# Linear Regression with Multiple Variables

| | |
|---|---|
| Class | Machine Learning |
| Created | @May 22, 2020 6:40 PM |
| Materials | Lecture4.pdf |
| Reviewed | ☐ |
| Type | Lecture Notes |

👉 **Notes by Hemanth Kotagiri**

📧 **Email:** hemanth.kotagiri43@gmail.com
💾 **GitHub:** https://github.com/hemanth-kotagiri/
🏢 **LinkedIn**: https://www.linkedin.com/in/hemanth-kotagiri/

## Linear Regression with Multiple Variables

### Multivariate Linear Regression

- Linear regression with multiple variables is known as Multivariate Linear Regression.
- Here, based on the hypothesis we defined in the previous notes, if we have a hypothesis function with multiple variables such as x1, x2, x3... xn; then, we would represent the hypothesis function like this:

$$h\theta(x) = \theta 0 + \theta 1 * x1 + \theta 2 * x2 + ... + \theta n * xn$$

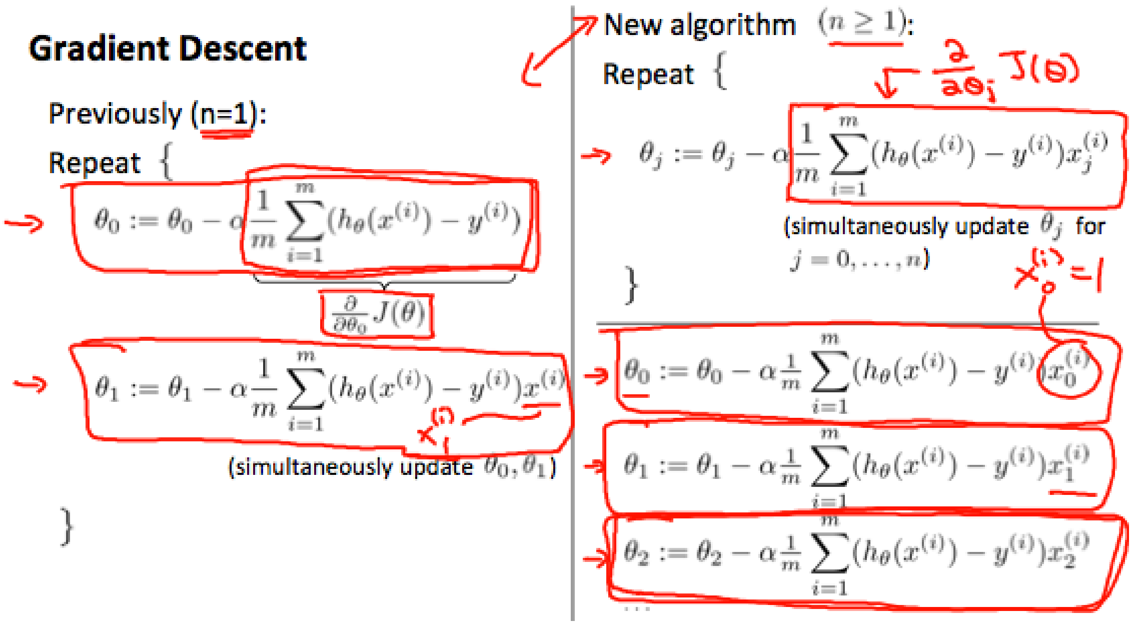
- We can choose to represent our hypothesis function in the means of vectors, in many programming languages such as MATLAB and Python, using a vectoral representation increases the computational speed.

## GRADIENT DESCENT FOR MULTIPLE VARIABLES

- Let n denote the number of features given in the data set.
- Previously, we had n = 1, that is just one feature, but in this, we may have more than 2 features. So the Gradient Descent algorithm has a minor change such as this:

```
repeat until convergence: {
    $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$        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 {

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

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

(simultaneously update $\theta_0, \theta_1$)

}

New algorithm $(n \geq 1)$:

Repeat {

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

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

}

$x_0^{(i)} = 1$

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

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

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

...

## WAYS TO SPEED UP GRADIENT DESCENT

- We can speed up gradient descent by having each of our input values in roughly the same range so that the contour plots which are highly skewed can be brought down to circles so that the gradient descent would find it easier to get to the global minimum.
- We can ideally choose the range of our input values to be between -1 and 1. We can modify the input feature values so that all of them are

roughly the same.

- For this, we can use two techniques. Feature Scaling or Mean Normalization.

### ▼ FEATURE SCALING

- In this method, we divide the input values by the range (the maximum value minus the minimum value) resulting in a new range of just 1.

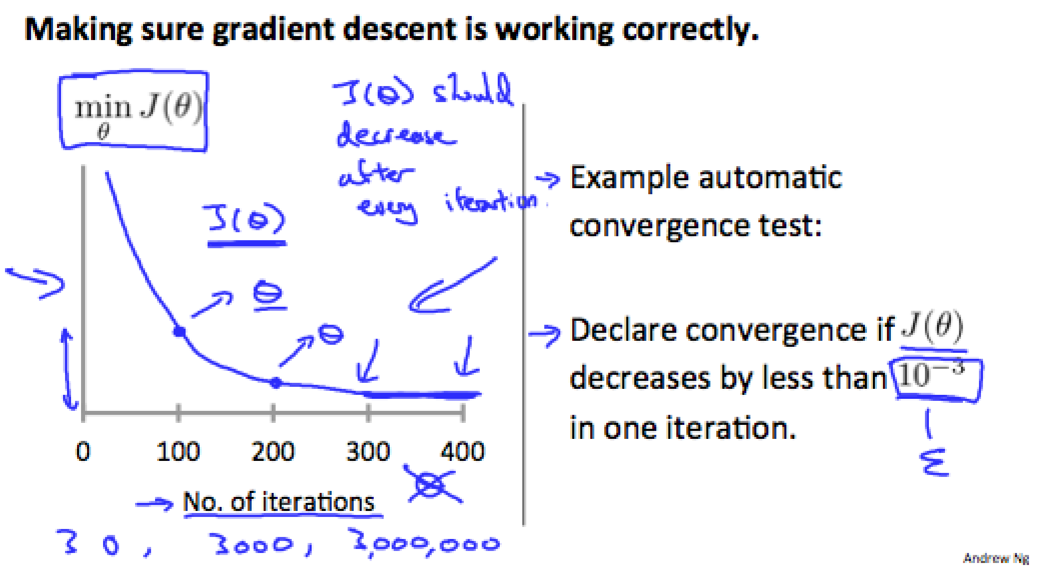
### ▼ MEAN NORMALIZATION

- In this method, we can subtract the average value of the input feature with the input variable and divide it by the standard deviation or by the range of the input feature.
- Note that dividing it with the standard deviation with yield different results compared to diving it with the range.
- for example if xi represents housing prices with a range of 100 to 2000 and a mean value of 1000, then xi = (price - 1000)/1900.

## ▼ LEARNING RATE : α

- **Debugging Gradient Descent:** One good way to check if gradient descent is working good is by plotting. To debug the 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:

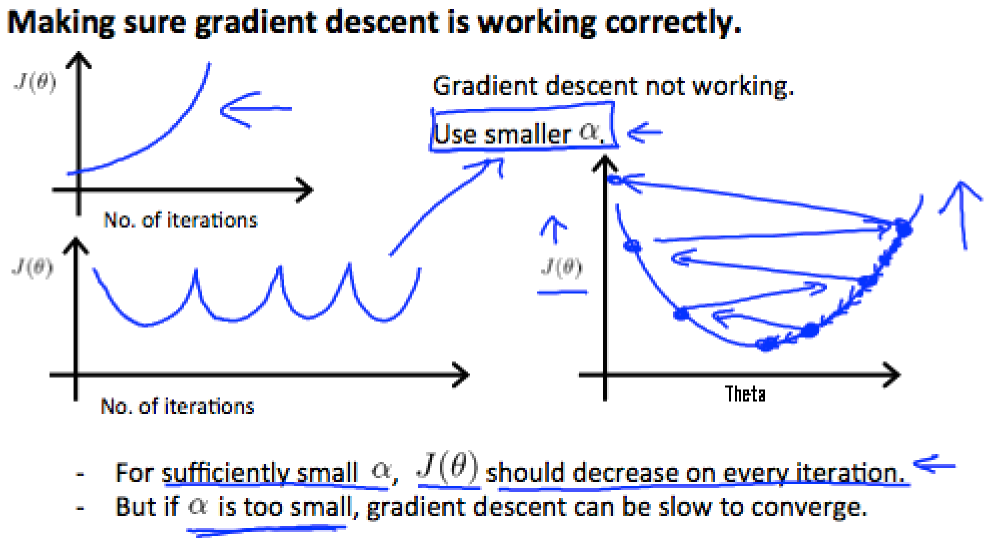
Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10^−3. However in practice it's difficult to choose this threshold value.


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.

- ⚠ **It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration. That doesn't mean to choose an extremely low learning rate which will potentially slow down gradient descent even more.**


Making sure gradient descent is working correctly.

Gradient descent not working.

Use smaller $\alpha$.

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

- **SUMMARY**: If α is too small: slow convergence. If α is too large: may not decrease on every iteration and thus may not converge.

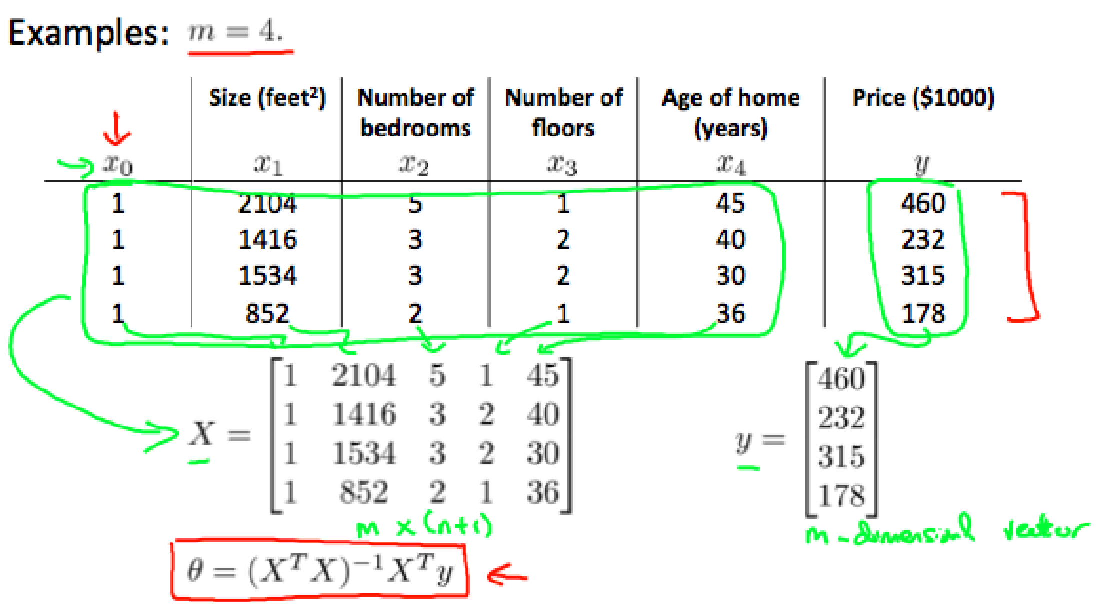
## ▼ POLYNOMIAL REGRESSION

- Our hypothesis function need not to be always linear to fit the data reasonably. We can choose polynomial terms too in the function to hopefully get a good hypothesis function.
- We can improve our hypothesis function in a couple of different ways.

- We can combine multiple features into one. For example, we can combine x1 and x2 as x1*x2 and make a new feature x3.
- One thing to keep in mind while including polynomial terms is that feature scaling becomes very important. Because, some of the features may shoot to a very large value making the curve quite unstable.
- Say including the feature x3 = x1*(x2^3). In this case, if the value of x2 is 100, and x1 is 10, then the value of x3 will be 100000000 which is very huge. In this case, feature scaling comes handy to drop that value to a desired range so that the curve doesn't act unstable.

## COMPUTING PARAMETERS ANALYTICALLY

- We can also compute the parameters vector theta using analytical techniques such as like this: theta = ((XT*X)^-1)*(XT*Y)
- There is no **need to do feature scaling** while finding the parameters analytically.



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 |

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

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

**Comparison between Gradient Descent and Normal Equation**

| Gradient Descent | Normal Equation |
|---|---|
| Need to choose alpha | No need to choose alpha |
| Needs many iterations | No need to iterate |

| Gradient Descent | Normal Equation |
|---|---|
| O (kn^2kn2) | O (n^3) : to calculate inverse of (X^T*X) |
| Works well when n is large | Slow if n is very large |