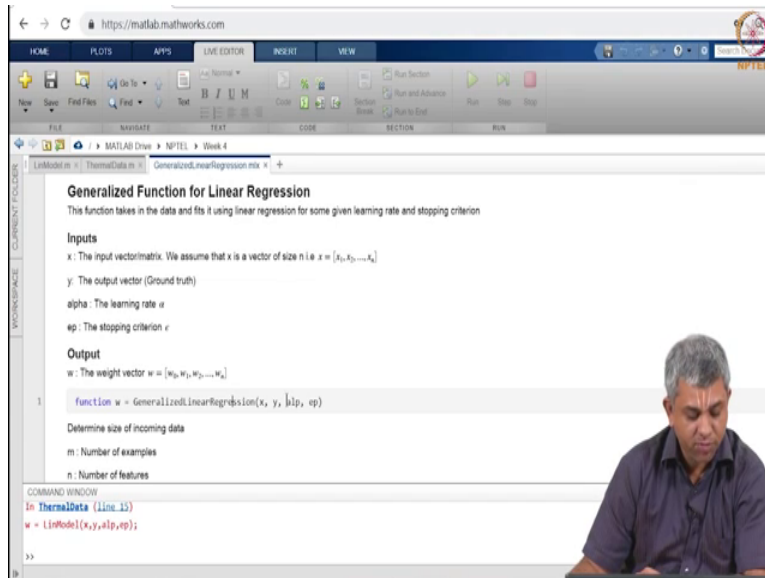


**Machine Learning for Engineering and Science Applications**  
**Professor Dr. Balaji Srinivasan**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology, Madras**  
**Generalized Functions for Linear Regression**

(Refer Slide Time: 00:15)



```
function w = GeneralizedLinearRegression(x, y, alpha, epsilon)
    Determine size of incoming data
    m = Number of examples
    n = Number of features
```

```
In ThermalData (line 15)
w = LinModel(x,y,alpha,epsilon);
```

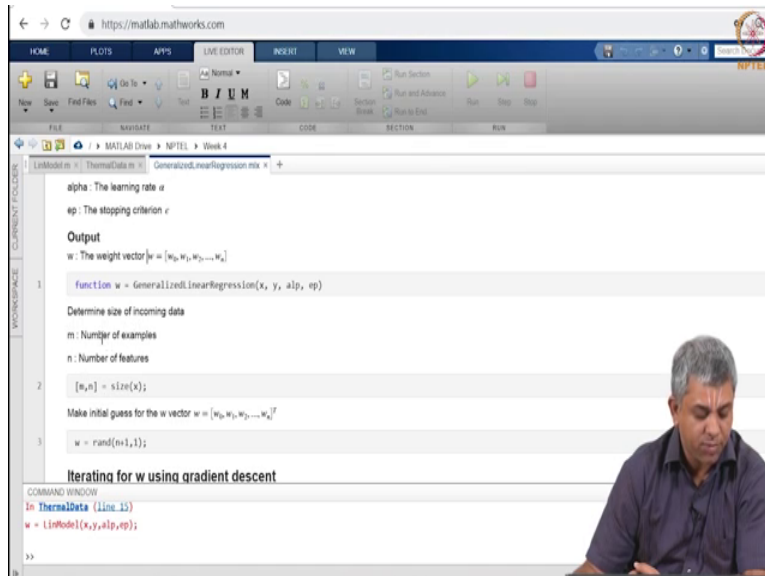
Welcome back in the last video I has shown you how to write a generalized linear regression routine we had decided we have discussed how to take  $x$  as a vector with a scalar  $y$  as output and how to map one to the other using a linear model. So in this short video I would like to show you how to code this up I am going to just briefly go over it we will leave the responsibility of reading the code and understanding it further most part this is a simple a MATLAB routine which will be available in you with you for the in the NPTEL website in the week 4 portions. So what I have written here is a function.

So this function which we called generalized linear regression a simpler version of this without all the comments is available in lin model it is exactly the same as this code both this codes are available to you on your NPTEL website. Once again I am repeating this but we are welcome to write some such thing for yourself in any other language that you are comfortable with. So this is just for your understanding so what we have here is a regression model which takes in  $x$  the input

vector  $y$ , the corresponding output vector remember you have a whole bunch of examples which is why I have written input matrix here because  $x$  for each example is actually a vector ok.

As we discussed in the previous video  $x$  could itself have  $x_1, x_2, x_3$  etc depending on the number of features or attributes that you have. Alpha is the learning rate that you think will be good for this problem and epsilon is the stopping criteria for the problem.

(Refer Slide Time: 02:05)



The screenshot shows the MATLAB Live Editor interface. The main window displays a function named `GeneralizedLinearRegression` with the following code:

```
alpha: The learning rate  $\alpha$ 
ep: The stopping criterion  $\epsilon$ 

Output
w: The weight vector  $w = [w_0, w_1, w_2, \dots, w_n]$ 

1 function w = GeneralizedLinearRegression(x, y, alp, ep)
   Determine size of incoming data
   m: Number of examples
   n: Number of features
2 [m,n] = size(x);
   Make initial guess for the w vector  $w = [w_0, w_1, w_2, \dots, w_n]^T$ 
3 w = rand([n+1,1]);

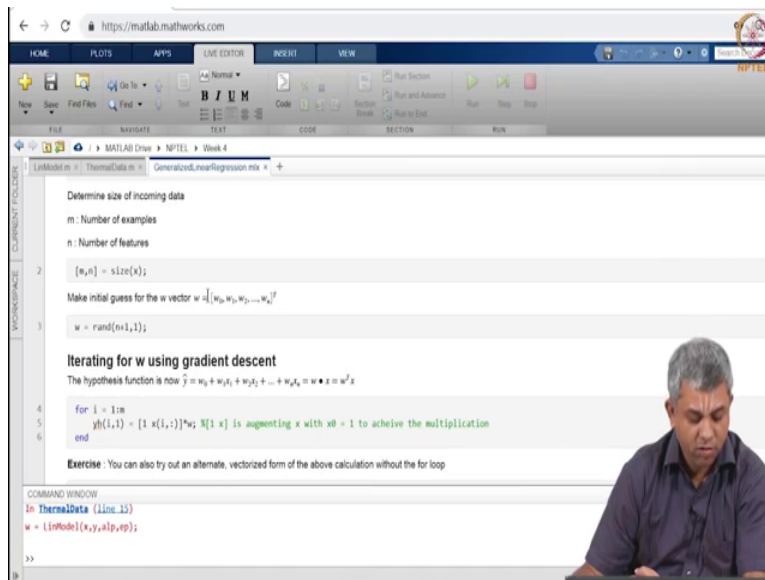
Iterating for w using gradient descent

COMMAND WINDOW
In ThermalData (Line 15)
w = LinearModel(x,y,alp,ep);
>>
```

A small inset image of a man with grey hair, wearing a dark blue shirt, is visible in the bottom right corner of the screenshot, looking down at the code.

The output which this function gives out is the whole weight vector and notice as we had discussed in the last video I am actually including the bias term  $w_0$  which some people call  $B$  and including that in the weight vector therefore the size of the weight vector has to be this  $n$  features plus the one bias ok. So the first thing that we do since we have not given it explicitly this is your choice MATLAB has an easy way for you to determine the size of the incoming data, this might or might not be available in other programming languages. So we have used this feature easily so we find out the number of examples  $m$  and the number of features  $n$  simply by looking at the size of  $x$ .

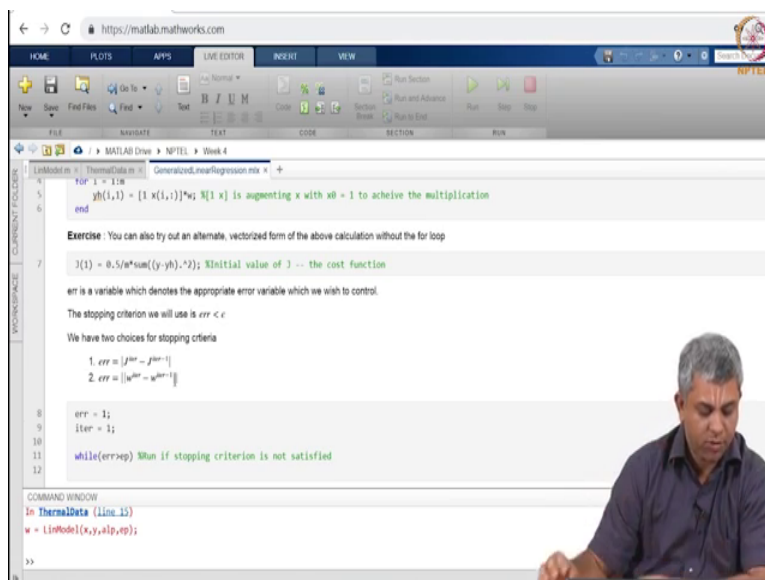
(Refer Slide Time: 02:50)



```
https://matlab.mathworks.com
HOME PLOTS APPS LIVE EDITOR INDEX VIEW
New Save Find Files Go To Find Code Run Section Run and Advance Run and End Run Stop
FILE NAVIGATE TEXT CODE SECTION RUN
MATLAB Drive > NPTEL > Week 4
LinearModel.m ThermalData.m GeneralizedLinearRegression.m
Determine size of incoming data
m : Number of examples
n : Number of features
2 [w,n] = size(x);
Make initial guess for the w vector w = [w_0, w_1, w_2, ..., w_n]^T
3 w = rand(n+1,1);
Iterating for w using gradient descent
The hypothesis function is now  $\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = w \cdot x = w^T x$ 
4 for i = 1:m
5   yh(i,1) = [1 x(i,:)]*w; % [1 x] is augmenting x with w0 = 1 to achieve the multiplication
6 end
Exercise : You can also try out an alternate, vectorized form of the above calculation without the for loop
COMMAND WINDOW
In ThermalData (line 15)
w = LinearModel(x,y,alp,eps);
>>
```

We also make an initial guess notice its n plus 1 is the size of vector because they include  $w_0$  also and then this is all of it is the same as before.

(Refer Slide Time: 02:56)

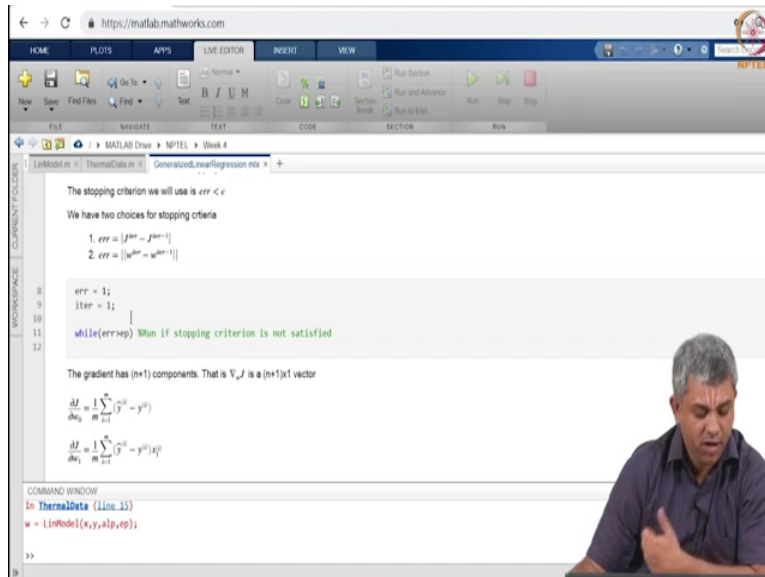


```
https://matlab.mathworks.com
HOME PLOTS APPS LIVE EDITOR INDEX VIEW
New Save Find Files Go To Find Code Run Section Run and Advance Run and End Run Stop
FILE NAVIGATE TEXT CODE SECTION RUN
MATLAB Drive > NPTEL > Week 4
LinearModel.m ThermalData.m GeneralizedLinearRegression.m
4 for i = 1:m
5   yh(i,1) = [1 x(i,:)]*w; % [1 x] is augmenting x with w0 = 1 to achieve the multiplication
6 end
Exercise : You can also try out an alternate, vectorized form of the above calculation without the for loop
7 J(i) = 0.5*m*sum((y-yh).^2); %Initial value of J -- the cost function
err is a variable which denotes the appropriate error variable which we wish to control.
The stopping criterion we will use is  $err < \epsilon$ 
We have two choices for stopping criteria
1.  $err = |J^{iter} - J^{iter-1}|$ 
2.  $err = ||w^{iter} - w^{iter-1}||$ 
8 err = J;
9 iter = 1;
10 while(err>eps) %Run if stopping criterion is not satisfied
11
12
COMMAND WINDOW
In ThermalData (line 15)
w = LinearModel(x,y,alp,eps);
>>
```

We actually iterate for  $w$  using gradient descent notice now that the hypothesis function is simply  $w$  not plus  $w_1 x$  the  $w$  knot plus  $w_1 x_1$  plus  $w_2 x_2$  upto  $w_n x_n$  which can be written as  $w$  transpose  $x$  in case  $w$  is the modified  $w$  including  $w$  knot also and assuming that  $x$  knot is equal to 1 ok. So we find out the hypothesis function we will notice a 1 sitting here this 1 is sitting because I am writing  $x_k$  not explicitly  $x$  is simply  $x_1$  through  $x_n$  ok.

There are many ways of writing it I have left that as an exercise to you I have written a slightly inefficient version but you can write more efficient versions ok. Once again like before you have several choices for stopping criteria either you can choose the difference between the current value of the loss function and the previous value of the loss function or you can find out how much does the current  $w$  differ from the previous  $w$ , ok.

(Refer Slide Time: 04:02)



```
https://matlab.mathworks.com
HOME PLOTS APPS LIVE EDITOR INSIGHT VIEW
New Save Find Files Delete Find Text Bold Italic Underline Code Run Section Run and Advance Run Stop
FILE NAVIGATE TEXT CODE SECTION RUN
LHMModel.m ThermalData.m Generation.m,meanRegression.m
The stopping criterion we will use is  $err < \epsilon$ 
We have two choices for stopping criteria
1.  $err = |J^{(m)} - J^{(m+1)}|$ 
2.  $err = ||w^{(m)} - w^{(m+1)}||$ 
8 err = 1;
9 iter = 1;
10 |
11 while(err>ep) %Run if stopping criterion is not satisfied
12
The gradient has (n+1) components. That is  $\nabla J$  is a (n+1)x1 vector

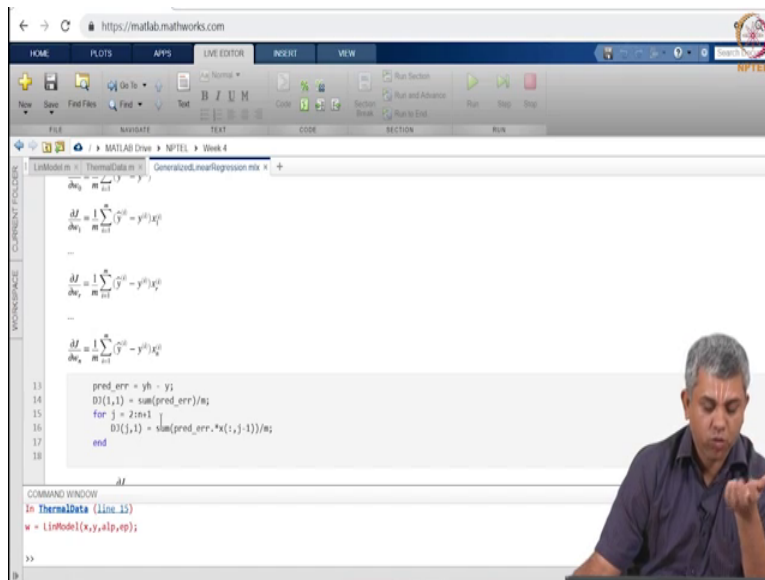
$$\frac{\partial J}{\partial w_0} = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - y^{(i)})$$


$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - y^{(i)}) x^{(i)}$$

COMMAND WINDOW
In ThermalData (line 11)
w = linModel(x,y,alp,ep);
>>
```

In either case we simply call this stopping criterion as error and the moment the till it is greater than epsilon keep on running.

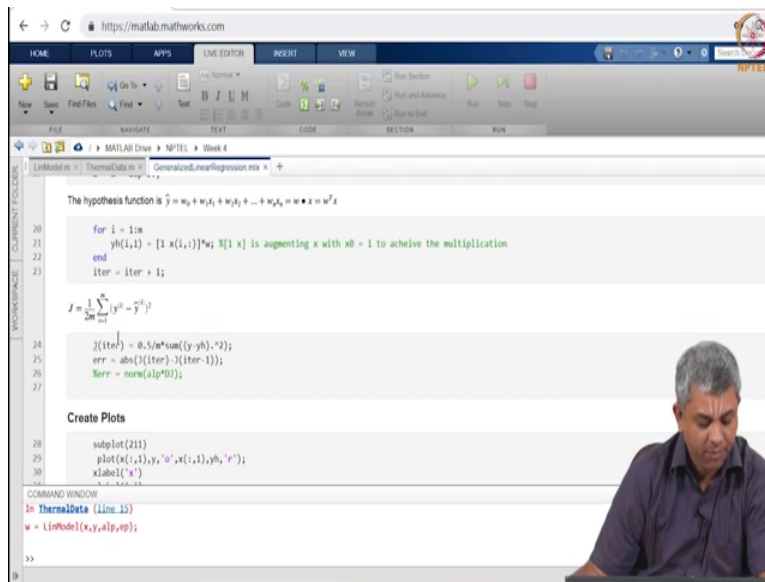
(Refer Slide Time: 04:12)



Our main task of course is to find out the gradients of  $J$  with respect to the weight vector  $w$ , I have written the formulation here we had also derived it twice in the previous videos and you notice prediction error multiplied by the corresponding feature component ok.

So if you look at  $\frac{\partial J}{\partial w}$  it is going to be 1 by  $m$  times summation of the prediction error  $y$  ha minus  $\hat{y}$  multiplied by  $x$  ok which is what I have written here,  $D_j$  is what denotes  $\frac{\partial J}{\partial w}$  notice that  $D_j$  has  $n$  plus components starting from the first component which will correspond to  $w_0$  and will go upto  $n$  plus 1 which will correspond to  $w$ .

(Refer Slide Time: 04:58)



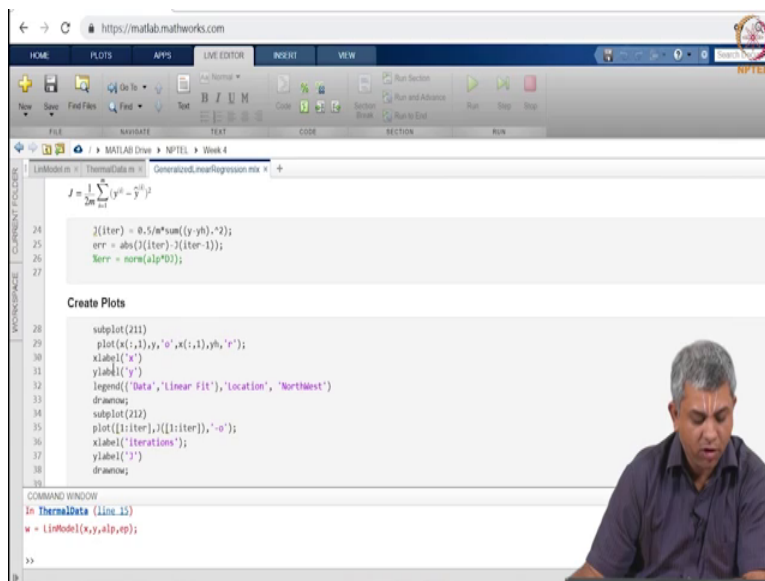
```
https://matlab.mathworks.com
HOME PLOTS APPS LIVE EDITOR INSERT VIEW
New Save Find Files Find Go To Text Code Section Break Run and Advance Run Step Stop
FILE NAVIGATE TEXT CODE SECTION RUN
MATLAB Drive > NPTEL > Week 4
ThermalData.m ThermalData.m GeneralizedLinearRegression.m
The hypothesis function is  $\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n = w \cdot x = w^T x$ 
20 for i = 1:m
21     yh(1,i) = [1 x(1,i)]*w; % [1 x] is augmenting x with x0 = 1 to achieve the multiplication
22 end
23 iter = iter + 1;

$$J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

24 J(iter) = 0.5/m*sum((y-yh).^2);
25 err = abs(J(iter)-J(iter-1));
26 %err = norm(alp*0);
27
Create Plots
28 subplot(211)
29 plot(x(:,1),y,'o','x(:,1),yh','r');
30 xlabel('x')
COMMAND WINDOW
In ThermalData (line 15)
w = LinModel(x,y,alp,eps);
>>
```

So we take that we calculate the change I w, w is ofcourse w minus alpha times grad j ok. Finally we calculate what the hypothesis is and this lets just calculate the j because y minus y hat square averaged over all the other all the examples actually gives you j.

(Refer Slide Time: 05:20)



```
https://matlab.mathworks.com
HOME PLOTS APPS LIVE EDITOR INSERT VIEW
New Save Find Files Find Go To Text Code Section Break Run and Advance Run Step Stop
FILE NAVIGATE TEXT CODE SECTION RUN
MATLAB Drive > NPTEL > Week 4
ThermalData.m ThermalData.m GeneralizedLinearRegression.m

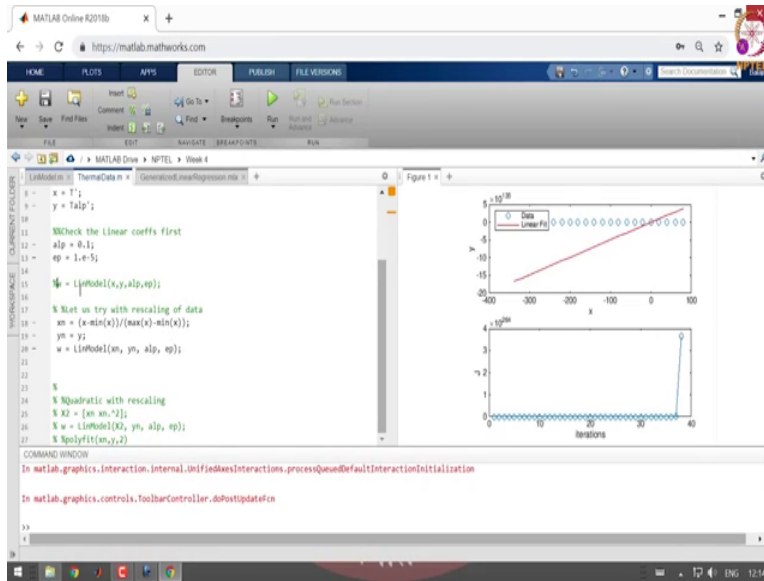
$$J = \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

24 J(iter) = 0.5/m*sum((y-yh).^2);
25 err = abs(J(iter)-J(iter-1));
26 %err = norm(alp*0);
27
Create Plots
28 subplot(211)
29 plot(x(:,1),y,'o','x(:,1),yh','r');
30 xlabel('x')
31 ylabel('y')
32 legend(['Data','Linear Fit'],'location','NorthWest')
33 drawnow;
34 subplot(212)
35 plot([1:iter],[J(1:iter)],'-o');
36 xlabel('Iterations');
37 ylabel('J')
38 drawnow;
COMMAND WINDOW
In ThermalData (line 15)
w = LinModel(x,y,alp,eps);
>>
```

I have also included some plots just for similarity from before and we can try running it in order to see how this performs. Now the important thing here is this is really general you can take any number of examples any number of m and also you can choose any number of features ok so the code is supposed to work and as we discussed in the previous video this lets us use not only

linear regression but it also lets us use polynomial regression because all we need to do is to change  $w_1$  to  $x$  power 1  $w_2$  to  $x$  power 2  $x$  square and  $w_n$  multiplying  $x$  power  $n$  ok. So if  $x$  the incoming vector is basically  $x_1, x_2, x_3, x_n$  you can simply use that as the features you as the polynomials and that will work ok.

(Refer Slide Time: 06:11)



So let us now try using our original data and see if we can now use our generalized code, generalized linear regression code in order to make linear quadratic and cubic predictions. You will see one small surprise here which will lead us to a small modification for what we want to do for linear regression ok. So as before  $t$  was the initial data and the  $\alpha$  is  $y$  ok notice that  $\alpha$  is multiplied by  $10$  power minus  $6$  as the expansion coefficient ok. So initially we define  $x$  as  $t$  and  $y$  as  $\alpha$  we choose a learning rate of  $1$  we choose an epsilon of  $10$  power minus  $5$  and we will try learning our generalized linear regression code and see what happens. So when we run please notice what is happening here you can see this numbers growing larger and larger in fact  $j$  is rapidly increasing and  $j$  has to now reach  $10$  power  $247$  and  $y$  has also reached  $10$  power  $128$  for the prediction because we are not getting convergences we are actually getting divergences.

Now you might decide that this is because of a large  $\alpha$  and try and reduce it. So let us say we make  $\alpha$   $0.1$  instead of  $1$  and we try running it again and you will see that the situation has not really improved it is still blowing up ok you have still got high values of  $j$  and you have got high values of  $y$ . Now why does it happen? This happens due to a certain reason that is because the  $t$

that you have here or the  $x$  that you have here if I write  $x$  let us try writing  $x$  here please notice  $x$  is going from 80 to minus 340 ofcourse our  $y$ 's are  $10^6$  minus 5 times all this values. So your coefficients that need to come so that this  $x$  can be mapped with this  $y$  are extremely small ok.

All this problems are essentially what are called normalization problems ok. For example let us say you are matching the area of a house to its price an example that I have used before ok. So in what units will you give the area? You could make the area in you know square foot which is normal thing or you can make it in square meters you could make it in square kilometers in which case your input vector will look really small or you could make it square centimeters etc.

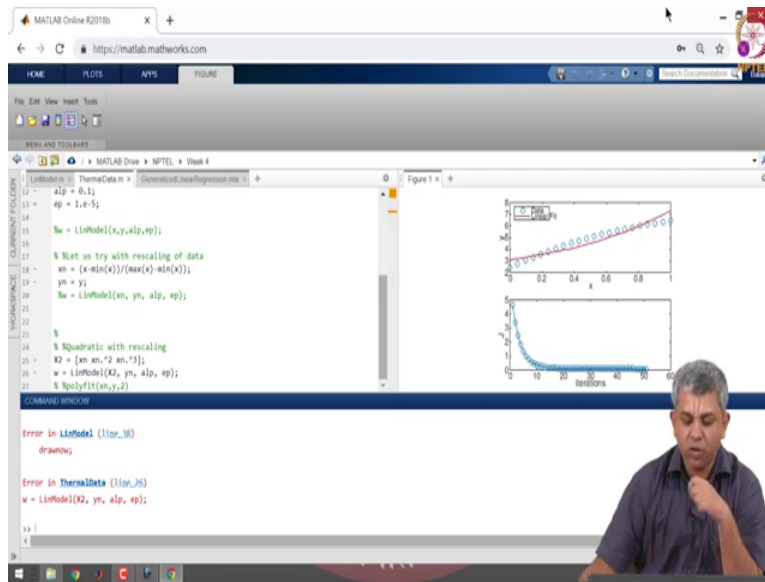
Similarly suppose you are mapping the height of a person to his or her weight and that is the regression problem that we wish to do in what you unit should you represent height? Should we represent it in micrometer, should we have represented in meter which seems reasonable to us or should be represent it in foot etc if you represent it in kilometers your numbers your input vector will look really small and your weights will change appropriately. So we tend to try and use units where you normalize  $x$  so that typically it varies to the order of 1 ok. So that turns out to be the simplest thing to do you can make it so that it varies from lest say minus 5 to 5 or from 0 to 1 which is probably the easiest.

So the choice that we will do right now and this step is called normalization or re-scaling is to rescale the data so that I have a new  $x$  I will also rescale  $y$  you know remember we have all this numbers and I was multiplying by  $10^6$  which is arbitrarily as far as the code is concerned I will comment this out so that  $y$  is now simply this numbers  $x$  is rescaled the way this rescaling was done is  $x$  goes to  $x - \text{minimum of } x$  by  $\text{maximum} - \text{minimum}$  okay when you do this this  $x$  will actually get rescaled so that it is only between 0 and 1. So now that we have done the rescaling let us try and see what  $x$  looks like.

So we have new  $x$  here if I write it out you will notice that it goes between 0 and 1 ok so you can see that the maximum is 1 and the minimum is 0 this corresponds to the following  $x$  which was 80 to minus 340 all we have done is  $x$  has been rescaled the minimum has been subtracted out and has been rescaled by the range. Ok so now  $x$  goes from 0 to 1 or 1 to 0 ok and now we can try and see what happens when we continue our code ok.



(Refer Slide Time: 11:13)



For the same alpha you will now see that the linear fit starts working ok. So there is a drastic difference when we didn't rescale it was going completely wrong because the values of x were high and the corresponding value of y hat was also high.

So you can see that certain things can actually make a great difference as far as training goes, training means finding out the coefficients. So you will see that the fit is now working and the only change we made was we rescaled the data ok we rescale the data instead of having original x now you have x between 0 and 1 ok. So this trick is an important trick infact it has been generalized to something really big called batch norm which we will see later on in the course ok. So I will stop this stimulation. Now another thing you can do is with the same code we can now try and get quadratic. So notice this we keep the same x n we keep the same y n all I change is now I change my feature vectors and I say that the input is not only x but it is x as well as x square.

Now our code is written such that the moment I give it one extra x or an extra feature ok it will start reading more features and it will fit a bigger model ok so this is the trick that we use the moment I gave x and x square obviously the code doesn't know you have given x square as the second variable all it knows is x1 and x2 so the moment it sees x1 and x2 it will say that my model is now no longer  $w_0 + w_1 x_1$  but it is  $w_0 + w_1 x_1 + w_2 x_2$  which says at purposes because x2 is now x square ok. So let us see that now just to show you what happens I

will run this again so if you come here you will now see that the number of features with the code is recognizing this two and you will also notice that  $w$  is now a 3 by 1 vector and this is the initial guess it has given an initial (get of) guess of 0.8 for  $w$  knot 0.14 for  $w_1$  and 0.42 for  $w_2$  and this (( ))(13:32) are purposes ok.

So will continue here and you will see that now it is trying to fit it is trying to fit a curved line a quadratic line to this data I will let you run this on your own and it is not very hard t change this into a cubic fit because all you need to do is to add this extra term ok and we can now start from scratch and run this and it will try and fit a cubic plot you can see the this is slightly more curved and I would encourage you to play around with this code or write one on your own and see what's sort of fits you can get for this kind of data ok you can ofcourse try it for any data. So what we have seen in this video is that rescaling helps you and that you can actually fit with the same code you can fit linear, a quadratic or a polynomial depending on what sort of input vector you give it. I will write down what is needed to do this once more.

(Refer Slide Time: 14:46)

Normalization / Rescaling

Rescale  $\rightarrow \tilde{X} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$  New input vector  
 $\tilde{x} \in [0, 1]$  Batch Norm

Alternate:  $\tilde{x} = \frac{x - \mu}{\sigma}$   $\leftarrow$  Mean  $\rightarrow$  Normalization  
 $\sigma$   $\leftarrow$  Standard deviation

So if we look at normalization or rescaling what we did was  $x$  became  $x$  minus  $x$  min by  $x$  max minus  $x$  min and lets call this  $x$  tilda for our purposes and this is our new input vector what this does ofcourse is  $x$  tilda now will vary between 0 and 1 there are other alternatives for rescaling this is to say  $x$  tilda is equal to  $x$  minus  $\mu$  by  $\sigma$  where  $\mu$  is the mean of the data and  $\sigma$  is the standard deviation of data ok. Typically this is called normalization and this doesn't ensure

that you are going to lie between 0 and 1, if you usually go between negative 3 to plus 3 or somewhere in that range, this is simple rescaling ok.

So you can use one or the other normalization is used in with great effect in something called batch norm. Batch norm is very-very effectively used in several deep neural networks and you will see this a little bit later.