# Experiment 2: Multivariate Linear Regression

September 30, 2017

## 1 Description

In this exercise, you will investigate multivariate linear regression using gradient descent and the normal equations. You will also examine the relationship between the cost function $J(\theta)$, the convergence of gradient descent, and the learning rate $\alpha$.

## 2 Data

Download ex2Data.zip, and extract the files from the zip file. This is a training set of housing prices in Portland, Oregon, where the outputs $y^{(i)}$ are the prices and the inputs $x^{(i)}$ are the living area and the number of bedrooms. There are $m = 47$ training examples.

## 3 Preprocessing Your Data

Load the data for the training examples into your program and add the $x_0 = 1$ intercept term into your x matrix. Recall that the command in Matlab/Octave for adding a column of ones is

```
x = [ones(m, 1), x];
```

Take a look at the values of the inputs $x^{(i)}$ and note that the living areas are about 1000 times the number of bedrooms. This difference means that preprocessing the inputs will significantly increase gradient descent's efficiency.

In your program, scale both types of inputs by their standard deviations and set their means to zero. In Matlab/Octave, this can be executed with

```
sigma = std(x);
mu = mean(x);
x(:,2) = (x(:,2) - mu(2))./ sigma(2);
x(:,3) = (x(:,3) - mu(3))./ sigma(3);
```

## 4 Gradient Descent

Previously, you implemented gradient descent on a univariate regression problem. The only difference now is that there is one more feature in the matrix x.

The hypothesis function is still

$$h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i,$$

and the batch gradient descent update rule is

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

Once again, initialize your parameters to $\theta = \vec{0}$.

# 5 Selecting A Learning Rate Using $J(\theta)$

Now it's time to select a learning rate $\alpha$. The goal of this part is to pick a good learning rate in the range of

$$0.001 \leq \alpha \leq 10$$

You will do this by making an initial selection, running gradient descent and observing the cost function, and adjusting the learning rate accordingly. Recall that the cost function is defined as

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

The cost function can also be written in the following vectorized form,

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

where

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

The vectorized version is useful and efficient when you're working with numerical computing tools like Matlab/Octave. If you are familiar with matrices, you can prove to yourself that the two forms are equivalent.

While in the previous exercise you calculated $J(\theta)$ over a grid of $\theta_0$ and $\theta_1$ values, you will now calculate $J(\theta)$ using the $\theta$ of the current stage of gradient descent. After stepping through many stages, you will see how $J(\theta)$ changes as the iterations advance.

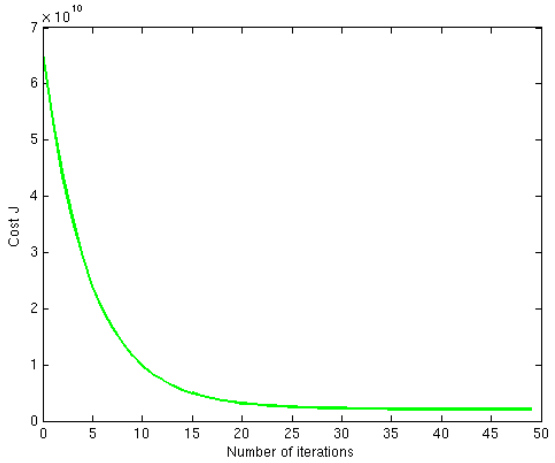
Now, run gradient descent for about 50 iterations at your initial learning rate. In each iteration, calculate $J(\theta)$ and store the result in a vector J. After the last iteration, plot the J values against the number of the iteration. In Matlab/Octave, the steps would look something like this:

```
theta = zeros(size(x(1,:)))'; % initialize fitting parameters
alpha = %% Your initial learning rate %%
J = zeros(50, 1);

for num_iterations = 1:50
  J(num_iterations) = %% Calculate your cost function here %%
  theta = %% Result of gradient descent update %%
end

% now plot J
% technically, the first J starts at the zero-eth iteration
% but Matlab/Octave doesn't have a zero index
figure;
plot(0:49, J(1:50), '-')
xlabel('Number of iterations')
ylabel('Cost J')
```

If you picked a learning rate within a good range, your plot should appear like the figure below.




If your graph looks very different, especially if your value of $J(\theta)$ increases or even blows up, adjust your learning rate and try again. We recommend testing alphas at a rate of of 3 times the next smallest value (i.e. 0.01, 0.03, 0.1, 0.3 and so on). You may also want to adjust the number of iterations you are running if that will help you see the overall trend in the curve.

To compare how different learning learning rates affect convergence, it's helpful to plot J for several learning rates on the same graph. In Matlab/Octave, this can be done by performing gradient descent multiple times with a *hold on* command between plots. Concretely, if you've tried three different values of alpha (you should probably try more values than this) and stored the costs in $J1$, $J2$ and $J3$, you can use the following commands to plot them on the same figure:

```
plot(0:49, J1(1:50), 'b-');
hold on;
plot(0:49, J2(1:50), 'r-');
plot(0:49, J3(1:50), 'k-');
```

The final arguments 'b-', 'r-', and 'k-' specify different plot styles for the plots. Type

```
help plot
```

at the Matlab/Octave command line for more information on plot styles.

**Answer the following questions:**

1. Observe the changes in the cost function happens as the learning rate changes. What happens when the learning rate is too small? Too large?

2. Using the best learning rate that you found, run gradient descent until convergence to find

   (a) The final values of $\theta$

   (b) The predicted price of a house with 1650 square feet and 3 bedrooms. Don't forget to scale your features when you make this prediction!

# 6 Normal Equation

In the Normal Equations video, you learned that the closed-form solution to a least squares fit is

$$\theta = \left(X^T X\right)^{-1} X^T \vec{y}.$$

Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no 'loop until convergence' like in gradient descent.

**Answer the following questions:**

1. In your program, use the formula above to calculate $\theta$. Remember that while you don't need to scale your features, you still need to add an intercept term.

2. Once you have found $\theta$ from this method, use it to make a price prediction for a 1650-square-foot house with 3 bedrooms. Did you get the same price that you found through gradient descent?