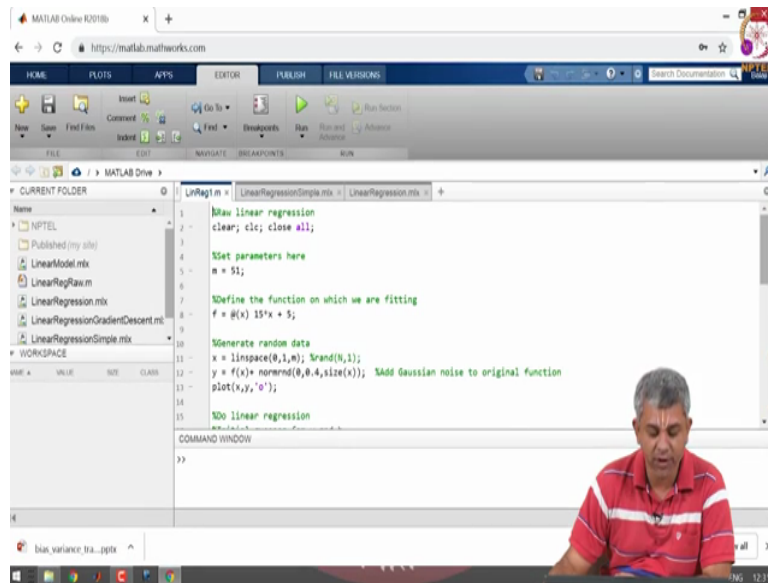


Machine Learning for Engineering and Science Applications
Professor Dr. Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology, Madras
Coding Linear Regression

(Refer Slide Time: 0:14)



In the previous video we saw how to do linear regression using gradient decent, in this video I will show you a simple example of how to do that an actual code, we will actually switch between two codes back and forth the same code one of them has more textual or text comments and another one is pure code as you would likely see. In case some of you find the fonts of this a little bit difficult to read, you can also follow along with the code which has been given in your on the website on NPTEL for week 4, so you will find these codes on the website for NPTEL also.

(Refer Slide Time: 1:04)

The image displays two screenshots of a MATLAB script editor. The first screenshot shows the following code:

```
1 alp = 0.1; %Learning rate
2 ep = 1.e-6; %Stopping tolerance
```

Below the code, there are sections for 'Set Parameters' and 'Data Generation' with explanatory text. The second screenshot shows the following code:

```
3 m = 51;
4 x = linspace(0,1,m);
5 y = 15*x + 5 + normrnd(0,0.4,size(x)); %Add Gaussian noise to original function
```

Below the code, there is a section for 'Visualize the raw Data' with the instruction 'Plot the data'. A small video inset of a man in a red shirt is visible in the bottom right corner of both screenshots.

So what we will be doing here is trying to setup a simple case, so I will show you this case here so we will try and setup a simple case for a linear regression, we will create synthetic data the previous example that I showed you was actual data of temperature versus alpha the thermal coefficient, but in this case we will create some fake x and y data I will also tell you how we are going to create that data just so that we can check you know how to do a linear regression with some simple data.

So if you remember the process that we had started for the course we first wanted to set the parameters, so alpha for example is the learning rate it is a hyper parameter as you might remember, we also wanted to set some stopping criteria and I am going to use epsilon for the stopping criteria and we will look at various different stopping criteria later, okay.

Now let us come to data generation, data generation means this is anyway a good idea the kind of generate synthetic data wherever we do not have real life data, in this case we just want something which will take some x and y, some randomly distributed points and we try and fit a line to that. so the way we are going to generate this data is I am going to create a formula by myself I am going to say y is actually 15 x plus 5 and then I am going to add some random noise to it, now how this random noise is added, what Gaussian noise means, etc we will come to much later but this is just some simple method for us to create the data.

(Refer Slide Time: 2:38)

The image displays two screenshots of a Jupyter Notebook interface. The top screenshot shows the code editor with the following Python code for linear regression:

```

3Do linear regression
4Initial guesses for w and b
5w1 = rand(1); w0 = rand(1); %Random guesses
6alp = 1.5;
7J(1) = 0.5/m*sum((y-w1*x-w0).^2);
8err = 1; iter = 1;
9while(err>1.e-5)
10    yh = w1*x + w0;
11    DJ0 = (yh-y); dw0 = -alp*sum(DJ0)/m;
12    DJ1 = (yh-y).*x; dw1 = -alp*sum(DJ1)/m;
13    w0 = w0 + dw0;
14    w1 = w1 + dw1;
15
16    iter = iter + 1;
17    J(iter) = 0.5/m*sum((y-w1*x-w0).^2);
18    err = abs(J(iter)-J(iter-1))
19    %err = norm([dw0,dw1]);
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

The bottom screenshot shows the same notebook with an annotation overlay. The annotation is titled "Make initial guesses for w" and "Iterating for w using gradient descent". It explains that random guesses are made for w_1 and w_0 , and that gradient descent is used to iteratively improve these values. The code snippet shown in the annotation is:

```

7    w1 = rand(1); w0 = rand(1); %Random guesses
8    J(1) = 0.5/m*sum((y-w1*x-w0).^2); %Initial value of J

```

Both screenshots include a scatter plot titled "Figure 1" showing a positive linear correlation between x and y. The x-axis ranges from 0 to 1, and the y-axis ranges from 4 to 22. The data points are represented by small blue circles.

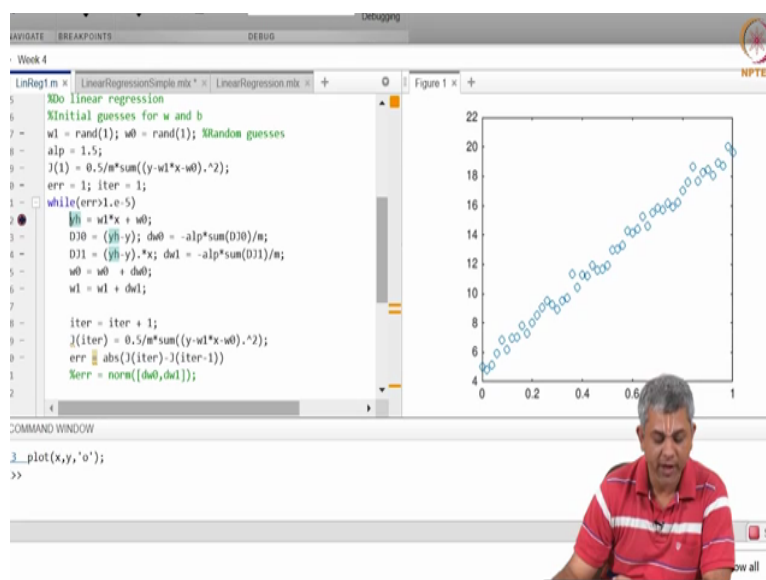
So let us now run this section by section and let us run this section also and you will see the form of data that we have. So on your screens you should be able to see some x, y data (let me reduce the size so that you can see this a little bit). So you have some x data here, y data

here and you can see that roughly it looks like a line will fit it well. so let me go back to the code here this is the same code without all the comments that I showed you earlier, so I am using what is called the debugger and you will see this the same data here x and y data with all these randomly distributed points, ideally we should get a line through all this, okay.

So now we go step by step with our process, we initially take some random values of w_1 and w_0 , remember we are trying to model this as w not plus $w_1 x$ that is our hypothesis function. So we will try and model with this this with some random w_0 and w_1 and as we go through this we will create some new variables error here is our actual stopping criterion, we would like our stopping criterion to be error is less than epsilon.

Now what are the various possibilities I showed you three possibilities there are multiple other possibilities also, but let us say you can look at you know how much is the cost function in this iteration versus how much was it in the previous iteration, if it kind of converges that is one form of error, another form of error is what is the w at this iteration minus the w at previous iteration remember this is a vector so instead of absolute value we will actually have to take a norm in order to find out how close the two values are to each other. So in order to start this loop we give some randomly high value of error in this case I am giving one and iter actually contains my iteration number.

(Refer Slide Time: 4:36)



The image shows a MATLAB debugger window with the following code in the editor:

```
5 %Do linear regression
6 %Initial guesses for w and b
7 w1 = rand(1); w0 = rand(1); %Random guesses
8 alp = 1.5;
9 J(1) = 0.5/m*sum((y-w1*x-w0).^2);
10 err = 1; iter = 1;
11 while(err>1.e-5)
12     yh = w1*x + w0;
13     D10 = (yh-y); dw0 = -alp*sum(D10)/m;
14     D11 = (yh-y).*x; dw1 = -alp*sum(D11)/m;
15     w0 = w0 + dw0;
16     w1 = w1 + dw1;
17
18     iter = iter + 1;
19     J(iter) = 0.5/m*sum((y-w1*x-w0).^2);
20     err = abs(J(iter)-J(iter-1))
21     %err = norm([dw0,dw1]);
22
```

The plot window (Figure 1) shows a scatter plot of data points (blue circles) and a fitted line (blue line) on a coordinate system with x-axis from 0 to 1 and y-axis from 4 to 22. The data points are scattered around the line, showing a positive correlation.

The command window shows the following output:

```
1 _plot(x,y,'o');
2 >>
```

Week 4

The hypothesis function is $\hat{y} = w_1x + w_0$

The gradient has two components

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

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

```

14 yh = w1*x + w0;
15
16 DJ0 = (yh-y);
17 dW0 = -alp*sum(DJ0)/m;
18 DJ1 = (yh-y).*x;
19 dW1 = -alp*sum(DJ1)/m;

```

COMMAND WINDOW

```

3 _plot(x,y,'o');
>>

```

Debugging

Week 4

```

6 Initial guesses for w and b
7 w1 = rand(1); w0 = rand(1); %Random guesses
8 alp = 1.5;
9 J(1) = 0.5/m*sum((y-w1*x-w0).^2);
10 err = 1; iter = 1;
11 while(err>1.e-5)
12     yh = w1*x + w0;
13     DJ0 = (yh-y); dW0 = -alp*sum(DJ0)/m;
14     DJ1 = (yh-y).*x; dW1 = -alp*sum(DJ1)/m;
15     w0 = w0 + dW0;
16     w1 = w1 + dW1;
17
18     iter = iter + 1;
19     J(iter) = 0.5/m*sum((y-w1*x-w0).^2);
20     err = abs(J(iter)-J(iter-1));
21     %err = norm([dW0,dW1]);

```

COMMAND WINDOW

```

3 _plot(x,y,'o');
>>

```

Debugging

Week 4

```

5     w0 = w0 + dW0;
6     w1 = w1 + dW1;
7
8     iter = iter + 1;
9     J(iter) = 0.5/m*sum((y-w1*x-w0).^2);
10    %err = abs(J(iter)-J(iter-1));
11    J: 1x2 double [dW0,dW1];
12    81.7955 66.6060
13    %err = norm([dW0,dW1]);
14    %err = norm([dW0,dW1]);
15    x,yh,'r');
16
17 xlabel('x');
18 ylabel('y');
19 legend({'Data','Linear Fit','Location','NorthWest'})
20 drawnow;
21
22 subplot(212)
23 plot(1:iter,J(1:iter),'-o');

```

COMMAND WINDOW

```

15.1895
>>

```

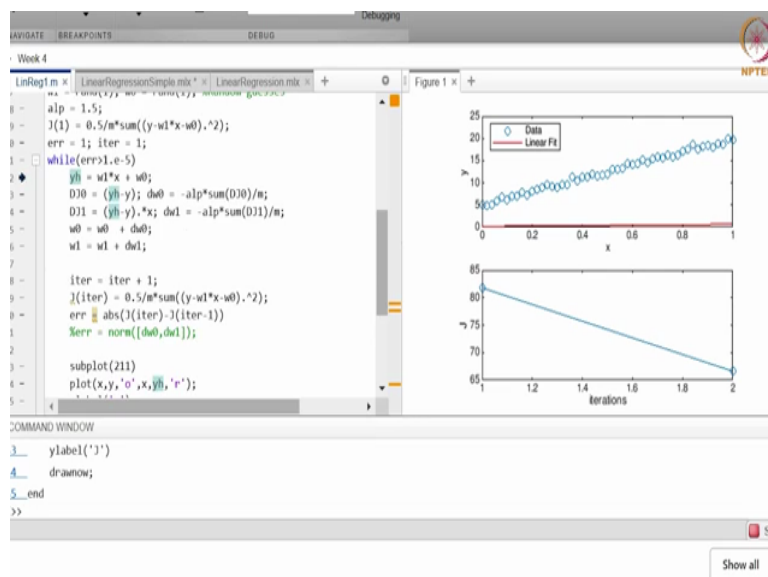
So let us come here we will run our code till this point, y_h is our hypothesis function. So a hypothesis function in this case is a simple linear function $w_0 + w_1 x$ so that is what this piece of code says. After you find this out you need to find out the gradient. So if you remember from the previous slide since we have two variables with which respect to which we are taking the gradient remember both w_0 is a free parameter and w_1 is a free parameter.

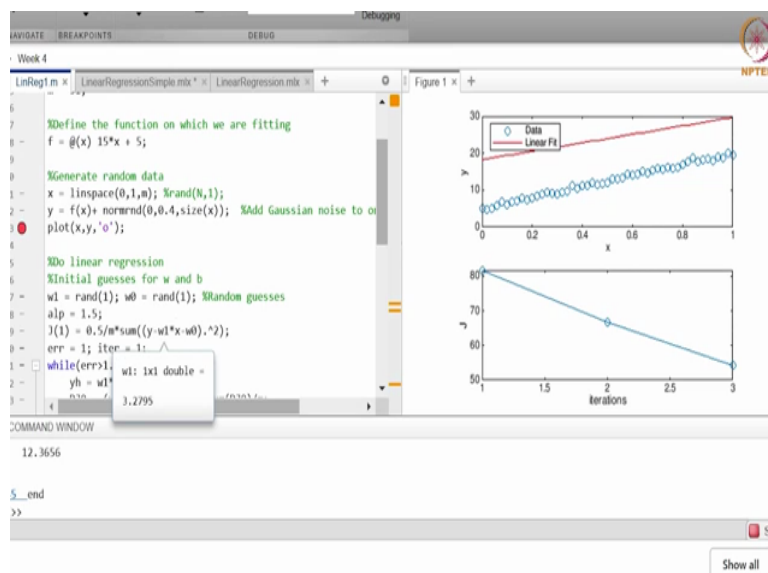
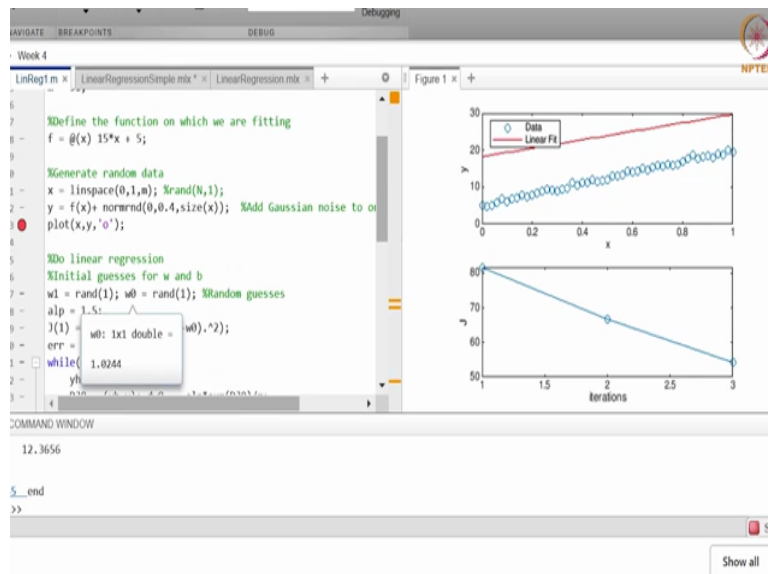
So the gradient of our loss function J has two components, we have derived this in the previous video. So $\frac{\partial J}{\partial w_0}$ is this formulation this formula basically mean of the error $y - \hat{y}$ and $\frac{\partial J}{\partial w_1}$ was $(y - \hat{y})x$ the error multiplied by the corresponding input. So this is what we had that is written here in code form at this point, okay. So what we do is we initialize some w_0 and w_1 , find out the corresponding hypothesis function for all points so this is the ground truth shown here and now you are going to have a line which is the line that you are going to fit based on the guessed values of w_0 and w_1 .

So if you see on your screen now the guessed value is 0.16 and w_1 the guessed value is 0.52 this is just some random guess that the machine has thrown at us, okay. So let us move a little bit further so if we come till here we have now an updated value of w_0 . Now you have a hugely updated value w_0 is 18.25 and w_1 is 11.44.

Now based on this update we are going to get an updated value of the cost function, so the previous cost function at 0 was approximately 82 now the cost decreased to approximately 66 this is what we notice in this. So now let us plot what our function actually looks like.

(Refer Slide Time: 6:54)

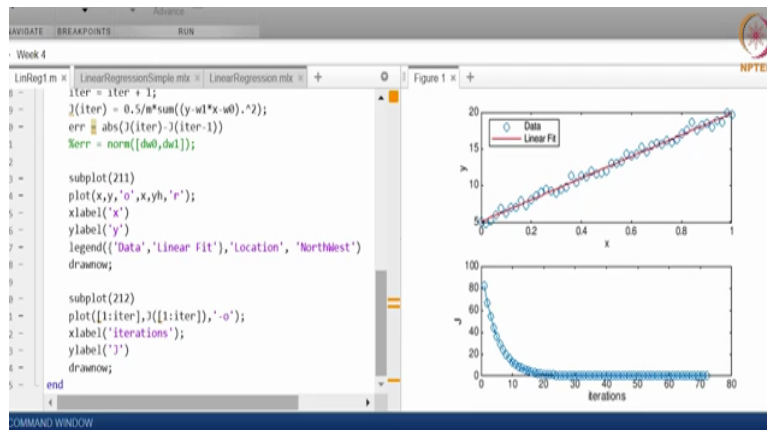




So you see the original data was like this all the dots, all the circles and our linear fit is actually somewhere here this is not at all fitting the original data. So this is after one iteration, this is the situation that we find ourselves in. you will see that the old J was 80 point something and the new J is approximately 66, so it is decreasing but it is still not good enough there is a lot of difference between the data and our actual points which is shown by the value of J also. So the value of J that we have is high, okay.

So let us run this once more, so now you find that the fit has now jumped from here to a little bit more, we can also check the values of w_0 , w_1 that we have got, w_0 is now 1 point something and w_1 is 3 point something and that is the guess that you get, this we are getting due to the update that we have for w according to our gradient decent formula which is sitting here.

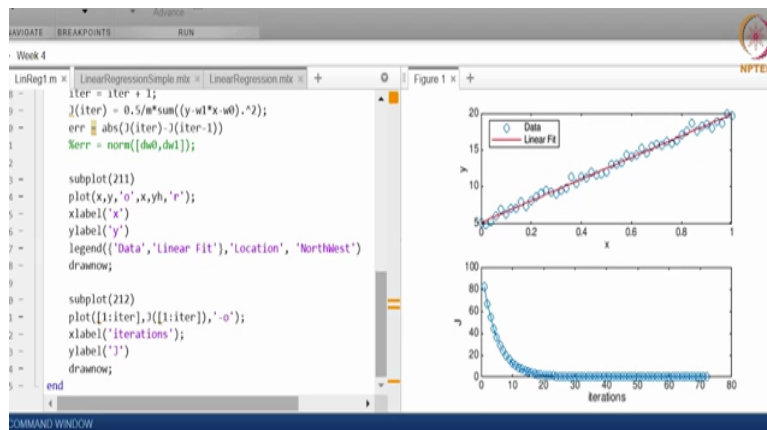
(Refer Slide Time: 7:55)



5.0426

> |

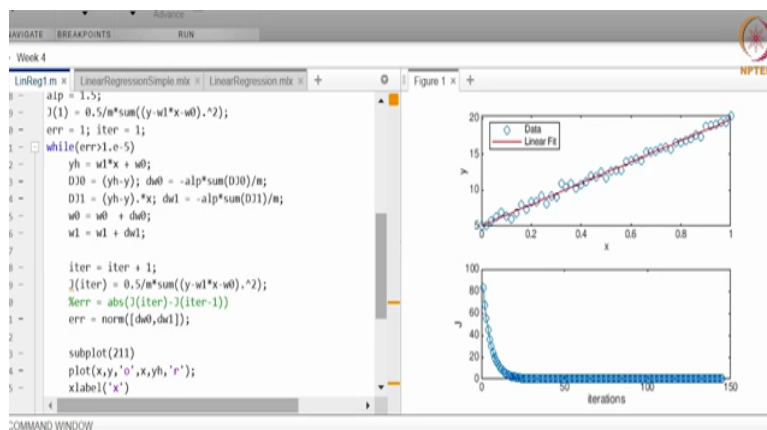
Show all



14.9152

> |

Show all



14.7663

>

Show all

Now let us continue once more and you see that J is uniformly decreasing, now a warning this will not necessarily happen in every practical code that you write it just happens to be true for this example that J is sort of continuously decreasing. So you see this again the fit has improved a little bit may be you might not agree that it has improved but you can see that the line is here and our ideal line would be somewhere there J is also constantly decreasing.

So now let us observe what happens as you keep upon moving, you will notice here what I am printing out here is actually the (J value) sorry the error value the difference between J at this iteration the difference and the J at the previous iteration, I have asked the code to stop as soon as it reaches below 10^{-6} , okay. So you will see that approximately after 72, 73 iterations we have got a very good fit, so this data fit is actually pretty good and we can now look at our final values of w , w not is 5.04, w is 14.9 so these are the values that we have obtained after these many iterations.

So linear regression has work, you can change your error criteria from the criteria of finding out the difference between two J 's to the criteria of finding out what is the difference between my w at this iteration and the previous iteration and we can run this too, you will see that the number of iteration is slightly different, you will also observe that visually the graph does not change very much as J has already reached very low values but it depends on how much tolerance you have for what J is, okay so around 150, 200 iterations w will stop, so w not is some value, w_1 is some other value.

Okay, so you can use any of the stopping criteria you want, some people actually put instead of putting an error criteria they will put a maximum number of iterations that is what is computationally feasible.

(Refer Slide Time: 10:35)

Week 4

```
LinReg1.m | LinearRegressionSimple.mtx | LinearRegression.mtx | Figure 1 x +
```

$w_0 = w_0 - \alpha \frac{\partial J}{\partial w_0}$

$w_1 = w_1 - \alpha \frac{\partial J}{\partial w_1}$

```
20  
21 w0 = w0 + dw0;  
22 w1 = w1 + dw1;  
23  
24 iter = iter + 1;  
  
25 J(iter) = 0.5/m*sum((y-w1*x-w0).^2);  
26 err = abs(J(iter)-J(iter-1));  
27 %err = norm([dw0,dw1]);  
28
```

COMMAND WINDOW

14.7663

Show all

Week 4

```
LinReg1.m | LinearRegressionSimple.mtx | LinearRegression.mtx | Figure 1 x +
```

Create Plots

```
29 subplot(211)  
30 plot(x,y,'o','x,yh','r');  
31 xlabel('x')  
32 ylabel('y')  
33 legend('Data','Linear Fit','location','NorthWest')  
34 drawnow;  
35 subplot(212)  
36 plot([1:iter],J([1:iter]),'-o');  
37 xlabel('iterations');  
38 ylabel('J')  
39 drawnow;  
40  
41 end
```

COMMAND WINDOW

14.7663

Show all

Week 4

```
LinReg1.m | LinearRegressionSimple.mtx | LinearRegression.mtx | Figure 1 x +
```

Linear Regression Example

This is an example for linear regression using simple data

Set Parameters

Set the learning rate and stopping criterion

```
1 alp = 0.1; %Learning rate  
2 ep = 1.e-6; %Stopping tolerance
```

Data Generation

Data was generated synthetically by taking linear data and adding noise to it.

```
3 m = 51;  
4 % = linspace(0,1, m);
```

COMMAND WINDOW

14.7663

Show all



So just to show you the same thing in the other place remember all we have done is w not is w not minus $\alpha \Delta J \Delta w$ not, w_1 is w_1 minus $\alpha \Delta J \Delta w_1$ and all these plots so you can take a look at both these things, this particular video showed you a simple example of how to use linear regression to fit some random data, there are some (\cdot) (10:58) here, one thing that we have not yet discussed is you know this I have just fit a line but in the previous example I showed you a linear fit, a quadratic fit, a cubic fit.

So can you do a linear quadratic cubic cube, quadratic and cubic also using the same procedure? It turns out that you can with a small minor modification which we will see in the next video, thank you.