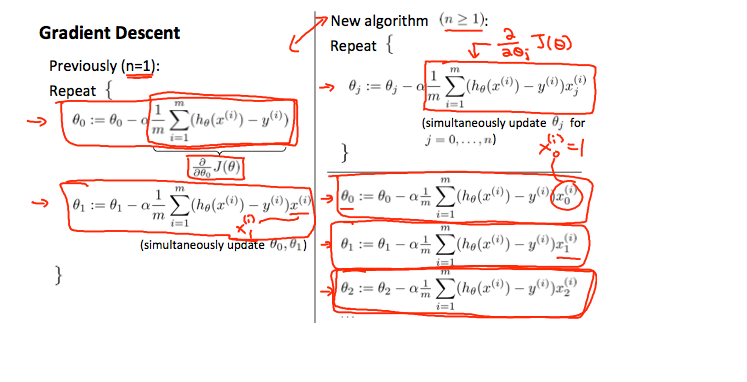
# Multivariate linear regression

Previously n was 1 i.e. only one feature, like based on area we predict the housing price, now multiple n, i.e. multiple features like area of house, age of house, no of floors etc.



# GD Feature scaling

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:

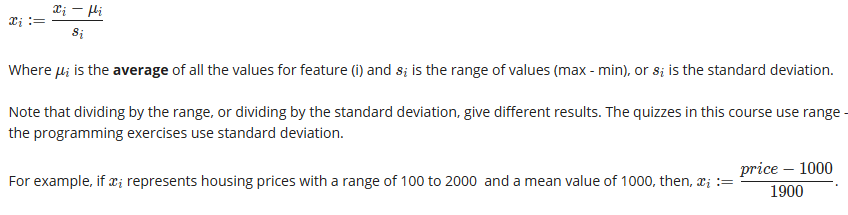
−1 ≤ x(i)x\_{(i)}x(i)​ ≤ 1

or

−0.5 ≤ x(i)x\_{(i)}x(i)​ ≤ 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.

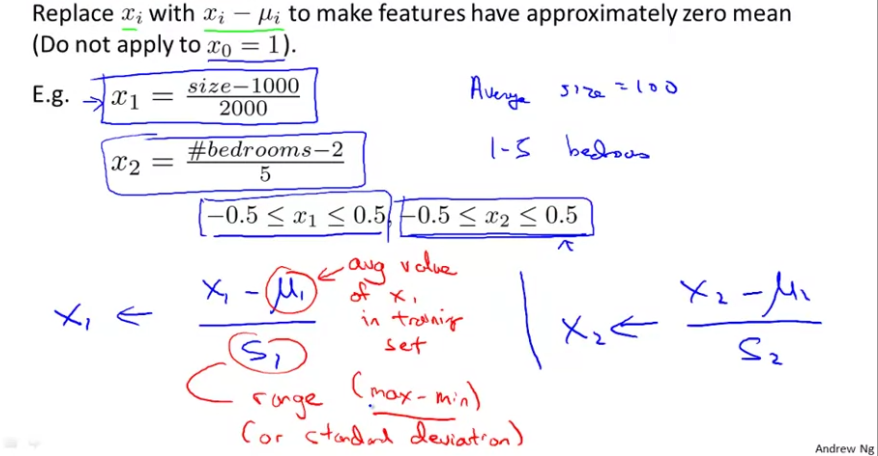
Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the 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:



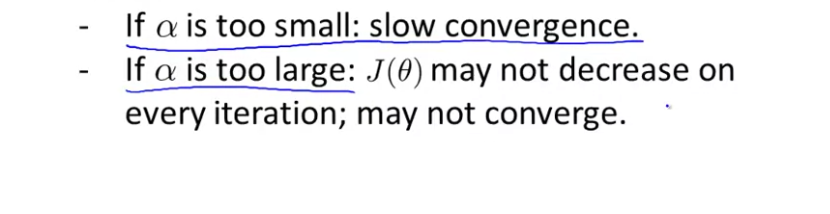
## Example to feature scaling and mean normalization

Say, you have two features i.e. size of bedroom and no of bedrooms. Size ranges from 0 to 2000m2 and no ranges from 1-5, so this makes the gradient descent algorithm more difficult to reach the local minimum, as the concentric ovals look tall and slim. To make it more oval or circle like the values have to scaled in a way that the values lie between -1 to 1, like divide the area of rooms/2000 and the no of bedrooms by 5.

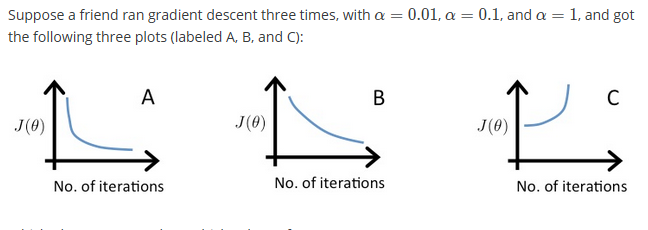
Scale the features in a way that the mean value of them is zero

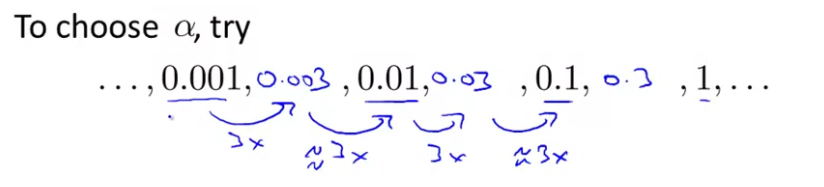


# Learning rate



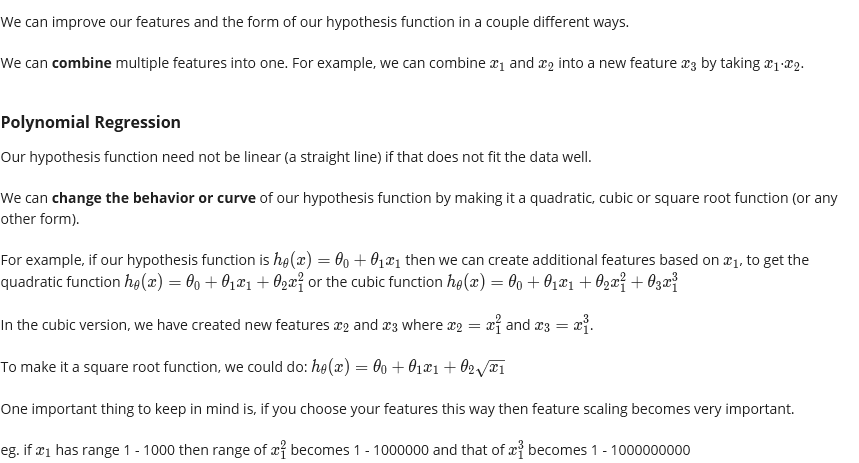




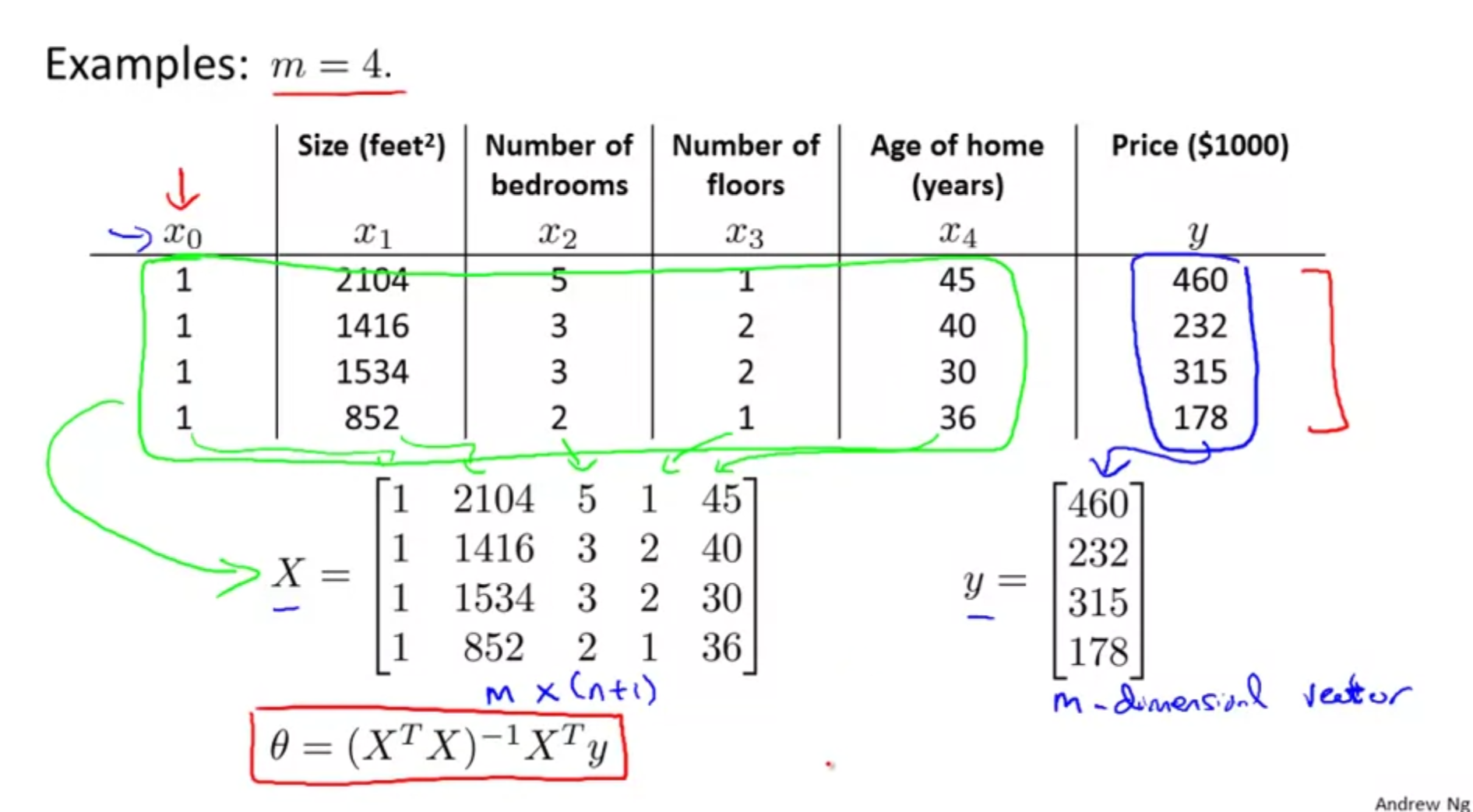


# Polynomial regression

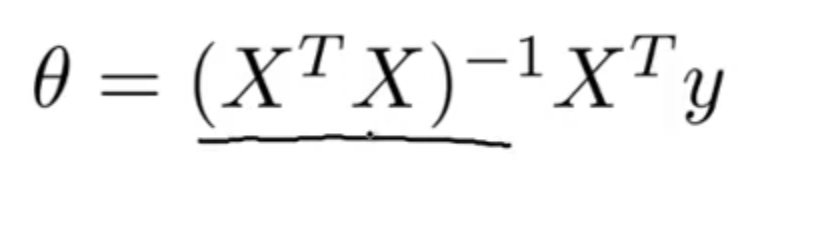
If the data points are in curved fashion then linear regression is bad for this data set, instead a polynomial regression is better, for the curves can come down cubic functions can be used.



# Solving for Theta without GD using Normal method

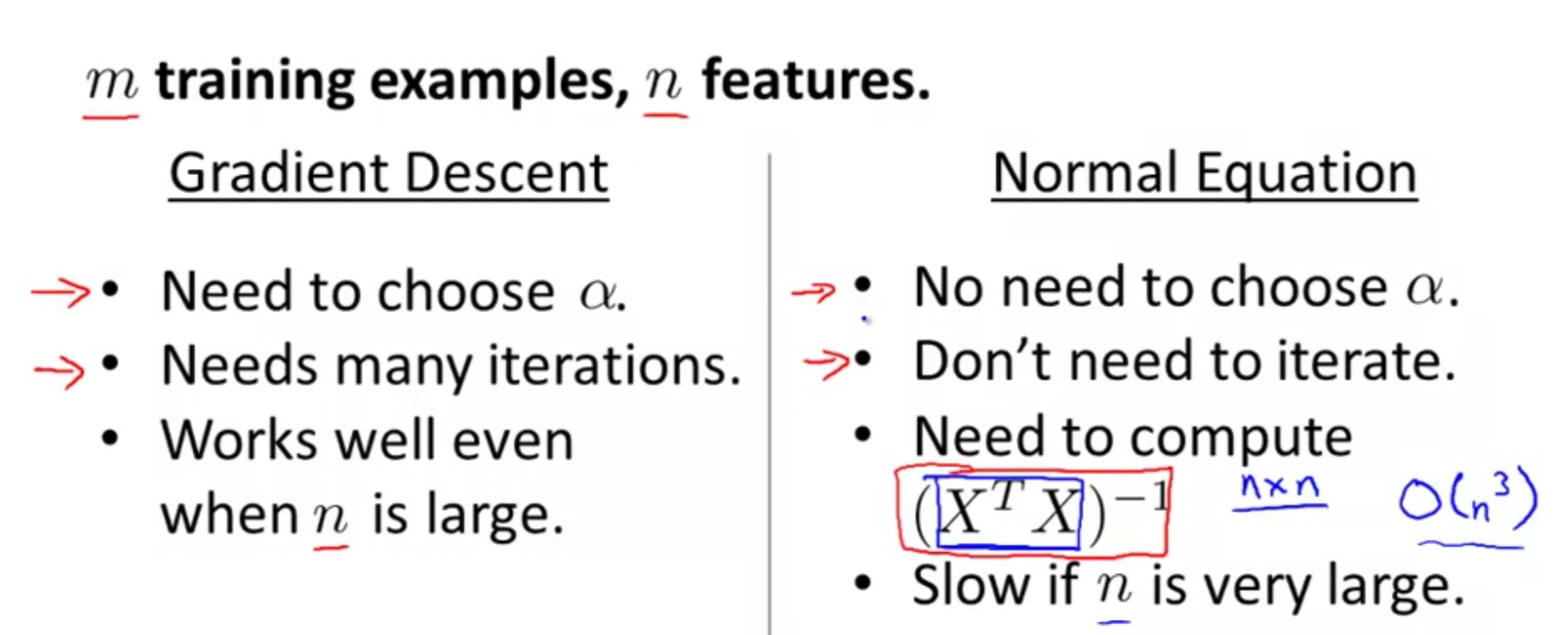


Gradient descent is an iterative algorithm, there’s a direct method instead of this



If number of features ‘n’ is large the normal method will take lots of time in the order of O (n^3), GD is faster here.

# Normal vs GD

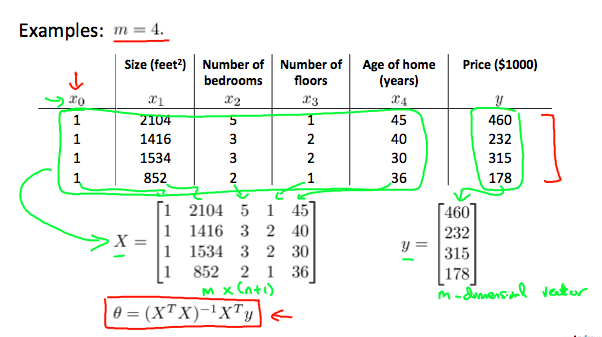


n in the range of 100-10000 is okay, more than that is bad using normal equation.

**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 x0x\_0x0​ and x1x\_1x1​. 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:

θ=(XTX)−1XTy\theta = (X^T X)^{-1}X^T yθ=(XTX)−1XTy



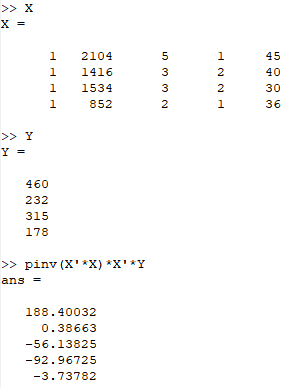
There is **no need** to do feature scaling with the normal equation.

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

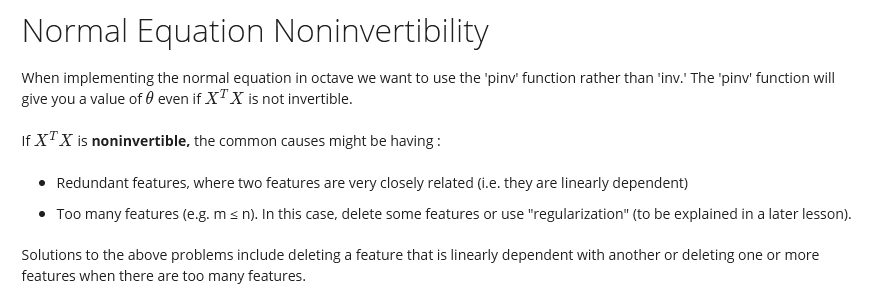
|  |  |
| --- | --- |
| **Gradient Descent** | **Normal Equation** |
| Need to choose alpha | No need to choose alpha |
| Needs many iterations | No need to iterate |
| O (kn2kn^2kn2) | O (n^3), need to calculate inverse of X**’**\*X |
| Works well when n is large | Slow if n is very large |

With the normal equation, computing the inversion has complexity, 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.

# Octave implementation



# Normal equation: non-invertibility



# Vectorization – faster than for loop method

