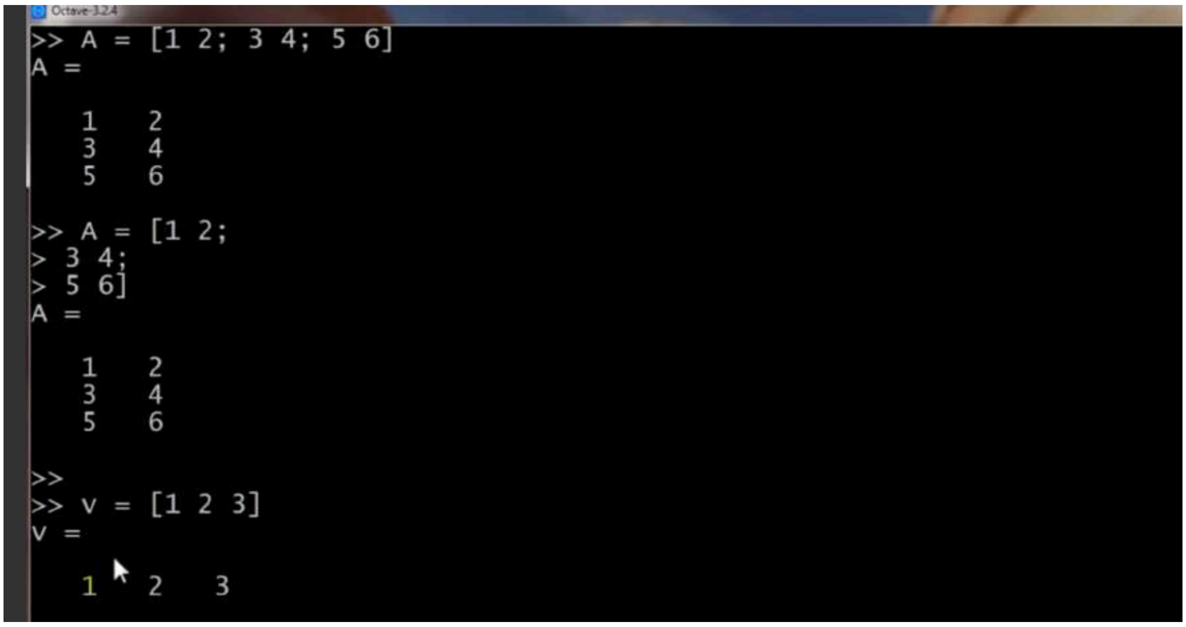
05:56 / 13:59

If you want to print out or display a variable, here's how you go about it. Let me set A equals Pi. And if I want to print A I can just type A like so, and it will print it out. For more complex printing there is also the DISP command which stands for Display. Display A just prints out A like so. You can also display strings so:

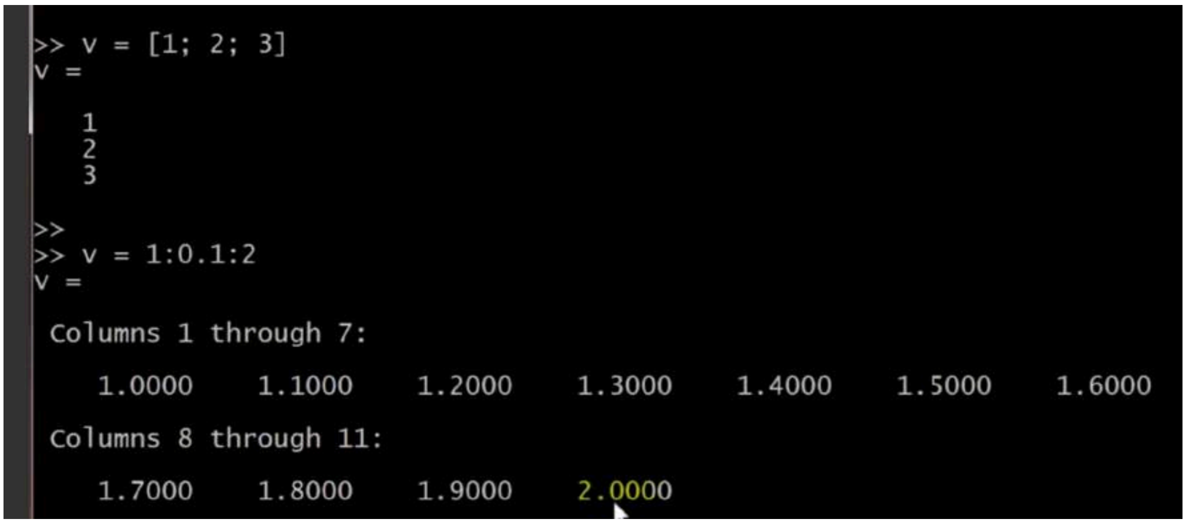
DISP, sprintf, two decimals, percent 0.2, F, comma, A. Like so. And this will print out the string. Two decimals, colon, 3.14. This is kind of an old style C syntax. For those of you that have programmed C before, this is essentially the syntax you use to print screen. So the Sprintf generates a string that is less than the 2 decimals, 3.1 plus string. This percent 0.2 F means substitute A into here, showing the two digits after the decimal points. And DISP takes the string DISP generates it by the Sprintf command. Sprintf. The Sprintf command. And DISP actually displays the string. And to show you another example, Sprintf six decimals percent 0.6 F comma A. And, this should print Pi with six decimal places. Finally, I was saying, a like so, looks like this. There are useful shortcuts that type type formats long. It causes strings by default. Be displayed to a lot more decimal places. And format short is a command that restores the default of just printing a small number of digits. Okay, that's how you work with variables. Now let's look at vectors and matrices. Let's say I want to assign MAT A to the matrix. Let me show you an example: 1, 2, semicolon, 3, 4, semicolon, 5, 6. This generates a three by two matrix A whose first row is 1, 2. Second row 3, 4. Third row is 5, 6. What the semicolon does is essentially say, go to the next row of the matrix. There are other ways to type this in. Type A 1, 2 semicolon 3, 4, semicolon, 5, 6, like so. And that's another equivalent way of assigning A to be the values of this three by two matrix. Similarly you can assign vectors. So V equals 1, 2, 3. This is actually a row vector. Or this is a 3 by 1 vector. Where that is a fat Y vector, excuse me, not, this is a 1 by 3 matrix, right. Not 3 by 1. If I want to assign this to a column vector, what I would do instead is do v 1;2;3. And this will give me a 3 by 1. There's a 1 by 3 vector. So this will be a column vector. Here's some more useful notation. V equals 1: 0.1: 2. What this does is it sets V to the bunch of elements that start from 1. And increments and steps of 0.1 until you get up to 2. So if I do this, V is going to be this, you know, row vector. This is what one by eleven matrix really. That's 1, 1.1, 1.2, 1.3 and so on until we get up to two.



```
>>
>> a=pi;
>> a
a =  3.1416
>> disp(a);
 3.1416
>> disp(sprintf('2 decimals: %0.2f', a))
2 decimals: 3.14
>> disp(sprintf('6 decimals: %0.6f', a))
6 decimals: 3.141593
>> a
a =  3.1416
>> format long
>> a
a =  3.14159265358979
>> format short
>> a
a =  3.1416
>>
```

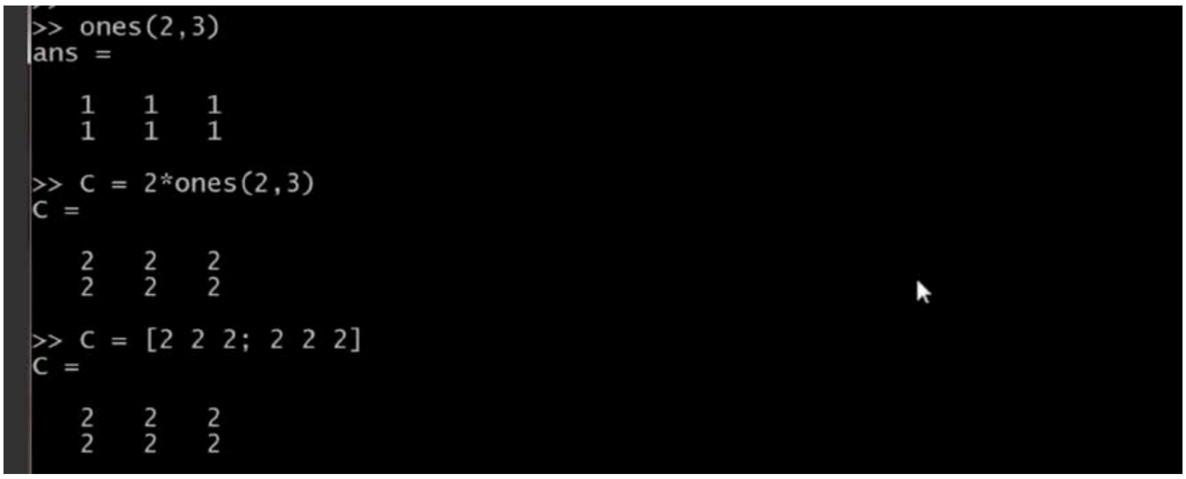


```
>> A = [1 2; 3 4; 5 6]
A =

   1   2
   3   4
   5   6

>> A = [1 2;
> 3 4;
> 5 6]
A =

   1   2
   3   4
   5   6

>>
>> v = [1 2 3]
v =

   1   2   3
```

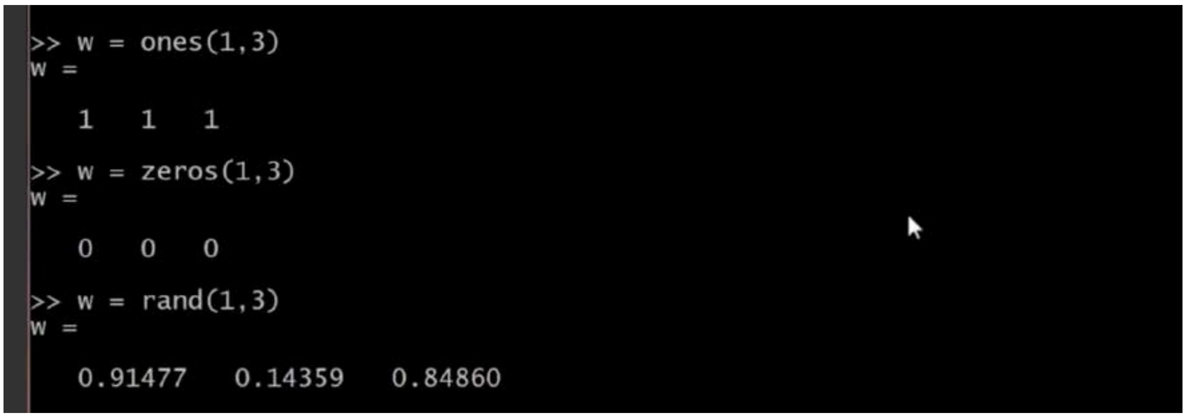


```
>> v = [1; 2; 3]
v =

   1
   2
   3

>>
>> v = 1:0.1:2
v =

 Columns 1 through 7:

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000

 Columns 8 through 11:

    1.7000    1.8000    1.9000    2.0000
```

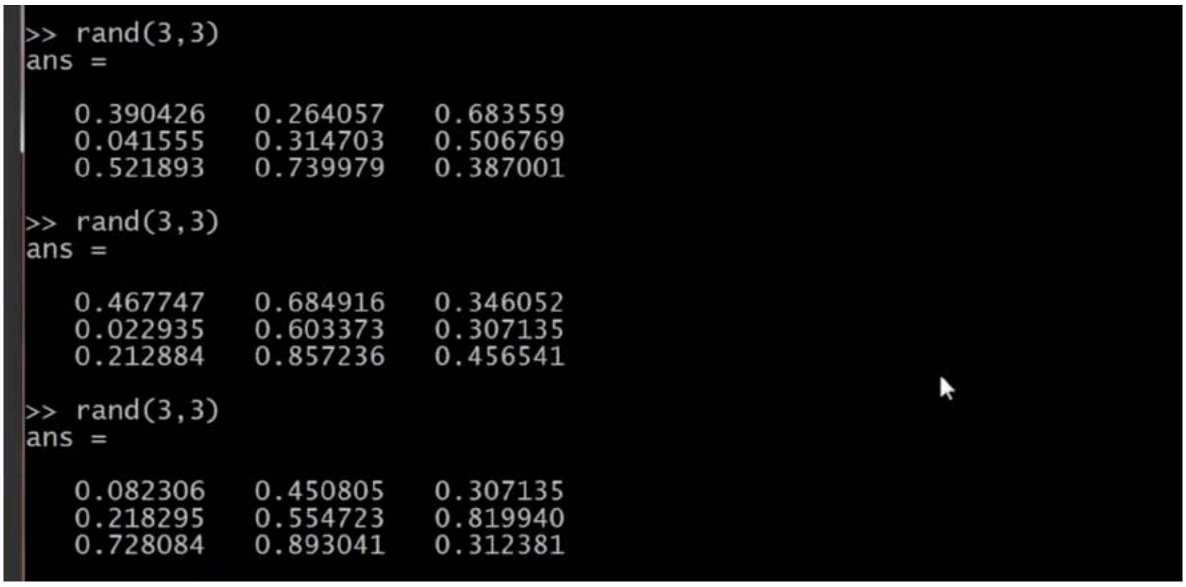
Now, and I can also set V equals one colon six, and that sets V to be these numbers. 1 through 6, okay. Now here are some other ways to generate matrices. Ones 2.3 is a command that generates a matrix that is a two by three matrix that is the matrix of all ones. So if I set that c2 times ones two by three this generates a two by three matrix that is all two's. You can think of this as a shorter way of writing this and c2,2,2's and you can call them 2,2,2, which would also give you the same result. Let's say W equals one's, one by three, so this is going to be a row vector or a row of three one's and similarly you can also say w equals zeroes, one by three, and this generates a matrix. A one by three matrix of all zeros. Just a couple more ways to generate matrices . If I do W equals Rand one by three, this gives me a one by three matrix of all random numbers. If I do Rand three by three. This gives me a three by three matrix of all random numbers drawn from the uniform distribution between zero and one. So every time I do this, I get a different set of random numbers drawn uniformly between zero and one. For those of you that know what a Gaussian random variable is or for those of you that know what a normal random variable is, you can also set W equals Rand N, one by three. And so these are going to be three values drawn from a Gaussian distribution with mean zero and variance or standard deviation equal to one. And you can set more complex things like W equals minus six, plus the square root ten, times, lets say Rand N, one by ten thousand. And I'm going to put a semicolon at the end because I don't really want this printed out. This is going to be a what? Well, it's going to be a vector of, with a hundred thousand, excuse me, ten thousand elements. So, well, actually, you know what? Let's print it out. So this will generate a matrix like this. Right? With 10,000 elements. So that's what W is.



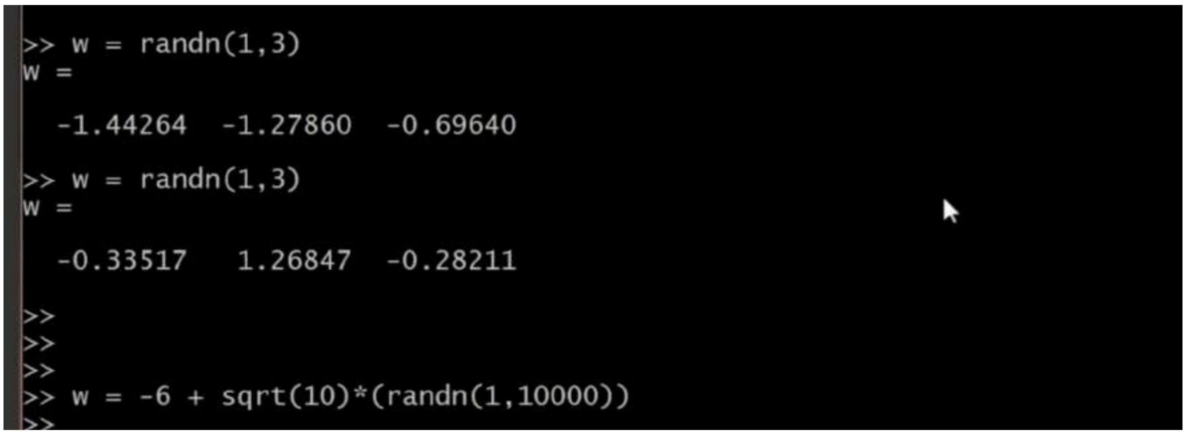
```
>> ones(2,3)
ans =

   1   1   1
   1   1   1

>> C = 2*ones(2,3)
C =

   2   2   2
   2   2   2

>> C = [2 2 2; 2 2 2]
C =

   2   2   2
   2   2   2
```



```
>> w = ones(1,3)
w =

   1   1   1

>> w = zeros(1,3)
w =

   0   0   0

>> w = rand(1,3)
w =

   0.91477   0.14359   0.84860
```

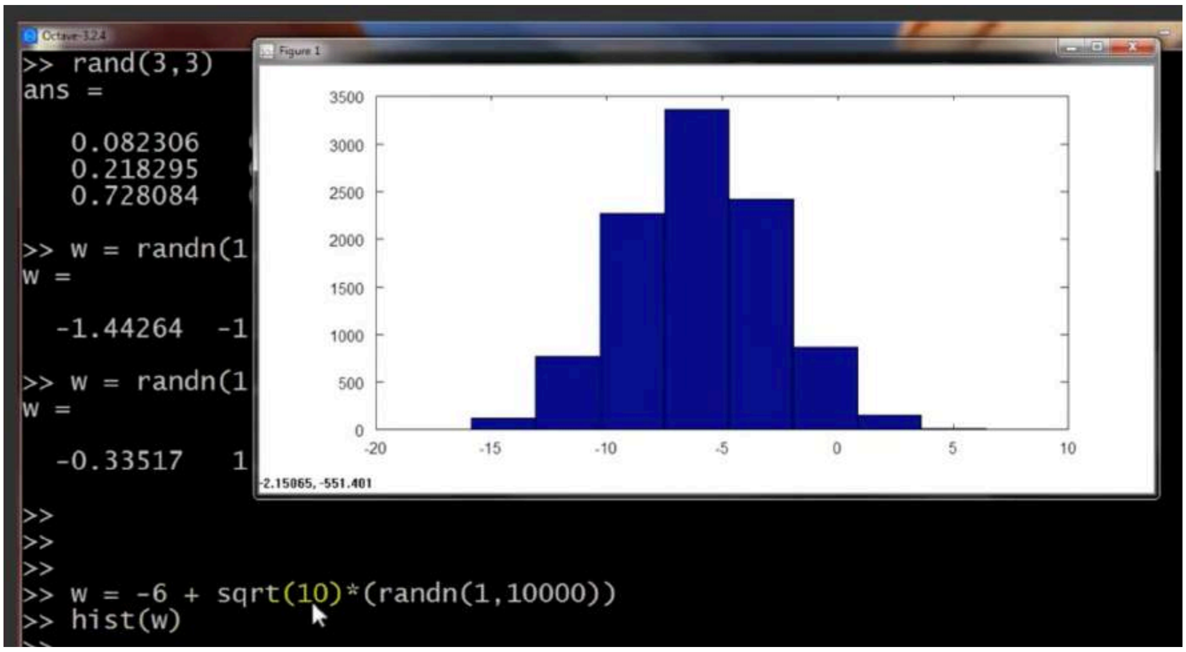


```
>> rand(3,3)
ans =

   0.390426   0.264057   0.683559
   0.041555   0.314703   0.506769
   0.521893   0.739979   0.387001

>> rand(3,3)
ans =

   0.467747   0.684916   0.346052
   0.022935   0.603373   0.307135
   0.212884   0.857236   0.456541

>> rand(3,3)
ans =

   0.082306   0.450805   0.307135
   0.218295   0.554723   0.819940
   0.728084   0.893041   0.312381
```

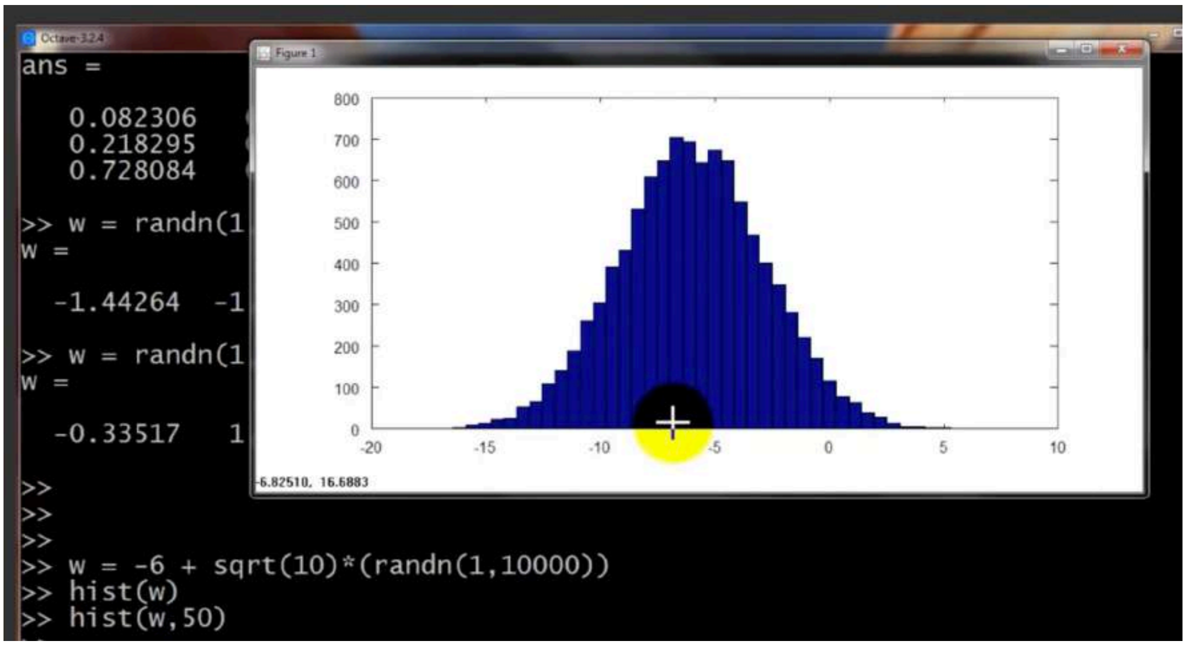


```
>> w = randn(1,3)
w =

  -1.44264  -1.27860  -0.69640

>> w = randn(1,3)
w =

  -0.33517   1.26847  -0.28211

>>
>>
>>
>> w = -6 + sqrt(10)*(randn(1,10000))
>>
```

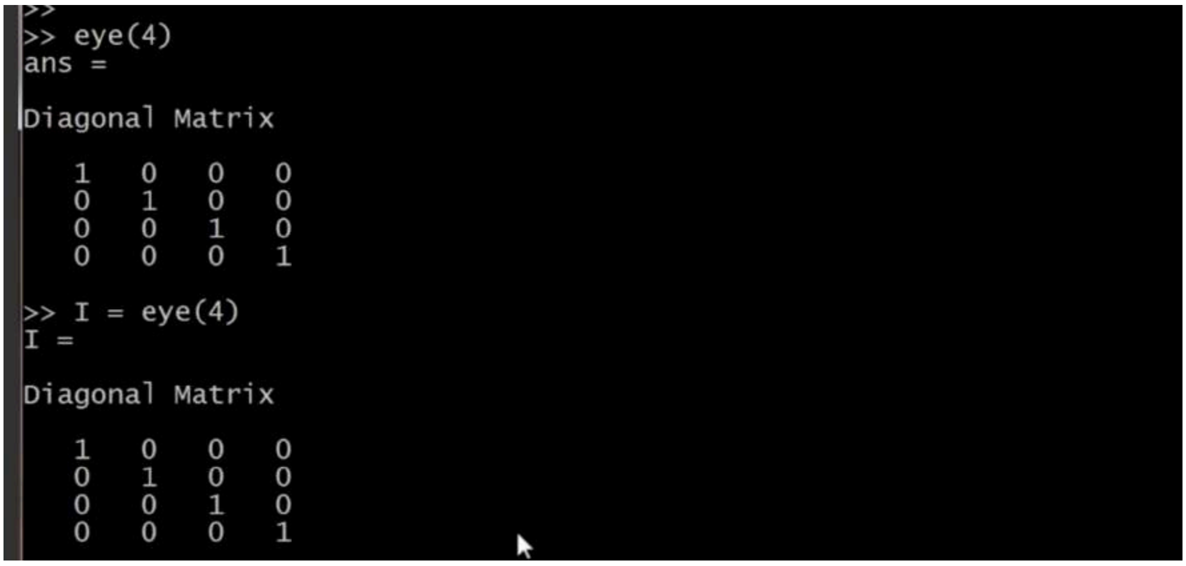
And if I now plot a histogram of W with a hist command, I can now. And Octave's print hist command, you know, takes a couple seconds to bring this up, but this is a histogram of my random variable for W. There was minus 6 plus zero ten times this Gaussian random variable. And I can plot a histogram with more buckets, with more bins, with say, 50 bins. And this is my histogram of a Gaussian with mean minus 6. Because I have a minus 6 there plus square root 10 times this. So the variance of this Gaussian random variable is 10 on the standard deviation is square root of 10, which is about what? Three point one. Finally, one special command for generator matrix, which is the I command. So I stands for this is maybe a pun on the word identity. It's server set eye 4. This is the 4 by 4 identity matrix. So I equals eye 4. This gives me a 4 by 4 identity matrix. And I equals eye 5, eye 6. That gives me a 6 by 6 identity matrix, i3 is the 3 by 3 identity matrix. Lastly, to wrap up this video, there's one more useful command. Which is the help command. So you can type help i and this brings up the help function for the identity matrix. Hit Q to quit. And you can also type help rand. Brings up documentation for the rand or the random number generation function. Or even help help, which shows you, you know help on the help function. So, those are the basic operations in Octave. And with this you should be able to generate a few matrices, multiply, add things. And use the basic operations in Octave. In the next video, I'd like to start talking about more sophisticated commands and how to use data around and start to process data in Octave.



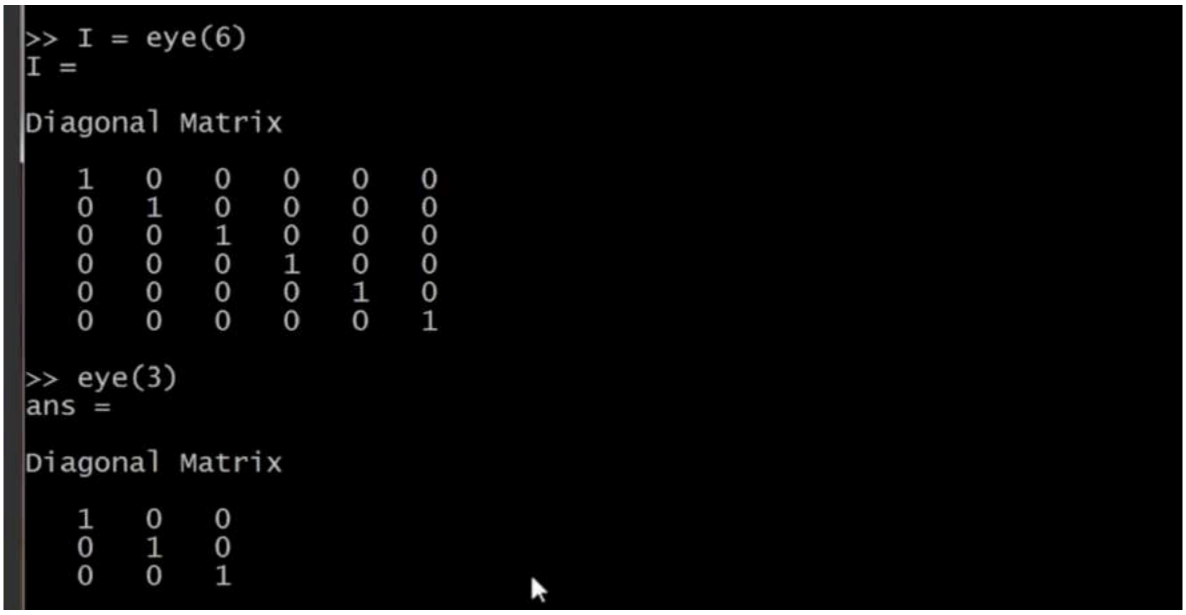
```
>> rand(3,3)
ans =

   0.082306
   0.218295
   0.728084

>> w = randn(1
w =

  -1.44264   -1

>> w = randn(1
w =

  -0.33517    1

>>
>>
>>
>> w = -6 + sqrt(10)*(randn(1,10000))
>> hist(w)
```



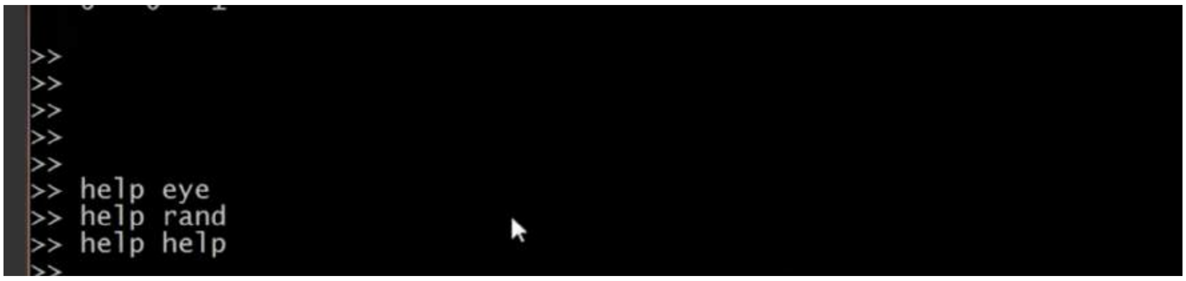
```
ans =

   0.082306
   0.218295
   0.728084

>> w = randn(1
w =

  -1.44264   -1

>> w = randn(1
w =

  -0.33517    1

>>
>>
>>
>> w = -6 + sqrt(10)*(randn(1,10000))
>> hist(w)
>> hist(w,50)
```



```
>> eye(4)
ans =

Diagonal Matrix

   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1

>> I = eye(4)
I =

Diagonal Matrix

   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```



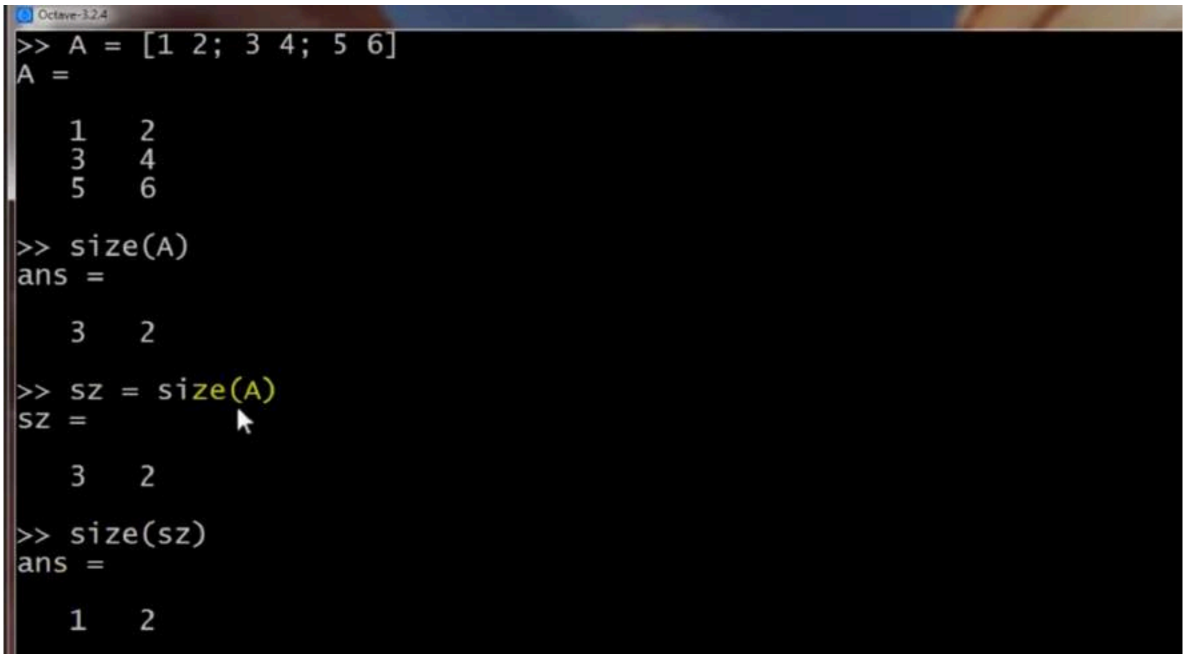
```
>> I = eye(6)
I =

Diagonal Matrix

   1   0   0   0   0   0
   0   1   0   0   0   0
   0   0   1   0   0   0
   0   0   0   1   0   0
   0   0   0   0   1   0
   0   0   0   0   0   1

>> eye(3)
ans =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1
```



```
>>
>>
>>
>>
>>
>> help eye
>> help rand
>> help help
```

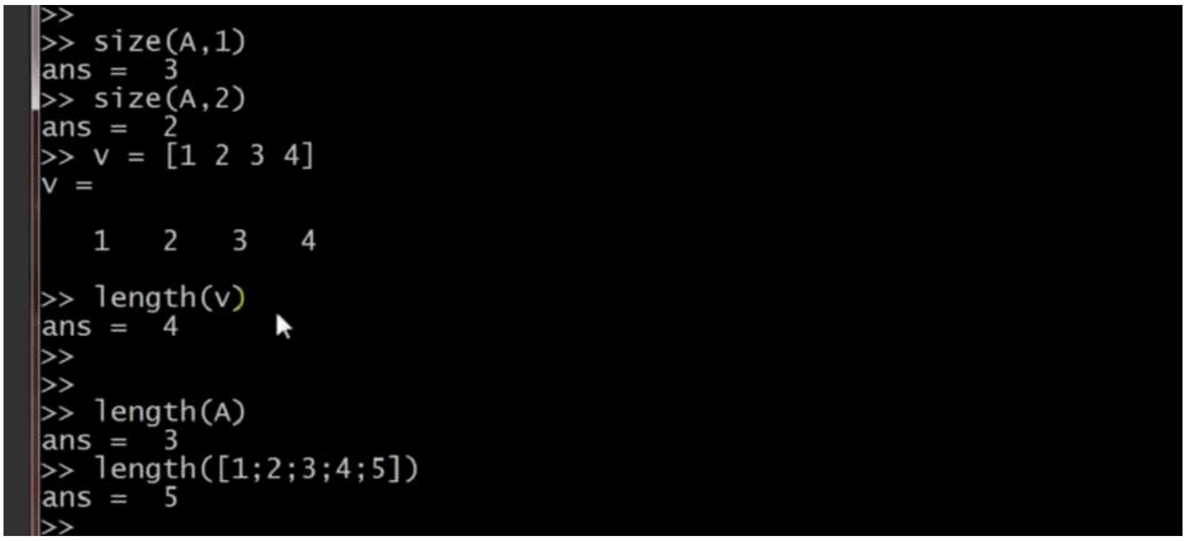
Moving Data Around

In this second tutorial video on Octave, I'd like to start to tell you how to move data around in Octave. So, if you have data for a machine learning problem, how do you load that data in Octave? How do you put it into matrix? How do you manipulate these matrices? How do you save the results? How do you move data around and operate with data? Here's my Octave window as before, picking up from where we left off in the last video. If I type A, that's the matrix so we generate it, right, with this command equals one, two, three, four, five, six, and this is a three by two matrix. The size command in Octave lets you, tells you what is the size of a matrix. So size A returns three, two. It turns out that this size command itself is actually returning a one by two matrix. So you can actually set SZ equals size of A and SZ is now a one by two matrix where the first element of this is three, and the second element of this is two. So, if you just type size of SZ. Does SZ is a one by two matrix whose two elements contain the dimensions of the matrix A.



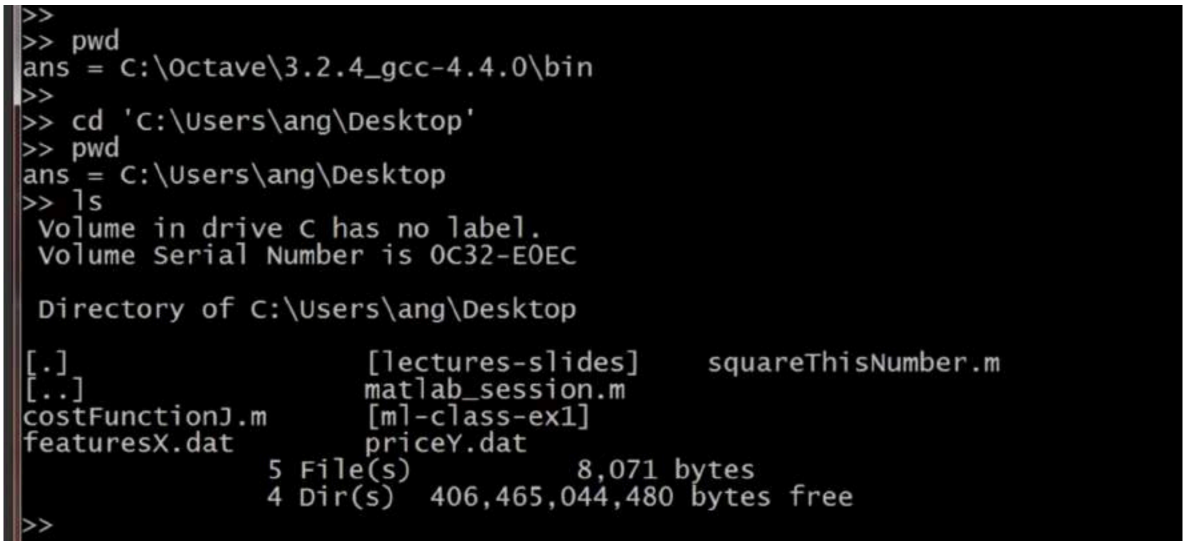
```
>> A = [1 2; 3 4; 5 6]
A =

   1   2
   3   4
   5   6

>> size(A)
ans =

   3   2

>> sz = size(A)
sz =

   3   2

>> size(sz)
ans =

   1   2
```

You can also type size A one to give you back the first dimension of A, size of the first dimension of A. So that's the number of rows and size A two to give you back two, which is the number of columns in the matrix A. If you have a vector V, so let's say V equals one, two, three, four, and you type length V. What this does is it gives you the size of the longest dimension. So you can also type length A and because A is a three by two matrix, the longer dimension is of size three, so this should print out three. But usually we apply length only to vectors. So you know, length one, two, three, four, five, rather than apply length to matrices because that's a little more confusing.



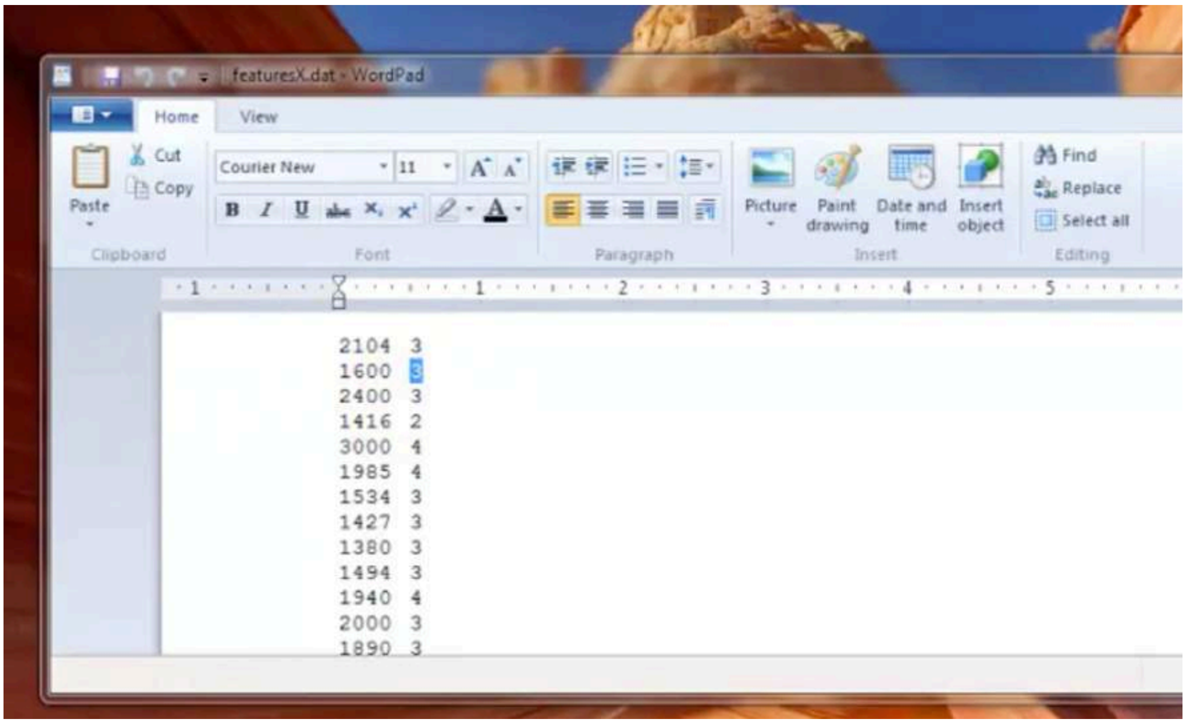
```
>>
>> size(A,1)
ans =  3
>> size(A,2)
ans =  2
>> v = [1 2 3 4]
v =

   1   2   3   4

>> length(v)
ans =  4
>>
>>
>> length(A)
ans =  3
>> length([1;2;3;4;5])
ans =  5
>>
```

Now, let's look at how the load data and find data on the file system. When we start an Octave we're usually, we're often in a path that is, you know, the location of where the Octave location is. So the PWD command shows the current directory, or the current path that Octave is in. So right now we're in this maybe somewhat off scale directory. The CD command stands for change directory, so I can go to C:/Users/Ang/Desktop, and now I'm in, you know, in my Desktop and if I type ls, ls is, it comes from a Unix or a Linux command. But, ls will list the directories on my desktop and so these are the files that are on my Desktop right now.

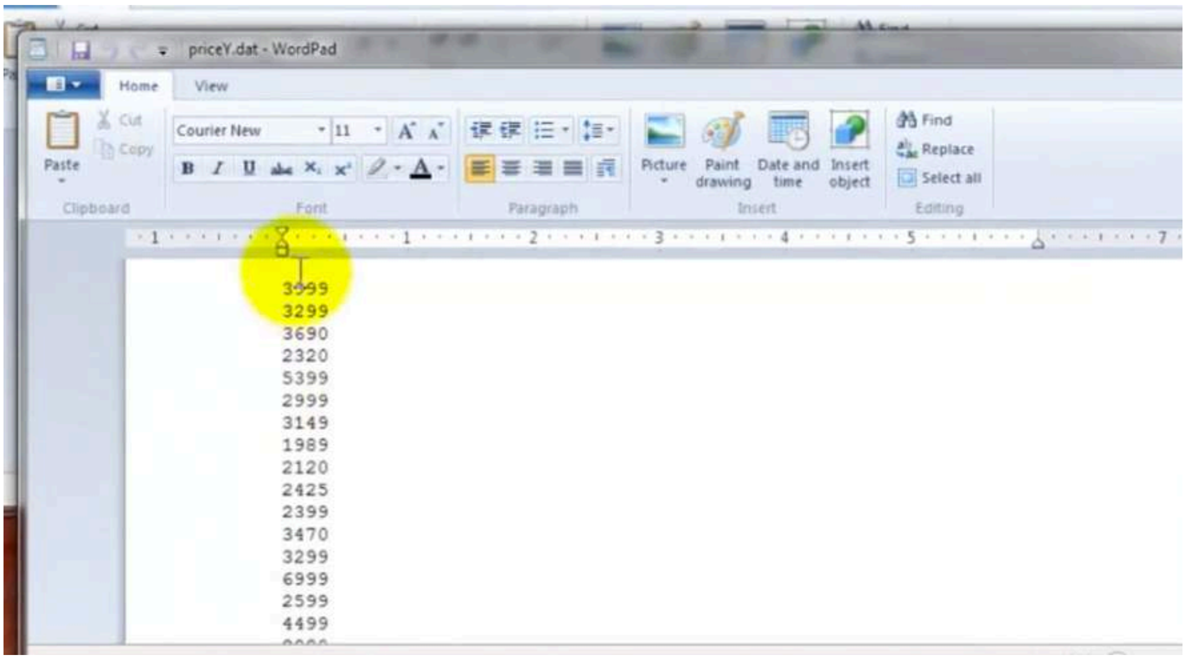


```
>>
>> pwd
ans = C:\Octave\3.2.4_gcc-4.4.0\bin
>>
>> cd 'C:\Users\ang\Desktop'
>> pwd
ans = C:\Users\ang\Desktop
>> ls
 Volume in drive C has no label.
 Volume Serial Number is 0C32-E0EC

 Directory of C:\Users\ang\Desktop

[.]                    [lectures-slides]    squareThisNumber.m
[..]                   matlab_session.m
costFunctionJ.m        [ml-class-ex1]
featuresX.dat          priceY.dat
               5 File(s)          8,071 bytes
               4 Dir(s)  406,465,044,480 bytes free
>>
```

In fact, on my desktop are two files: Features X and Price Y that's maybe come from a machine learning problem I want to solve. So, here's my desktop. Here's Features X, and Features X is this window, excuse me, is this file with two columns of data. This is actually my housing prices data. So I think, you know, I think I have forty-seven rows in this data set. And so the first house has size two hundred four square feet, has three bedrooms; second house has sixteen hundred square feet, has three bedrooms; and so on. And Price Y is this file that has the prices of the data in my training set. So, Features X and Price Y are just text files with my data.

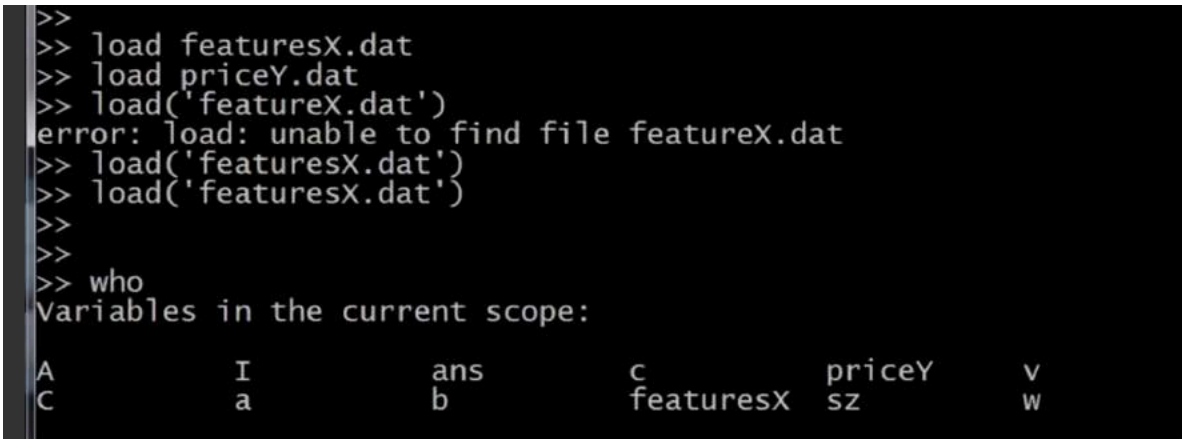


```
2104  3
1600  3
2400  3
1416  2
3000  4
1985  4
1534  3
1427  3
1380  3
1494  3
1940  4
2000  3
1890  3
```

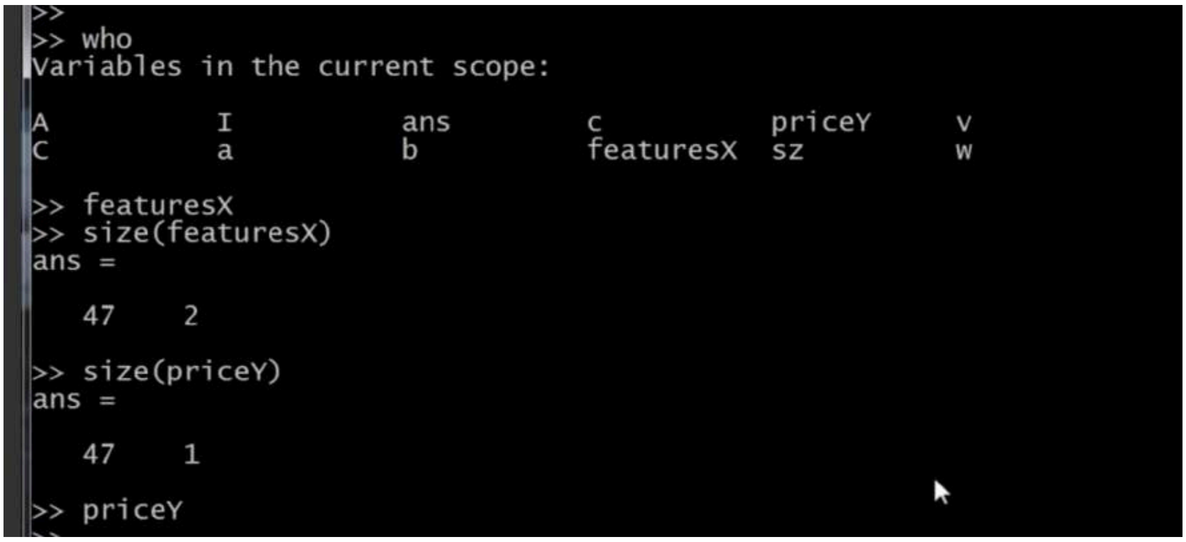


```
3999
3299
3690
2320
5399
2999
3149
1989
2120
2425
2399
3470
3299
6999
2599
4499
```

How do I load this data into Octave? Well, I just type the command load Features X dot dat and if I do that, I load the Features X and can load Price Y dot dat. And by the way, there are multiple ways to do this. This command if you put Features X dot dat on that in strings and load it like so. This is a typo there. This is an equivalent command. So you can, this way I'm just putting the file name of the string in the founding in a string and in an Octave use single quotes to represent strings, like so. So that's a string, and we can load the file whose name is given by that string. Now the WHO command now shows me what variables I have in my Octave workspace. So Who shows me whether the variables that Octave has in memory currently. Features X and Price Y are among them, as well as the variables that, you know, we created earlier in this session. So I can type Features X to display features X. And there's my data. And I can type size features X and that's my 47 by two matrix. And some of these size, press Y, that gives me my 47 by one vector. This is a 47 dimensional vector. This is all common vector that has all the prices Y in my training set.

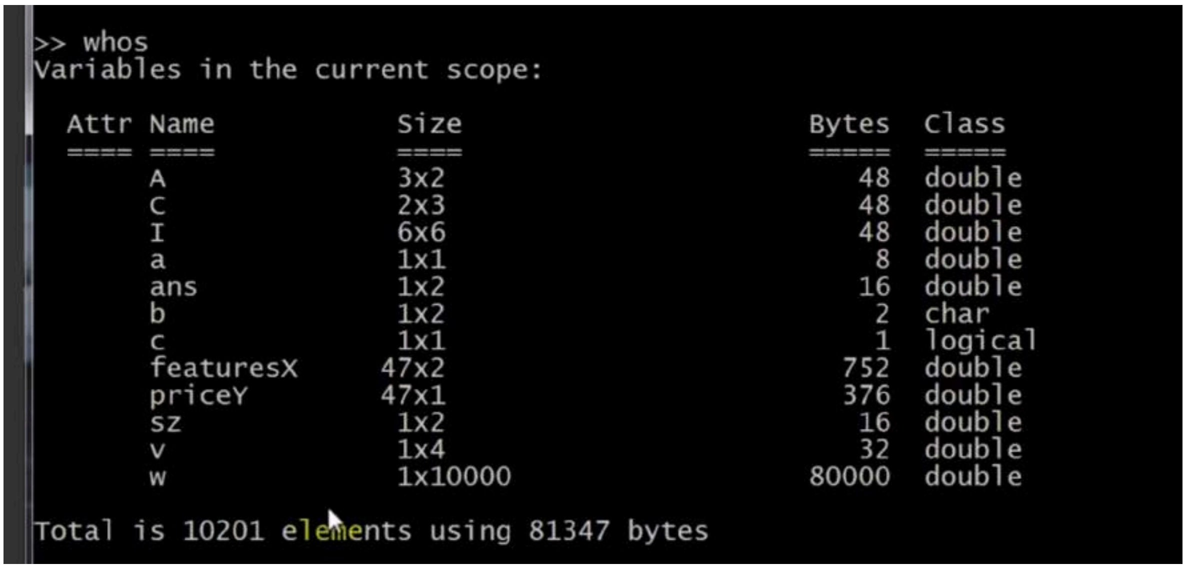


```
>>
>> load featuresX.dat
>> load priceY.dat
>> load('featureX.dat')
error: load: unable to find file featureX.dat
>> load('featuresX.dat')
>> load('featuresX.dat')
>>
>>
>> who
Variables in the current scope:

A          I          ans        c          priceY     v
C          a          b          featuresX  sz         w
```



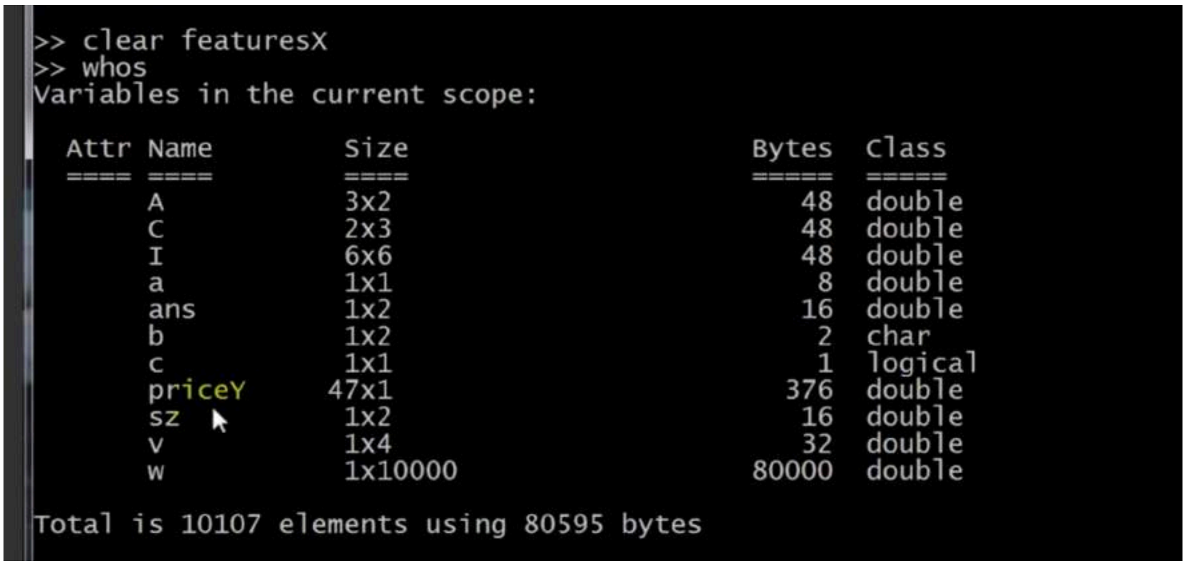
```
>>
>> who
Variables in the current scope:

A          I          ans        c          priceY     v
C          a          b          featuresX  sz         w

>> featuresX
>> size(featuresX)
ans =

   47    2

>> size(priceY)
ans =

   47    1

>> priceY
```

Now the who function shows you one of the variables that, in the current workspace. There's also the who S variable that gives you the detailed view. And so this also, with an S at the end this also lists my variables except that it now lists the sizes as well. So A is a three by two matrix and features X as a 47 by 2 matrix. Price Y is a 47 by one matrix. Meaning this is just a vector. And it shows, you know, how many bytes of memory it's taking up. As well as what type of data this is. Double means double position floating point so that just means that these are real values, the floating point numbers.



```
>> whos
Variables in the current scope:

  Attr Name            Size                     Bytes  Class
  ==== ====            ====                     =====  =====
       A               3x2                         48  double
       C               2x3                         48  double
       I               6x6                         48  double
       a               1x1                          8  double
       ans             1x2                         16  double
       b               1x2                          2  char
       c               1x1                          1  logical
       featuresX      47x2                        752  double
       priceY         47x1                        376  double
       sz              1x2                         16  double
       v               1x4                         32  double
       w               1x10000                  80000  double

Total is 10201 elements using 81347 bytes
```

Now if you want to get rid of a variable you can use the clear command. So clear features X and type whose again. You notice that the features X variable has now disappeared.



```
>> clear featuresX
>> whos
Variables in the current scope:

  Attr Name         Size                     Bytes  Class
  ==== ====         ====                     =====  =====
       A            3x2                         48  double
       C            2x3                         48  double
       I            6x6                         48  double
       a            1x1                          8  double
       ans          1x2                         16  double
       b            1x2                          2  char
       c            1x1                          1  logical
       priceY      47x1                        376  double
       sz           1x2                         16  double
       v            1x4                         32  double
       w            1x10000                  80000  double

Total is 10107 elements using 80595 bytes
```

(Continued…….)