Homework 01

Linear Regression with One Variable

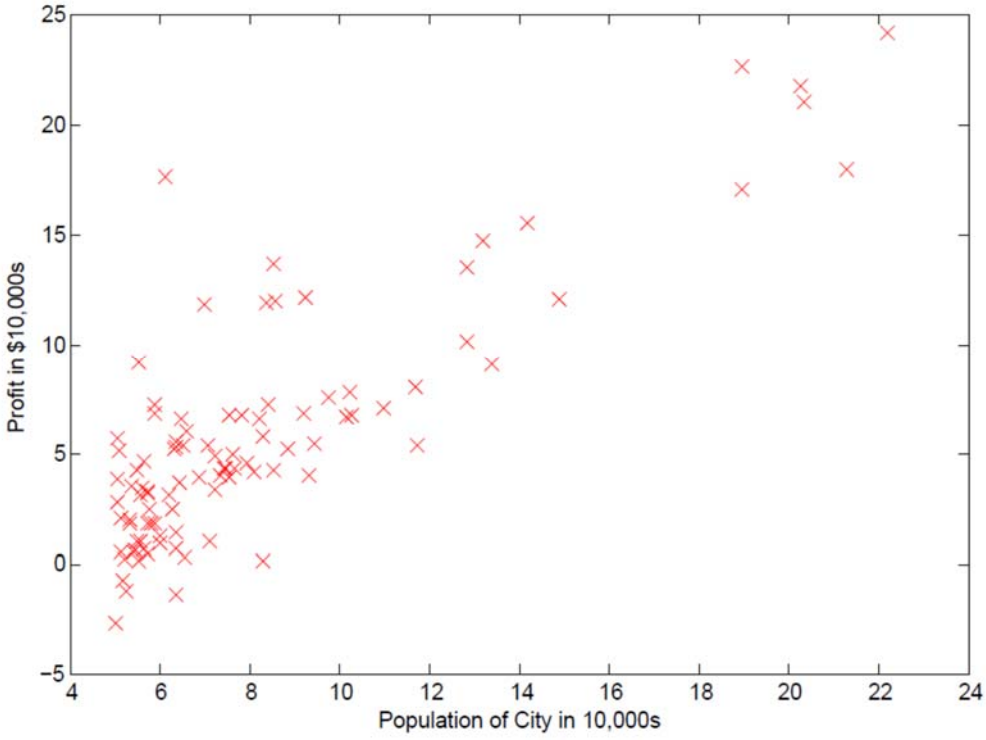
In this part of this exercise, you will implement linear regression with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities. You would like to use this data to help you select which city to expand to next.

The file **hw1data1.txt** contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss. The **hw1.m** script has already been set up to load this data for you.

Part 1: Plotting the Data

Before starting on any task, it is often useful to understand the data by visualizing it. For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot (profit and population).

In **hw1.m**, the dataset is loaded from the data file into the variables X and y. Next, the script calls the **plotData** function to create a scatter plot of the data. Your job is to complete **plotData.m** to draw the plot. Now, when you continue to run **hw1.m**, our end result should look like the Figure below, with the same red \x" markers and axis labels.



Part 2: Compute the Cost

In this part, you will fit the linear regression parameters θ to our dataset using gradient descent.

In **hw1.m**, we have already set up the data for linear regression. Starting from line 28, we add another dimension to our data to accommodate the θ_0 intercept term. We also initialize the initial parameters to 0 and the learning rate **alpha** to 0.01.

As you perform gradient descent to learn minimize the cost function $J(\theta)$, it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate $J(\theta)$ so you can check the convergence of your gradient descent implementation. You need to complete the code in the file **computeCost.m**, which is a function that computes $J(\theta)$. As you are doing this, remember that the variables X and y are not scalar values, but matrices whose rows represent the examples from the training set. Once you have completed the function, the next step in **hw1.m** will run **computeCost** once using θ initialized to zeros, and you will see the cost printed to the screen. You should expect to see a cost of **32.07**.

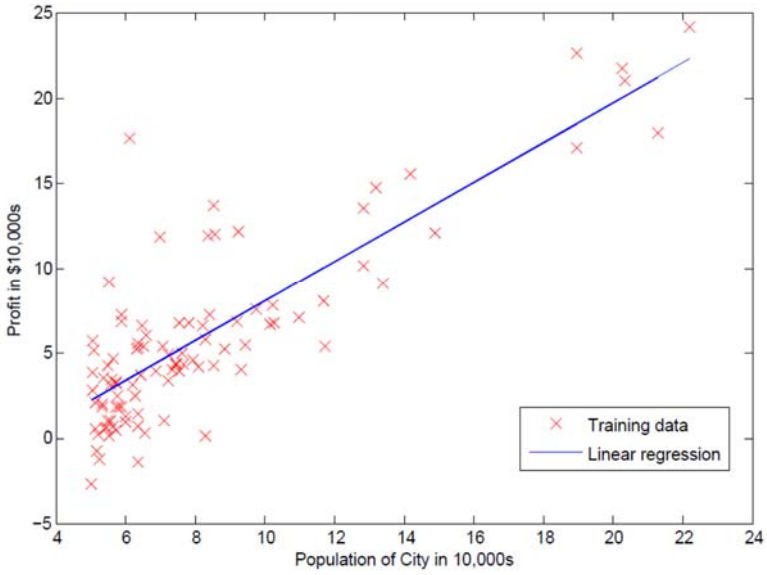
Part 3: Gradient Descent

Next, you will implement gradient descent in the file **gradientDescent.m**. The loop structure has been written for you, and you only need to supply the updates to θ within each iteration.

As you program, make sure you understand what you are trying to optimize and what is being updated. Keep in mind that the cost $J(\theta)$ is parameterized by the vector θ, not X and y. That is, we minimize the value of $J(\theta)$ by changing the values of the vector θ, not by changing X or y. A good way to verify that gradient descent is working correctly is to look at the value of $J(\theta)$ and check that it is decreasing with each step. The starter code for **gradientDescent.m** calls **computeCost** on every iteration and prints the cost. Assuming you have implemented gradient descent and computeCost correctly, your value of $J(\theta)$ should never increase, and should converge to a steady value by the end of the algorithm.

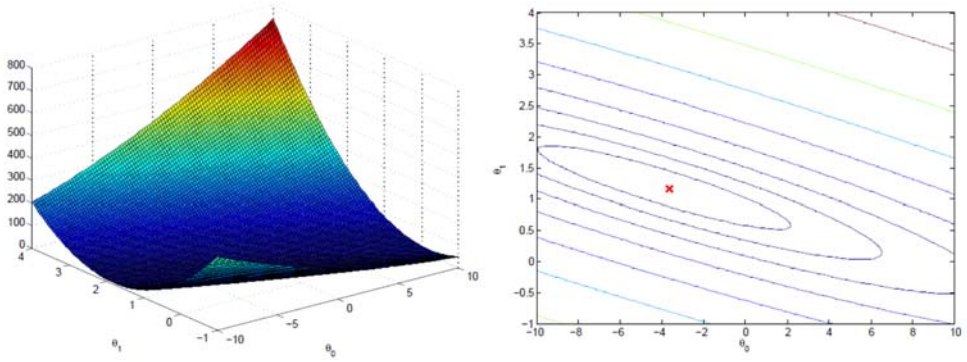
After you are finished, **hw1.m** will use your final parameters to plot the linear fit. The result should look something like this Figure. Your final values for θ will also be used to make predictions on profits in areas of 35,000 and 70,000 people.

Homework 01



Part 4: Visualizing $J(\theta)$

To understand the cost function $J(\theta)$ better, you will now plot the cost over a 2-dimensional grid of θ_0 and θ_1 values. You will not need to code anything new for this part, but you should understand how the code you have written already is creating these images. In the next step of **hw1.m**, there is code set up to calculate $J(\theta)$ over a grid of values using the **computeCost** function that you wrote. After these lines are executed, you will have a 2-D array of $J(\theta)$ values. The script **hw1.m** will then use these values to produce surface and contour plots of $J(\theta)$ using the **surf** and **contour** commands. The plots should look something like these Figures:



(a) Surface (b) Contour, showing minimum

Submission:

To submit, turn in **computeCost.m**, **gradientDescent.m**, and **plotData.m** on Canvas.