Week 12: Gradient Descent and More MATLAB Practice

Problem 1: Gradient Descent

In the previous weeks, we defined the cost function to be

$$J(x) = \frac{1}{2m}\sum_{i=1}^{m}\left(h\left(x^i\right) - y^i\right)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(\theta_0 + \theta_1 x^i - y^i\right)^2$$

The lower the value of the cost function, the better our hypothesis function h(x) approximates the actual data. Whenever we would like to minimize a function, we should consider taking a derivative. It turns out that J(x) is entirely convex, and a straightforward process called gradient descent will let us nearly minimize J(x).

First, we compute the gradient, and this is done by taking the partial derviatives of J with respect to θ_0 and θ_1:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}\left(\theta_0 + \theta_1 x^i - y^i\right)$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}\left(\theta_0 + \theta_1 x^i - y^i\right)x^i$$

If we follow the gradient, we will take the fastest ascent to the peak of the function. Since we want to minimize J, we will follow the negative of the gradient to take the fastest descent to the bottom of the function. The gradient descent update can be characterized as follows:

$$\theta_0 := \theta_0 - \alpha\frac{\partial J}{\partial \theta_0} \quad ; \quad \theta_1 := \theta_1 - \alpha\frac{\partial J}{\partial \theta_1}$$

where α is called the learning rate and it is a small value such as 0.0001 (think of this like following the tangent line to a function by a small amount). If we repeat this update some fixed amount of time, such as 1000 iterations, and if the learning rate is suitable, the algorithm will get very close to finding the optimal theta for J(x).

Given that you are now able to generate random data, please implement this gradient descent function using as much vectorization as you can. Typically, we initialize the function by guessing θ_0 and θ_1 are both 0. Clearly these are bad guesses, but gradient descent will use the gradient to help improve the guesses.

If you finish this and are up for the challenge, try implementing a three-dimensional version of this. You'll have a third variable (also called a feature), θ_2, that you will have to work into your cost function and gradient descent.

Problem 2: List Selection

Implement a vectorized function

```
        function [positiveData] = selectPositive( data )
```

with no for loops or if statements that removes all non-positive data from a list.

Sample cases:
selectPositive([1 , 2 , 3 , 4 , 5]) = [1 , 2 , 3 , 4 , 5]
selectPositive([0 , -1 , -2 , -3 , -4 , -5]) = []
selectPositive([1 , -1 , 2 , -2 , 3 , -3 , 5 , -5 , 4 , -4]) = [1 , 2 , 3 , 4 , 5]

Problem 3: Count Trend Reversal

One classic MATLAB vectorization exercise is to count the number of trend reversals in some data. You should be familiar with how to do this before your exam! For example, the data [1, 2, 3, 2, 1] has 1 trend reversal: it is increasing at first 1 -> 2 -> 3, and then it starts decreasing with 3 -> 2 -> 1. Implement a vectorized function,

```
             function [count] = countReversals( data )
```

with no for loops or if statements that counts the number of trend reversal in some data points.

Sample cases:
countReversals([1 , 1 , 1 , 2 , 5 , 6 , 3 , 4 , 3 , 5 , 3 , 3 , 7 , 9]) = 6
countReversals([1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 8 , 8 , 8]) = 0
countReversals([6 , 3 , 3 , 3 , 3 , 3 , 3 , 3]) = 0