Programming Exercise 1: Linear Regression

Introduction

In this exercise, you will implement linear regression and get to see it work on data. To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the cd command in Octave/MATLAB to change to this directory before starting this exercise.

Files included in this exercise

`ex1.m` - Octave/MATLAB script that steps you through the exercise

`ex1data1.txt` - Dataset for linear regression with one variable

[*] `warmUpExercise.m` - Simple example function in Octave/MATLAB

[*] `plotData.m` - Function to display the dataset

[*] `computeCost.m` - Function to compute the cost of linear regression

[*] `gradientDescent.m` - Function to run gradient descent

Throughout the exercise, you will be using the script `ex1.m`. This script sets up the dataset for the problems and makes calls to functions that you will write. You do not need to modify `ex1.m`. You are only required to modify functions in other files, by following the instructions in this assignment. For this programming exercise, you are only required to implement linear regression with one variable.

Where to get help

The exercises in this course use Octave[1] or MATLAB, a high-level programming language well-suited for numerical computations. At the Octave/MATLAB command line, typing help followed by a function name displays documentation for a built-in function. For example, `help plot` will bring up help information for plotting. Further documentation for Octave functions can be found at the Octave documentation pages. MATLAB documentation can be found at the MATLAB documentation pages.

---

[*] indicates files you will need to complete

[1] Octave is a free alternative to MATLAB. For the programming exercises, you are free to use either Octave or MATLAB.

1 Simple Octave/MATLAB function

The first part of `ex1.m` gives you practice with Octave/MATLAB syntax. In the file `warmUpExercise.m`, you will find the outline of an Octave/MATLAB function. Modify it to return a 5x5 identity matrix by filling in the following code:

```
A = eye(5);
```

When you are finished, run `ex1.m` (assuming you are in the correct directory, type “`ex1`" at the Octave/MATLAB prompt) and you should see output similar to the following:

```
ans =

Diagonal Matrix

      1  0  0  0  0
      0  1  0  0  0
      0  0  1  0  0
      0  0  0  1  0
      0  0  0  0  1
```

Now `ex1.m` will pause until you press any key, and then will run the code for the next part of the assignment. If you wish to quit, typing `ctrl-c` will stop the program in the middle of its run.

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 `ex1.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 `ex1.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, 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; modify the file and fill in the following code:

```
plot(x, y, 'rx', 'MarkerSize', 10);                             % Plot the data
ylabel('Profit in $10,000s');                            % Set the y-axis label
xlabel('Population of City in 10,000s');                 % Set the x-axis label
```

Now, when you continue to run `ex1.m`, our end result should look like Figure 1, with the same red “x" markers and axis labels.

To learn more about the plot command, you can type `help plot` at the Octave/MATLAB command prompt or to 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’);`)

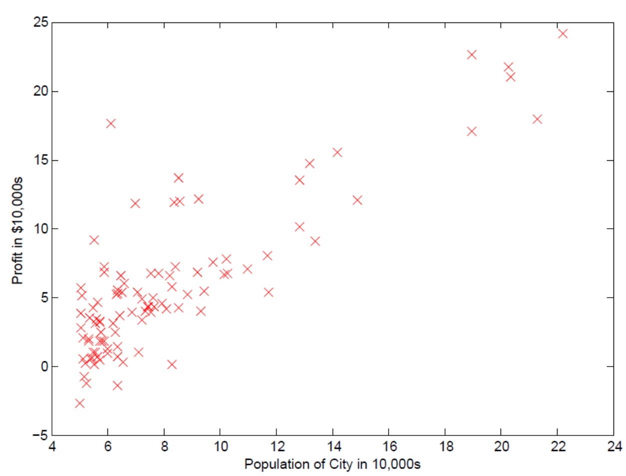

Figure 1: Scatter plot of training data

2.2 Gradient Descent

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

2.2.1 Update Equations

The objective of linear regression is to minimize the cost function

$$J(\theta) = \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

Where the hypothesis h(x) is given by the linear model

$$h(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

Recall that the parameters of your model are the θ_j values. These are the values you will adjust to minimize cost $J(\theta)$. One way to do this is to use the gradient descent algorithm. In gradient descent, each iteration performs the update

$$\theta_j := \theta_j - \alpha. \ 2\sum_{i=1}^{m} \left[\left(h(x^{(i)}) - y^{(i)}\right) \left(x_j^{(i)}\right) \right] \quad \textit{simultaneously update } \theta_j \textit{ for all } j$$

With each step of gradient descent, your parameters θ_j come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

Implementation Note: We store each example as a row in the the X matrix in Octave/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'.

2.2.2 Implementation

In `ex1.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 0.00005.

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

iterations = 1500;
alpha = 0.00005;
```

2.2.3 Computing the cost $J(\theta)$

Your next task is 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 `ex1.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 6222.11.

2.2.4 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. Refer to the equations in this handout and to the video lectures if you are uncertain. After you are finished, `ex1.m` will use your final parameters to plot the linear fit. The result should look something like Figure 2. Your final values for θ

will also be used to make predictions on profits in areas of 35,000 and 70,000 people. Note the way that the following lines in `ex1.m` uses matrix multiplication, rather than explicit summation or looping, to calculate the predictions. This is an example of code vectorization in Octave/MATLAB.

```
predict1 = [1, 3.5] * theta;
predict2 = [1, 7] * theta;
```

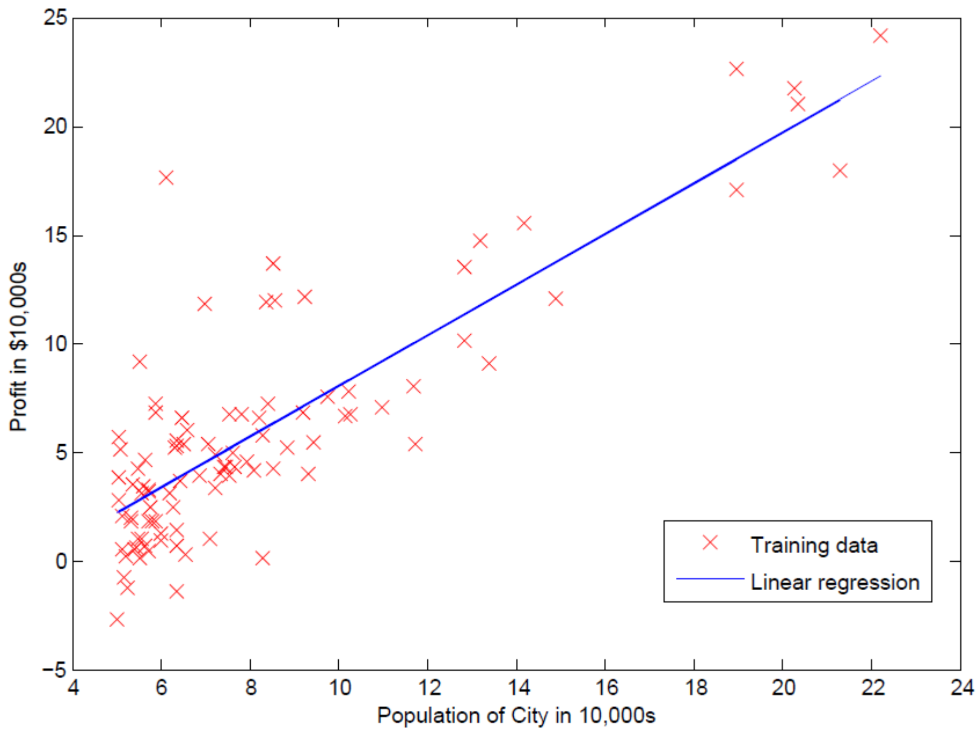

Figure 2: Training data with linear regression fit

2.3 Debugging

Here are some things to keep in mind as you implement gradient descent:

- Octave/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, Octave/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 Octave/MATLAB. For example, A*B does a matrix multiply, while A.*B does an element-wise multiplication.

2.4 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 `ex1.m`, there is code set up to calculate $J(\theta)$ 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 $J(\theta)$ values. The script `ex1.m` will then use these values to produce surface of $J(\theta)$ using the `surf` command. The plot should look something like Figure 3.

The purpose of this graph is to show you that how $J(\theta)$ varies with changes in θ_0 and θ_1. The cost function $J(\theta)$ is bowl-shaped and has a global minimum. This minimum is the optimal point for θ_0 and θ_1, and each step of gradient descent moves closer to this point.

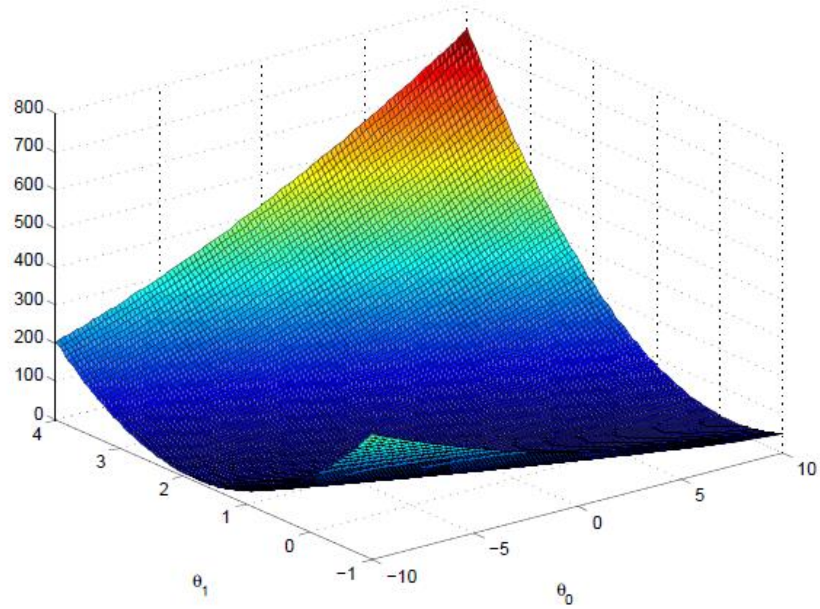

Figure 3: Cost function $J(\theta)$