Práctica 1: Linear Regression

1 Files included

ex1data1.txt - Dataset for linear regression with one variable
myPlot.m - Function to display the dataset
computeCost.m - Function to compute the cost of linear regression
gradientDescent.m - Function to run gradient descent to learn theta
myGradient.m - Function to run gradient descent
myVisual.m – function to visualize cost function

2 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 ex1data1.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 myPlot.m script has already been set up to load this data for you.

2.1 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). (Many other problems that you will encounter in real life are multi-dimensional and can't be plotted on a 2-d plot.) In myPlot.m, the dataset is loaded from the data file into the variables X and y:

```
data = load('ex1data1.txt' ); % read comma separated data
X = data(:, 1); y = data(:, 2);
m = length(y); % number of training examples
```

Next, we need to create a scatter plot of the data. Your job is to complete myPlot.m to draw the plot; modify the file and using the plot, xlabel and ylabel commands. To learn more about these commands, you can use the help command. For instance, you can type help plot at the Matlab command prompt (alternatively you can search online for plotting documentation. (To change the markers to red “x”, we used the option `rx' together with the plot command, i.e., plot(..,[your options here],.., `rx');).

2.2 Gradient Descent

In this part, you will fit the linear regression parameters θ to our dataset using gradient descent. The objective of linear regression is to minimize the cost function given in class. Recall that the parameters of your model are the θ_j values. These are the values you will adjust to minimize cost J(θ). One way to do this is to use the gradient descent algorithm. In gradient descent, each iteration performs the update

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

With each step of gradient descent, your parameters θ_j come closer to the optimal values that

will achieve the lowest cost J(θ).

Implementation Note: We store each example as a row in the the X matrix in Matlab. To take into account the intercept term (θ_0), we add an additional first column to X and set it to all ones. This allows us to treat θ_0 as simply another `feature'.

In myGradient.m, we have already set up the data for linear regression. In the following lines, 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.

```
X = [ones(m, 1), data(:,1)]; % Add a column of ones to x
theta = zeros(2, 1); % initialize fitting parameters

iterations = 1500;
alpha = 0.01;
```

2.2.1 Computing the cost J(θ)

As you perform gradient descent to learn minimize the cost function J(θ), it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate J(θ) so you can check the convergence of your gradient descent implementation.

Your next task is to complete the code in the file computeCost.m, which is a function that computes J(θ). 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 is to run myGradient.m which will execute computeCost using θ initialized to zeros. Test your computeCost function with θ initialized to zeros, you should expect an initial cost of 32.07.

2.2.2 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(θ) is parameterized by the vector θ, not X and y. That is, we minimize the value of J(θ) 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(θ) 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(θ) should never increase, and should converge to a steady value by the end of the algorithm.

After you are finished, myGradient.m will use your final parameters to plot the linear fit. Your final values for θ will also be used to make predictions on profits in areas of 35,000 and 70,000 people.

Implementation Notes:

- Matlab array indices start from one, not zero. If you're storing θ_0 and θ_1 in a vector called theta, the values will be theta(1) and theta(2).
- If you are seeing many errors at runtime, inspect your matrix operations to make sure that you're adding and multiplying matrices of compatible dimensions. Printing the dimensions of variables with the size command will help you debug.
- By default, Matlab interprets math operators to be matrix operators. This is a common source of size incompatibility errors. If you don't want matrix multiplication, you need to add the “dot” notation to specify this to Matlab. For example, A*B does a matrix multiply, while A.*B does an element-wise

multiplication.

2.3 Visualizing J(θ)

To understand the cost function J(θ) 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 myVisual.m, there is code set up to calculate J(θ) over a grid of values using the computeCost function that you wrote.

```
% initialize J vals to a matrix of 0's
J vals = zeros(length(theta0 vals), length(theta1 vals));

% Fill out J vals
for  i = 1:length(theta0 vals)
   for  j = 1:length(theta1 vals)
      t = [theta0 vals(i); theta1 vals(j)];
      J vals(i,j) = computeCost(x, y, t);
   end
end
```

After these lines are executed, you will have a 2-D array of J(θ) values. The script myVisual.m will then use these values to produce surface and contour plots of J(θ) using the surf and contour commands.

The purpose of these graphs is to show you that how J(θ) varies with changes in θ_0 and θ_1. The cost function J(θ) is bowl-shaped and has a global minima. (This is easier to see in the contour plot than in the 3D surface plot). This minimum is the optimal point for θ_0 and θ_1, and each step of gradient descent moves closer to this point.

Submitting your answer

The práctica can be solved in teams of two people (1 submission per team). Submission is through the Aula Global. Submissions should contain the code of the files you modified and the plots your programs generated. Deadline is the beginning of the next práctica. Late submissions will be penalized.