

Problem Set 1: Linear Regression¹

See website for due date

Submission

What to submit (staple in this order):

1. *grade1.txt
2. ex1.m: Octave script that will help step you through the exercise
3. *warmUpExercise.m: Simple warm up exercise for Octave
4. *plotData.m: Function to display the dataset
5. *computeCost.m: Function to compute the cost of linear regression
6. *gradientDescent.m: Function to run gradient descent
7. solutions to written exercises

* complete and print out code, include screenshots of the output

Grading. We will randomly choose a subset of submissions and verify that 1) the code produces the answers and 2) the points computation in “grade.txt” is correct. Only one of the written problems will be chosen at random for grading. Total: 130 points.

0. Octave Installation

Follow the instruction at

<https://sites.google.com/site/umlcsmachinelearningspring2014/octave/octave-installation> to install Octave on your machine. All the code should be compatible with Matlab and you may use Matlab instead.

1. Octave tutorial

1.1 Modify **warmUpExercise.m** to return a 5x5 matrix of ones.

Hint: This exercise is pretty straightforward, all you need to do is adding “A = ones(5);” in the blank area. However, whenever you were asked to add some code do remember: (1) add your code ONLY to the blank area that is marked as “YOUR CODE HERE”; (2) DO NOT modify any code elsewhere.

¹ Note: this assignment is partially adapted from programming assignment 1 of Andrew Ng’s online Machine Learning course(<https://class.coursera.org/ml>).

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

```
Running warmUpExercise ...  
5x5 matrix of ones:
```

```
ans =
```

```
1 1 1 1 1  
1 1 1 1 1  
1 1 1 1 1  
1 1 1 1 1  
1 1 1 1 1
```

```
Program paused. Press enter to continue.
```

[5 points for getting this answer]

1.2 Let A be a 10×10 matrix and x be a 10-element column vector. Your friend wants to compute the product Ax and writes the following code:

```
v = zeros(10, 1);  
for i = 1:10  
    for j = 1:10  
        v(i) = v(i) + A(i, j) * x(j);  
    end  
end
```

How would you vectorize this code to run without any for loops? Check all that apply.

- A. $v = x' * A;$
- B. $v = Ax;$
- C. $v = A * x;$
- D. $v = \text{sum}(A * x);$

You can try all of the above in the Octave prompt and get the correct answer.

[5 points for getting the correct answer](We did not give away the correct answer, but you should easily find it by trying all of them in the Octave prompt)

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 `ex1data.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('ex1data.txt');           % read comma separated data
X = data(:, 1); y = data(:, 2);
m = length(y);                         % number of training examples
```

Next, the script calls the script `plotData` function to create a scatter plot of the data. Your job is to complete `plotData.m` to draw the plot.

Hint: All you need to do is modifying 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`, your end result should look like Figure 1, with the same red “x” markers and axis labels.

[10 points for getting this plot]

To learn more about the `plot` command, you can type “`help plot`” at the Octave 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');`)

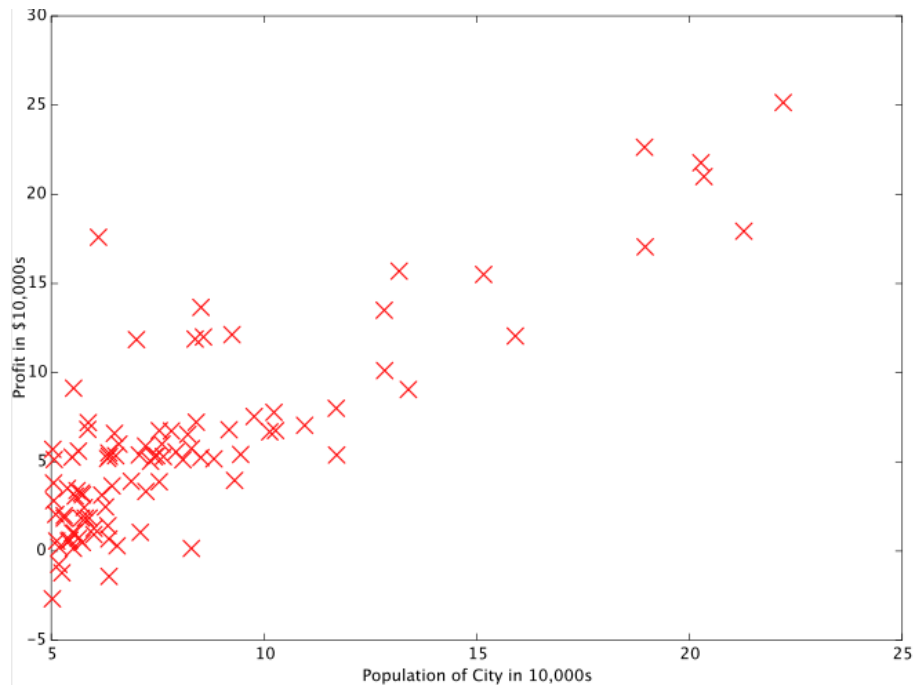


Figure 1. Scatter plot of training data

2.2 Gradient Descent

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

2.2.1 Update Equations

The objective of linear regression is to minimize the cost function

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

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

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

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

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

With each step of gradient descent, your parameter 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 X matrix in Octave. 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 α 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;
```

2.2.3 Computing the cost $J(\theta)$

As you perform gradient descent to learn to 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 **approximately 33.04**. [30 points for getting this answer]

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 .

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:

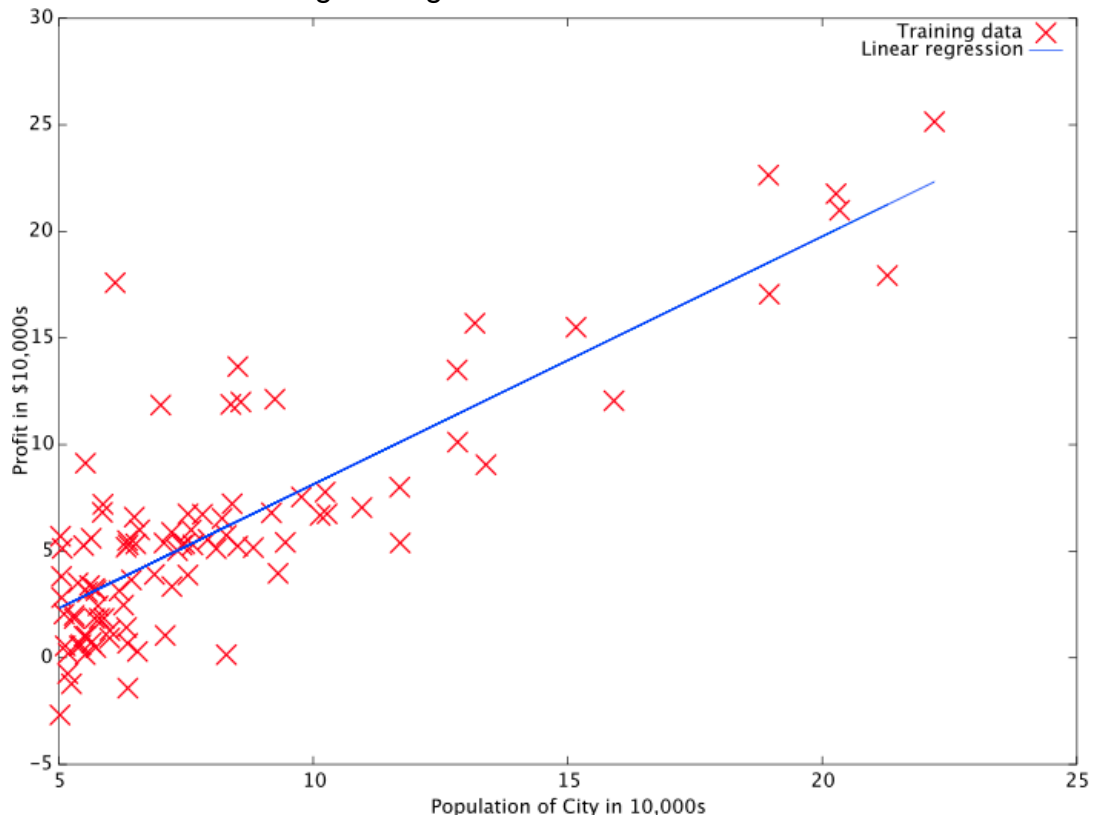


Figure 2. Training data with linear regression fit

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.

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

You should expect to see **theta approximately [-3.497; 1.163]**. [40 points for getting this answer]

2.3 Debugging

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

- Octave 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 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. 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 and contour plots of $J(\theta)$ using the `surf` and `contour` commands. The plots should look something like Figure 3a and 3b:

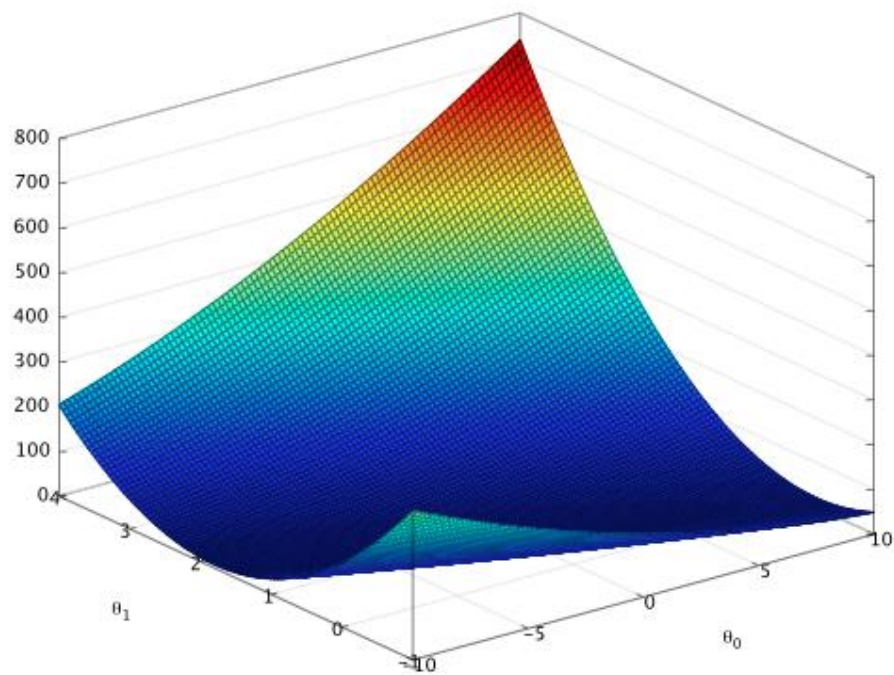


Figure 3a: Surface plot of cost function $J(\theta)$

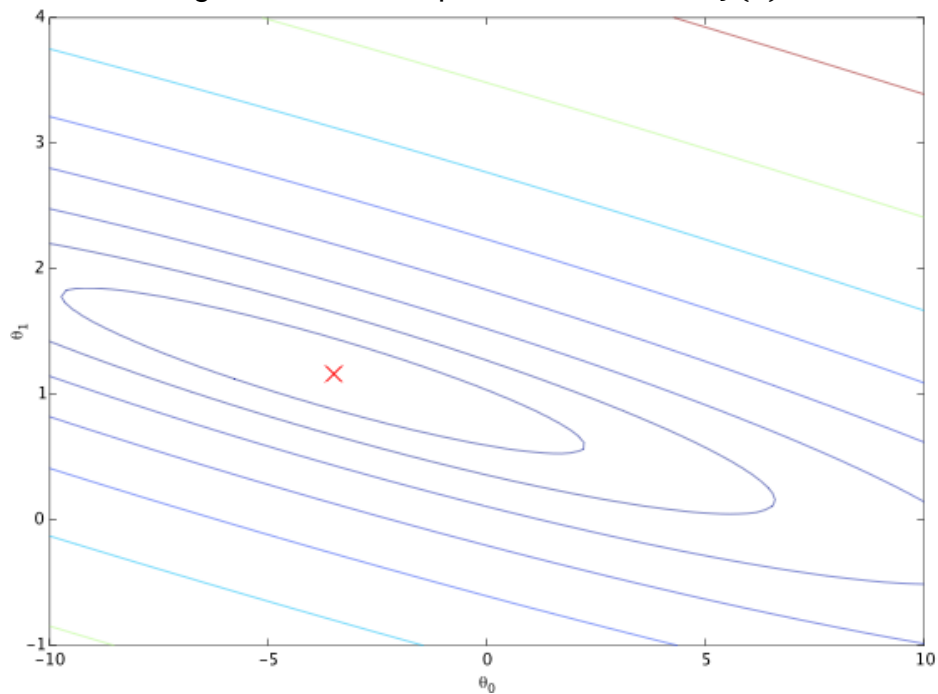


Figure 3b: Contour plot of cost function $J(\theta)$, showing minimum
[10 points for getting these two plots]

The purpose of these graphs is to show you 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 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.

3. Linear regression with multiple variables (optional, ungraded)

1. Why is feature normalization important when features differ by orders of magnitude?
2. Modify your code of linear regression with one variable to support any number of features (vectorize your code.)
3. How can we choose an appropriate learning rate? See what will happen if the learning rate is too small or too large.

4. Written problems

[30 points; we will randomly pick one problem and assign points]

4.1 Maximum Likelihood Estimate for Coin Toss

The probability distribution of a single binary variable $x \in \{0,1\}$ that takes value 1 with probability μ is given by the *Bernoulli* distribution

$$\text{Bern}(x|\mu) = \mu^x(1 - \mu)^{1-x}$$

For example, we can use it to model the probability of seeing ‘heads’ ($x = 1$) or ‘tails’ ($x = 0$) after tossing a coin, with μ being the probability of seeing ‘heads’. Suppose we have a dataset of independent coin flips $D = \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ and we would like to estimate μ using Maximum Likelihood. Recall that we can write down the likelihood function as

$$p(D|\mu) = \prod_{i=1}^m p(x^{(i)}|\mu) = \prod_{i=1}^m \mu^{x^{(i)}}(1 - \mu)^{1-x^{(i)}}$$

The log of the likelihood function is

$$\ln p(D|\mu) = \sum_{i=1}^m x^{(i)} \ln \mu + (1 - x^{(i)}) \ln(1 - \mu)$$

Show that the ML solution for μ is given by $\mu_{ML} = \frac{h}{m}$, where h is the total number of ‘heads’ in the dataset. Show all of your steps.

4.2 Localized linear regression

Suppose we want to estimate *localized* linear regression by weighting the contribution of the data points by their distance to the query point $x^{(q)}$, i.e. using the cost

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

where $(x^{(i)} - x^{(q)})^{-2} = (w^{(i)})^2$ is the inverse Euclidean distance between the training point $x^{(i)}$ and query (test) point $x^{(q)}$.

Derive the modified normal equations for the above cost function $E(x^{(q)})$. (*Hint: first, rewrite the cost function in matrix/vector notation, using a diagonal matrix to represent the weights $w^{(i)}$*).

4.3 Betting on Trick Coins

A game is played with three coins in a jar: one is a normal coin, one has “heads” on both sides, one has “tails” on both sides. All coins are “fair”, i.e. have equal probability of landing on either side. Suppose one coin is picked randomly from jar and tossed, and lands with “heads” on top. What is the probability that the bottom side is also “heads”? Show all your steps.