Programming Exercise 1: Linear regression

In this exercise, you will implement linear regression and see how it works. For working on this exercise, you will be given a starter code. You may need to use `cd` command to change to the right directory.

Following files are either complete or partially complete. If it is partially complete, you will have to complete it.

```
ex1.m                %%complete
warmUpExercise.m     %% partially complete
plotData.m           %% partially complete
computeCost.m        %% partially complete
gradientDescent.m    %% partially complete
```

You will be running `ex1.m`. This file is complete. You will not need to do any modification to this file. `ex1.m` set up the dataset for the problem and make calls to functions that you will have to write. Do not modify `ex1.m`. However, you will have to modify functions in other files according to the instructions that will be provided later in this document.

1. Running a simple Ocave/MATLAB function

At the very beginning of `ex1.m`, you get to practice with MATLAB/Octave. In the file `warmUpExercise.m`, you will find the outline of a function. You should modify it so that the function returns a 10X10 identity matrix.

[hint: the command for identity matrix is `eye(n)`, this will return a `nxn` identity matrix]

When you are done modifying the `warmUpExercise.m` file, you will run the `ex1.m` file and you should see the output similar to the following.:

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

2. Plotting Data

The next file you need to modify is `plotData.m`. Actual loading of the data is already done in `ex1.m` file. You are given the data in file `ex1data1.txt`. `ex1.m` also calls the function for plotting the data

which is `plotData.m`. This data file has *m* lines of data, each with two numbers separated by comma. The first number before comma is the population of a city in 10,000s and the second number is the profit from a food truck in that city in $10,000s.

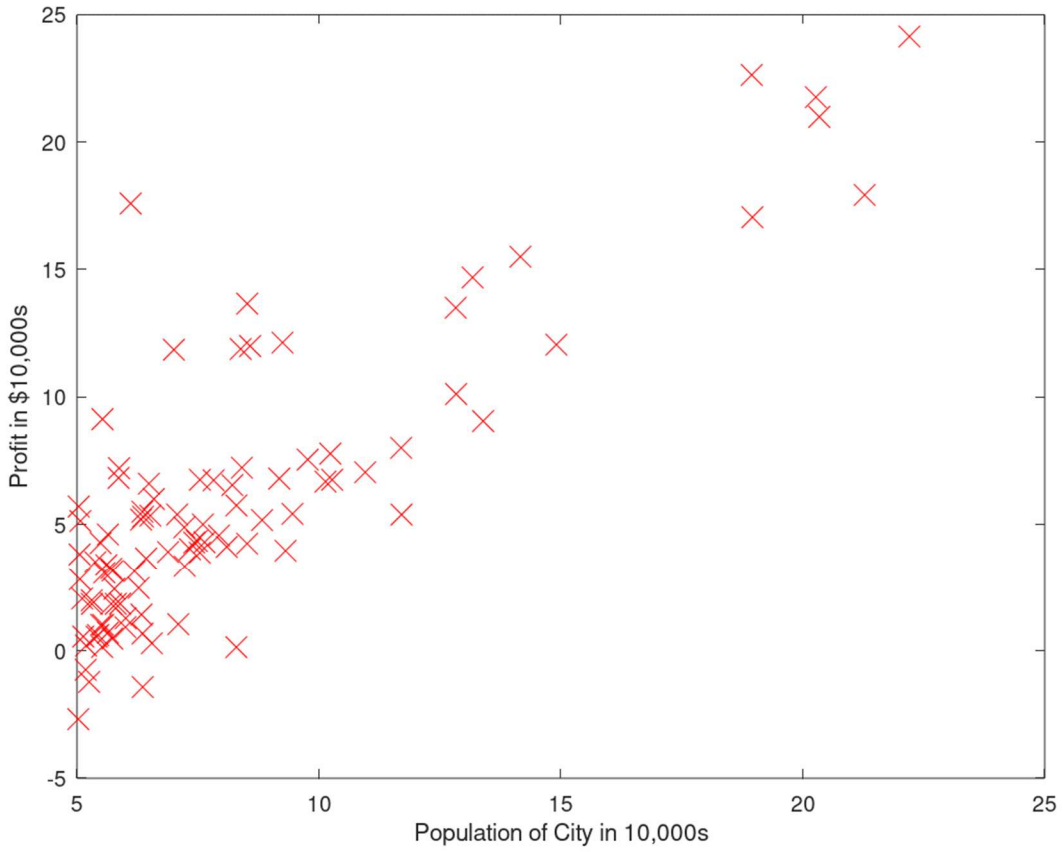
Dataset is loaded using the following commands in `ex1.m`:

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

These *m* lines of data are training data. Your job is to plot these training data using plot command. You should also use the `xlabel` and `ylabel` commands to explain the data in X axis and Y axis. Make sure the markers appear in red crosses in the plot.

You can assume that population and profit data have been passed in as that x and y arguments of this function. The plot should look like below. You should also be able to answer the following question.

How many city's data are available to us from the training dataset?



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. The chain restaurant already has trucks in various cities, and you have data for profits and populations from these cities. As a CEO of this company, you would like to use this data to help you select which city to expand to next.

`ex1data1.txt` contains 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.

Gradient Descent Theory

In this part, you will fit the linear regression parameters θ to our dataset using gradient descent. Your goal is to find θ through gradient descent.

Update equations:

The objective of linear regression is to minimize the cost function

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

where the hypothesis $h_\theta(x)$is given by the linear model

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

Remember 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 batch gradient descent algorithm. In batch gradient descent, each iteration performs the update

$$\theta_j = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(you have to update θ_j simultaneously 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

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 0.01.

```
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;
```

3. Computing the cost $J(\theta)$

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.

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 32.07. Also, for θ = [- 1 2], expected cost is approximately 54.24.

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 if you are uncertain.

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, `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;
```

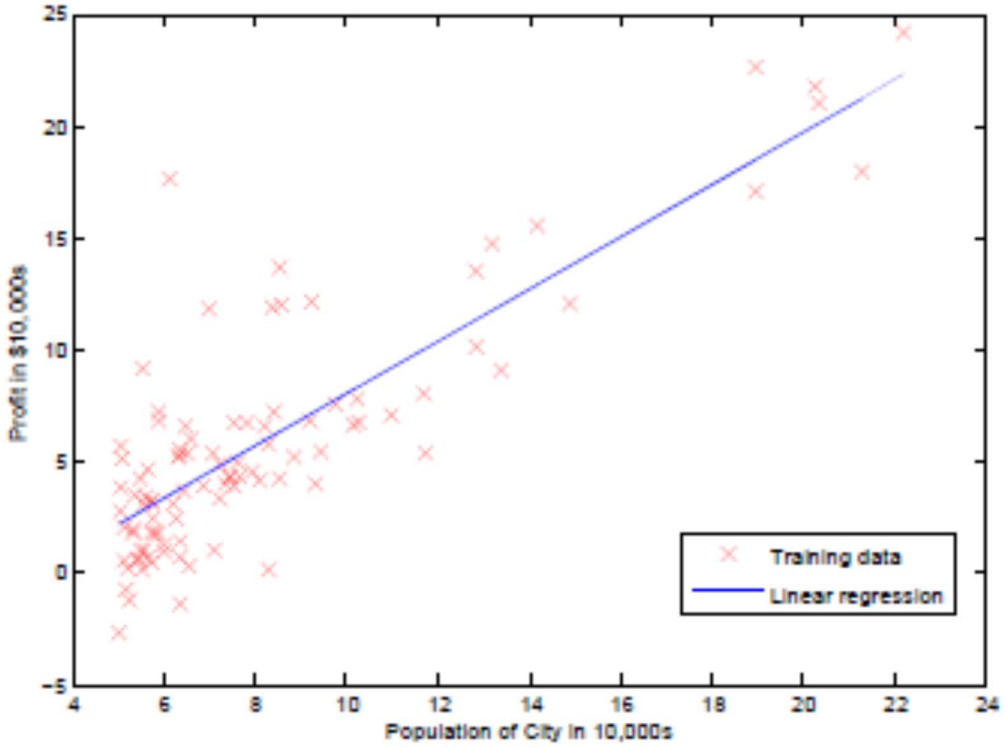


Figure 2: Training data with linear regression fit

Distribution of points

Finish in the class, for 100% points.
Submit by midnight today, for 92% of points.
Submit by midnight tomorrow, for 88% of points.
Submit by midnight day after tomorrow, for 84% of points.

Problem	**Points**
`WarmUpExercise`	3
`PlotData`	6
`ComputeCost`	10
`GradientDescent`	16