Machine Learning: Programming Exercise 1

Linear Regression

In this exercise, you will implement linear regression and get to see it work on data.

Files needed for this exercise

- `ex1.mlx` - MATLAB Live Script that steps you through the exercise
- ex1data1.txt - Dataset for linear regression with one variable
- ex1data2.txt - Dataset for linear regression with multiple variables
- `plotData.m` - Function to display the dataset
- `computeCost.m` - Function to compute the cost of linear regression
- `gradientDescent.m` - Function to run gradient descent
- `featureNormalize.m` - Function to normalize features

Confirm that your Current Folder is set correctly

Click into this section, then click the 'Run section' button above. This will execute the `dir` command below to list the files in your Current Folder. The output should contain all of the files listed above and the 'lib' folder. If it does not, right-click the 'ex1' folder and select 'Open' before proceding or see the instructions in `README.mlx` for more details.

```
dir
```

```
.                        computeCost.m            ex1.pdf                  ex1data2.txt             gradientDe
..                       computeCostMulti.m       ex1_companion.mlx        featureNormalize.m       lib
Copy_of_ex1.mlx          ex1.mlx                  ex1data1.txt             gradientDescent.m        normalEqn.
```

Before you begin

The workflow for completing and submitting the programming exercises in MATLAB Online differs from the original course instructions. Before beginning this exercise, make sure you have read through the instructions in `README.mlx` which is included with the programming exercise files. `README.mlx` also contains solutions to the many common issues you may encounter while completing and submitting the exercises in MATLAB Online. Make sure you are following instructions in `README.mlx` and have checked for an existing solution before seeking help on the discussion forums.

Table of Contents

1. A simple MATLAB function

The first part of this script gives you practice with MATLAB syntax and the homework submission process. In the file `warmUpExercise.m`, you will find the outline of a MATLAB function. Modify it to return a 5 x 5 identity matrix by filling in the following code:

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

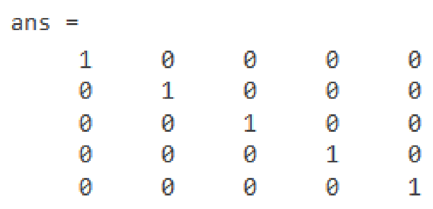
When you are finished, save `warmUpExercise.m`, then run the code contained in this section to call `warmUpExercise().`

5x5 Identity Matrix:

```
warmUpExercise()
```

```
ans = 5×5
     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
```

You should see output similar to the following:



```
ans =

     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
```

You can toggle between right-hand-side output and in-line output for printing results and displaying figures inside a Live Script by selecting the appropriate box in the upper right of the Live Editor window.

1.1 Submitting Solutions

After completing a part of the exercise, you can submit your solutions for that section by running the section of code below, which calls the `submit.m` script. Your score for each section will then be displayed as output. **Enter your login and your unique submission token *inside the command window* when prompted. For**

future submissions of this exercise, you will only be asked to confirm your credentials. Your submission token for each exercise is found in the corresponding homework assignment course page. New tokens can be generated if you are experiencing issues with your current token. You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.

You should now submit your solutions. Enter **`submit`** *at the command prompt, then enter your login and token when prompted.*

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. This 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.

Run the code below to load the dataset from the data file into the variables X and y:

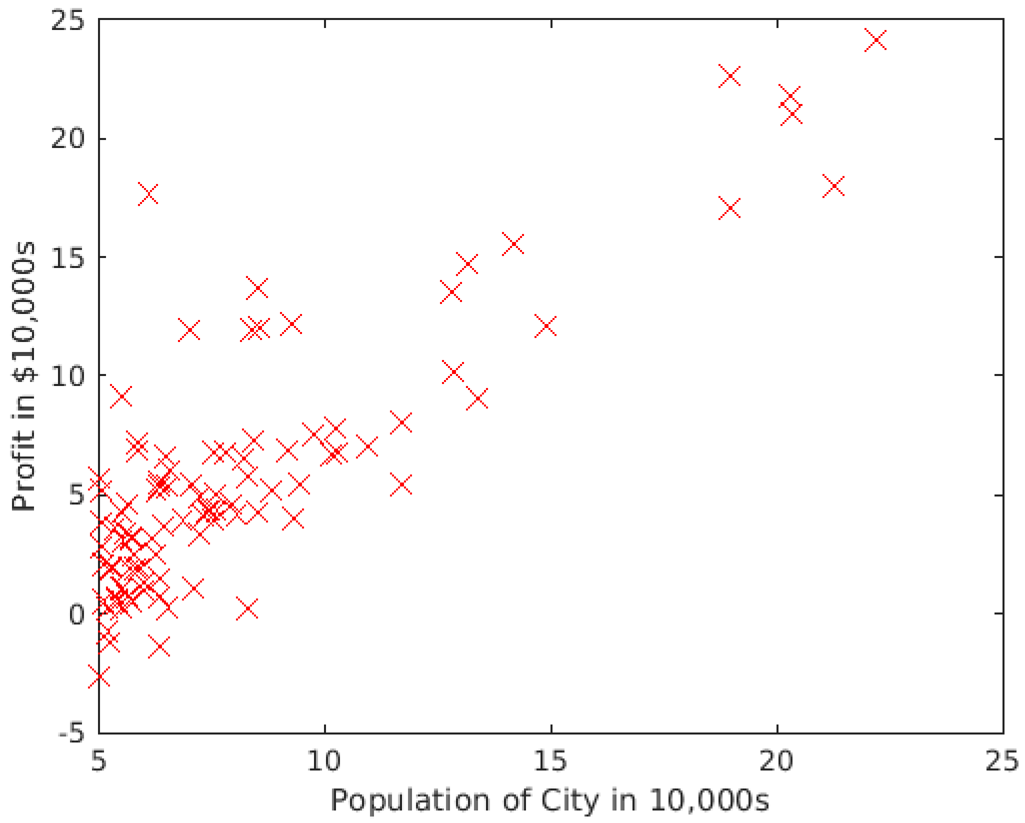
```
data = load('ex1data1.txt'); % read comma separated data
X = data(:, 1); y = data(:, 2);
```

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
```

Once you are finished, save `plotData.m,` and execute the code in this section which will call `plotData`.

```
plotData(X,y)
```



The resulting plot should appear as in Figure 1 below:

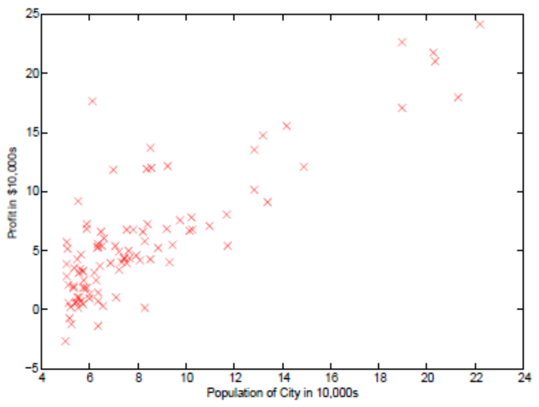


Figure 1: Scatter plot of training data

To learn more about the `plot` command, you can run the command **`help plot`** at the command prompt, type `plot()` inside the MATLAB Live Editor and click on the "(?)" tooltip, or you can search the MATLAB documentation for "plot". Note that to change the markers to red x's in the plot, we used the option: **`'rx'`** together with the **`plot`** command, i.e.,

```
plot(..,[your options here],..,'rx');
```

2.2 Gradient Descent

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

2.2.1 Update Equations

The objective of linear regression is to minimize the cost function

$$J(\theta) = \frac{1}{2m}\sum\nolimits_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

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 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

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum\nolimits_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \qquad (\text{simultaneously update } \theta_j \text{ for all } j)$$

With each step of gradient descent, your parameters *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 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 this script, 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. Run the code below to initialize the parameters to 0 and the learning rate `alpha` to 0.01.

```
m = length(X) % number of training examples
```

```
m = 97
```

```
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 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 definition, run this section. The code below will call `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 for the first output below:

```
% Compute and display initial cost with theta all zeros
computeCost(X, y, theta)
```

```
ans = 32.0727
```

Next we call `computeCost` again, this time with non-zero `theta` values as an additional test. You should expect to see an output of 54.24 below:

```
% Compute and display initial cost with non-zero theta
computeCost(X, y,[-1; 2])
```

```
ans = 54.2425
```

If the outputs above match the expected values, you can submit your solution for assessment. If the outputs do not match or you receive an error, check your cost function for mistakes, then rerun this section once you have addressed them.

You should now submit your solutions. Enter **`submit`** *at the command prompt, then enter or confirm your login and token when prompted.*

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 given earlier and to the video lectures if you are uncertain.

A good way to verify that gradient descent is working correctly is to look at the value of J 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, run this execute this section. The code below will use your final parameters to plot the linear fit. The result should look something like Figure 2 below:

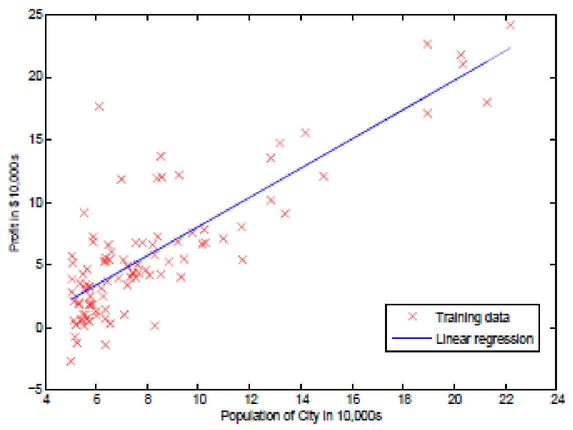


Figure 2: Training data with linear regression fit

Your final values for θ will also be used to make predictions on profits in areas of 35,000 and 70,000 people.

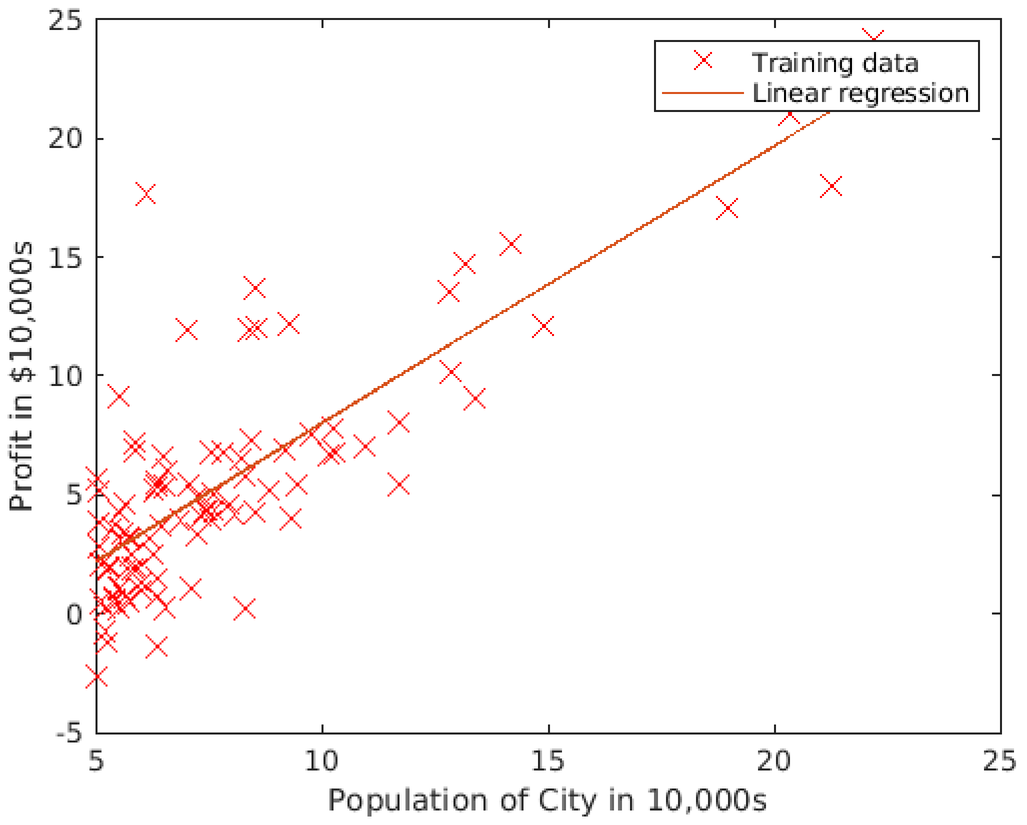
```
% Run gradient descent:
% Compute theta
theta = gradientDescent(X, y, theta, alpha, iterations);
```

```
ans = 6.7372
ans = 5.9316
ans = 5.9012
ans = 5.8901
ans = 5.8850
ans = 5.8799
ans = 5.8749
ans = 5.8698
ans = 5.8648
ans = 5.8598
ans = 5.8548
ans = 5.8499
ans = 5.8449
ans = 5.8400
ans = 5.8351
ans = 5.8302
ans = 5.8253
ans = 5.8205
ans = 5.8156
ans = 5.8108
ans = 5.8060
ans = 5.8012
ans = 5.7965
ans = 5.7917
ans = 5.7870
ans = 5.7822
ans = 5.7775
ans = 5.7729
ans = 5.7682
ans = 5.7635
ans = 5.7589
ans = 5.7543
ans = 5.7497
ans = 5.7451
ans = 5.7405
ans = 5.7360
ans = 5.7315
ans = 5.7269
ans = 5.7224
ans = 5.7179
ans = 5.7135
ans = 5.7090
ans = 5.7046
ans = 5.7002
ans = 5.6958
ans = 5.6914
ans = 5.6870
ans = 5.6826
ans = 5.6783
ans = 5.6740
ans = 5.6697
ans = 5.6654
ans = 5.6611
ans = 5.6568
ans = 5.6526
ans = 5.6483
ans = 5.6441
ans = 5.6399
ans = 5.6357
ans = 5.6315
ans = 5.6274
ans = 5.6232
ans = 5.6191
ans = 5.6150
ans = 5.6109
ans = 5.6068
ans = 5.6027
ans = 5.5987
ans = 5.5946
ans = 5.5906
ans = 5.5866
ans = 5.5826
ans = 5.5786
ans = 5.5747
ans = 5.5707
ans = 5.5668
ans = 5.5628
ans = 5.5589
ans = 5.5550
ans = 5.5512
ans = 5.5473
ans = 5.5434
ans = 5.5396
ans = 5.5358
ans = 5.5319
ans = 5.5281
ans = 5.5244
ans = 5.5206
ans = 5.5168
ans = 5.5131
ans = 5.5094
ans = 5.5056
ans = 5.5019
ans = 5.4982
ans = 5.4946
ans = 5.4909
ans = 5.4872
ans = 5.4836
ans = 5.4800
ans = 5.4764
ans = 5.4728
ans = 5.4692
ans = 5.4656
ans = 5.4620
ans = 5.4585
ans = 5.4550
ans = 5.4514
ans = 5.4479
ans = 5.4444
ans = 5.4409
ans = 5.4375
ans = 5.4340
ans = 5.4306
ans = 5.4271
ans = 5.4237
ans = 5.4203
ans = 5.4169
ans = 5.4135
ans = 5.4101
ans = 5.4068
ans = 5.4034
ans = 5.4001
ans = 5.3968
ans = 5.3935
ans = 5.3902
ans = 5.3869
ans = 5.3836
ans = 5.3803
ans = 5.3771
ans = 5.3738
ans = 5.3706
ans = 5.3674
ans = 5.3642
ans = 5.3610
ans = 5.3578
ans = 5.3546
ans = 5.3515
ans = 5.3483
ans = 5.3452
ans = 5.3420
ans = 5.3389
ans = 5.3358
ans = 5.3327
ans = 5.3296
ans = 5.3266
ans = 5.3235
ans = 5.3205
ans = 5.3174
ans = 5.3144
ans = 5.3114
ans = 5.3084
ans = 5.3054
ans = 5.3024
ans = 5.2994
ans = 5.2965
ans = 5.2935
ans = 5.2906
ans = 5.2876
ans = 5.2847
ans = 5.2818
ans = 5.2789
ans = 5.2760
ans = 5.2731
ans = 5.2703
ans = 5.2674
ans = 5.2646
ans = 5.2617
ans = 5.2589
ans = 5.2561
ans = 5.2533
ans = 5.2505
ans = 5.2477
ans = 5.2449
ans = 5.2422
ans = 5.2394
ans = 5.2367
ans = 5.2339
ans = 5.2312
ans = 5.2285
ans = 5.2258
ans = 5.2231
ans = 5.2204
ans = 5.2177
ans = 5.2150
ans = 5.2124
ans = 5.2097
ans = 5.2071
ans = 5.2045
ans = 5.2018
ans = 5.1992
ans = 5.1966
ans = 5.1940
ans = 5.1915
ans = 5.1889
ans = 5.1863
ans = 5.1838
ans = 5.1812
ans = 5.1787
ans = 5.1762
ans = 5.1736
ans = 5.1711
ans = 5.1686
ans = 5.1661
ans = 5.1637
ans = 5.1612
ans = 5.1587
ans = 5.1563
ans = 5.1538
ans = 5.1514
ans = 5.1489
ans = 5.1465
ans = 5.1441
ans = 5.1417
ans = 5.1393
ans = 5.1369
ans = 5.1346
ans = 5.1322
ans = 5.1298
ans = 5.1275
ans = 5.1251
ans = 5.1228
ans = 5.1205
ans = 5.1181
ans = 5.1158
ans = 5.1135
ans = 5.1112
ans = 5.1090
ans = 5.1067
ans = 5.1044
ans = 5.1022
ans = 5.0999
ans = 5.0977
ans = 5.0954
ans = 5.0932
ans = 5.0910
ans = 5.0888
ans = 5.0866
ans = 5.0844
ans = 5.0822
ans = 5.0800
ans = 5.0778
ans = 5.0757
ans = 5.0735
ans = 5.0714
ans = 5.0692
ans = 5.0671
ans = 5.0650
ans = 5.0628
ans = 5.0607
ans = 5.0586
ans = 5.0565
ans = 5.0544
ans = 5.0524
ans = 5.0503
ans = 5.0482
ans = 5.0462
ans = 5.0441
ans = 5.0421
ans = 5.0400
ans = 5.0380
ans = 5.0360
ans = 5.0340
ans = 5.0320
ans = 5.0300
ans = 5.0280
ans = 5.0260
ans = 5.0240
ans = 5.0220
ans = 5.0201
ans = 5.0181
ans = 5.0162
ans = 5.0142
ans = 5.0123
ans = 5.0104
ans = 5.0085
ans = 5.0065
ans = 5.0046
ans = 5.0027
ans = 5.0008
ans = 4.9989
ans = 4.9971
ans = 4.9952
ans = 4.9933
ans = 4.9915
ans = 4.9896
ans = 4.9878
ans = 4.9859
ans = 4.9841
ans = 4.9823
ans = 4.9805
ans = 4.9786
ans = 4.9768
ans = 4.9750
ans = 4.9732
ans = 4.9714
ans = 4.9697
ans = 4.9679
ans = 4.9661
ans = 4.9644
ans = 4.9626
ans = 4.9609
ans = 4.9591
ans = 4.9574
ans = 4.9556
ans = 4.9539
ans = 4.9522
ans = 4.9505
ans = 4.9488
ans = 4.9471
ans = 4.9454
ans = 4.9437
ans = 4.9420
ans = 4.9404
ans = 4.9387
ans = 4.9370
ans = 4.9354
ans = 4.9337
ans = 4.9321
ans = 4.9304
ans = 4.9288
ans = 4.9272
ans = 4.9255
ans = 4.9239
ans = 4.9223
ans = 4.9207
ans = 4.9191
ans = 4.9175
ans = 4.9159
ans = 4.9144
ans = 4.9128
ans = 4.9112
ans = 4.9096
ans = 4.9081
ans = 4.9065
ans = 4.9050
ans = 4.9034
ans = 4.9019
ans = 4.9004
ans = 4.8989
ans = 4.8973
ans = 4.8958
ans = 4.8943
ans = 4.8928
ans = 4.8913
ans = 4.8898
ans = 4.8883
ans = 4.8868
ans = 4.8854
ans = 4.8839
ans = 4.8824
ans = 4.8810
ans = 4.8795
ans = 4.8781
ans = 4.8766
ans = 4.8752
ans = 4.8738
ans = 4.8723
ans = 4.8709
ans = 4.8695
ans = 4.8681
ans = 4.8667
ans = 4.8653
ans = 4.8639
ans = 4.8625
ans = 4.8611
ans = 4.8597
ans = 4.8583
ans = 4.8569
ans = 4.8556
ans = 4.8542
ans = 4.8528
ans = 4.8515
ans = 4.8501
ans = 4.8488
ans = 4.8475
ans = 4.8461
ans = 4.8448
ans = 4.8435
ans = 4.8422
ans = 4.8408
ans = 4.8395
ans = 4.8382
ans = 4.8369
ans = 4.8356
ans = 4.8343
ans = 4.8330
ans = 4.8318
ans = 4.8305
ans = 4.8292
ans = 4.8279
ans = 4.8267
ans = 4.8254
ans = 4.8242
ans = 4.8229
ans = 4.8217
ans = 4.8204
ans = 4.8192
ans = 4.8180
ans = 4.8167
ans = 4.8155
ans = 4.8143
ans = 4.8131
ans = 4.8119
ans = 4.8107
ans = 4.8094
ans = 4.8083
ans = 4.8071
ans = 4.8059
ans = 4.8047
ans = 4.8035
ans = 4.8023
ans = 4.8012
ans = 4.8000
ans = 4.7988
ans = 4.7977
ans = 4.7965
ans = 4.7954
ans = 4.7942
ans = 4.7931
ans = 4.7919
ans = 4.7908
ans = 4.7897
ans = 4.7885
ans = 4.7874
ans = 4.7863
ans = 4.7852
ans = 4.7841
ans = 4.7830
ans = 4.7819
ans = 4.7808
ans = 4.7797
ans = 4.7786
ans = 4.7775
ans = 4.7764
ans = 4.7753
ans = 4.7743
ans = 4.7732
ans = 4.7721
ans = 4.7711
ans = 4.7700
ans = 4.7689
ans = 4.7679
ans = 4.7668
ans = 4.7658
ans = 4.7648
ans = 4.7637
ans = 4.7627
ans = 4.7617
ans = 4.7606
ans = 4.7596
ans = 4.7586
ans = 4.7576
ans = 4.7566
ans = 4.7556
ans = 4.7546
ans = 4.7536
ans = 4.7526
ans = 4.7516
ans = 4.7506
ans = 4.7496
ans = 4.7486
ans = 4.7476
ans = 4.7467
ans = 4.7457
ans = 4.7447
ans = 4.7438
ans = 4.7428
ans = 4.7418
ans = 4.7409
ans = 4.7399
ans = 4.7390
ans = 4.7380
ans = 4.7371
ans = 4.7362
ans = 4.7352
ans = 4.7343
ans = 4.7334
ans = 4.7325
ans = 4.7315
ans = 4.7306
ans = 4.7297
ans = 4.7288
ans = 4.7279
ans = 4.7270
ans = 4.7261
ans = 4.7252
ans = 4.7243
ans = 4.7234
ans = 4.7225
ans = 4.7216
ans = 4.7207
ans = 4.7199
ans = 4.7190
ans = 4.7181
ans = 4.7173
ans = 4.7164
ans = 4.7155
ans = 4.7147
ans = 4.7138
ans = 4.7130
ans = 4.7121
ans = 4.7113
ans = 4.7104
ans = 4.7096
ans = 4.7087
ans = 4.7079
ans = 4.7071
ans = 4.7062
ans = 4.7054
ans = 4.7046
ans = 4.7038
ans = 4.7030
ans = 4.7021
ans = 4.7013
ans = 4.7005
ans = 4.6997
ans = 4.6989
ans = 4.6981
ans = 4.6973
ans = 4.6965
ans = 4.6957
ans = 4.6949
ans = 4.6942
ans = 4.6934
ans = 4.6926
ans = 4.6918
ans = 4.6910
ans = 4.6903
ans = 4.6895
ans = 4.6887
ans = 4.6880
ans = 4.6872
ans = 4.6865
ans = 4.6857
ans = 4.6850
ans = 4.6842
ans = 4.6835
ans = 4.6827
ans = 4.6820
ans = 4.6812
ans = 4.6805
ans = 4.6798
ans = 4.6790
ans = 4.6783
ans = 4.6776
ans = 4.6769
ans = 4.6761
ans = 4.6754
ans = 4.6747
ans = 4.6740
ans = 4.6733
ans = 4.6726
ans = 4.6719
ans = 4.6712
ans = 4.6705
ans = 4.6698
ans = 4.6691
ans = 4.6684
ans = 4.6677
ans = 4.6670
ans = 4.6663
ans = 4.6656
ans = 4.6650
ans = 4.6643
ans = 4.6636
ans = 4.6629
ans = 4.6623
ans = 4.6616
ans = 4.6609
ans = 4.6603
ans = 4.6596
ans = 4.6590
ans = 4.6583
ans = 4.6577
ans = 4.6570
ans = 4.6564
ans = 4.6557
ans = 4.6551
ans = 4.6544
ans = 4.6538
ans = 4.6531
ans = 4.6525
ans = 4.6519
ans = 4.6513
ans = 4.6506
ans = 4.6500
ans = 4.6494
ans = 4.6488
ans = 4.6481
ans = 4.6475
ans = 4.6469
ans = 4.6463
ans = 4.6457
ans = 4.6451
ans = 4.6445
ans = 4.6439
ans = 4.6433
ans = 4.6427
ans = 4.6421
ans = 4.6415
ans = 4.6409
ans = 4.6403
ans = 4.6397
ans = 4.6391
ans = 4.6385
ans = 4.6380
ans = 4.6374
ans = 4.6368
ans = 4.6362
ans = 4.6356
ans = 4.6351
ans = 4.6345
ans = 4.6339
ans = 4.6334
ans = 4.6328
ans = 4.6322
ans = 4.6317
ans = 4.6311
ans = 4.6306
ans = 4.6300
ans = 4.6295
ans = 4.6289
ans = 4.6284
ans = 4.6278
ans = 4.6273
ans = 4.6267
ans = 4.6262
ans = 4.6257
ans = 4.6251
ans = 4.6246
ans = 4.6241
ans = 4.6235
ans = 4.6230
ans = 4.6225
ans = 4.6220
ans = 4.6214
ans = 4.6209
ans = 4.6204
ans = 4.6199
ans = 4.6194
ans = 4.6189
ans = 4.6183
ans = 4.6178
ans = 4.6173
ans = 4.6168
ans = 4.6163
ans = 4.6158
ans = 4.6153
ans = 4.6148
ans = 4.6143
ans = 4.6138
ans = 4.6133
ans = 4.6128
ans = 4.6123
ans = 4.6119
ans = 4.6114
ans = 4.6109
ans = 4.6104
ans = 4.6099
ans = 4.6094
ans = 4.6090
ans = 4.6085
ans = 4.6080
ans = 4.6076
ans = 4.6071
ans = 4.6066
ans = 4.6061
ans = 4.6057
ans = 4.6052
ans = 4.6048
ans = 4.6043
ans = 4.6038
ans = 4.6034
ans = 4.6029
ans = 4.6025
ans = 4.6020
ans = 4.6016
ans = 4.6011
ans = 4.6007
ans = 4.6002
ans = 4.5998
ans = 4.5993
ans = 4.5989
ans = 4.5985
ans = 4.5980
ans = 4.5976
ans = 4.5972
ans = 4.5967
ans = 4.5963
ans = 4.5959
ans = 4.5954
ans = 4.5950
ans = 4.5946
ans = 4.5942
ans = 4.5937
ans = 4.5933
ans = 4.5929
ans = 4.5925
ans = 4.5921
ans = 4.5916
ans = 4.5912
ans = 4.5908
ans = 4.5904
ans = 4.5900
ans = 4.5896
ans = 4.5892
ans = 4.5888
ans = 4.5884
ans = 4.5880
ans = 4.5876
ans = 4.5872
ans = 4.5868
ans = 4.5864
ans = 4.5860
ans = 4.5856
ans = 4.5852
ans = 4.5848
ans = 4.5844
ans = 4.5840
ans = 4.5837
ans = 4.5833
ans = 4.5829
ans = 4.5825
ans = 4.5821
ans = 4.5818
ans = 4.5814
ans = 4.5810
ans = 4.5806
ans = 4.5803
ans = 4.5799
ans = 4.5795
ans = 4.5791
ans = 4.5788
ans = 4.5784
ans = 4.5780
ans = 4.5777
ans = 4.5773
ans = 4.5770
ans = 4.5766
ans = 4.5762
ans = 4.5759
ans = 4.5755
ans = 4.5752
ans = 4.5748
ans = 4.5745
ans = 4.5741
ans = 4.5738
ans = 4.5734
ans = 4.5731
ans = 4.5727
ans = 4.5724
ans = 4.5720
ans = 4.5717
ans = 4.5713
ans = 4.5710
ans = 4.5707
ans = 4.5703
ans = 4.5700
ans = 4.5697
ans = 4.5693
ans = 4.5690
ans = 4.5687
ans = 4.5683
ans = 4.5680
ans = 4.5677
ans = 4.5673
ans = 4.5670
ans = 4.5667
ans = 4.5664
ans = 4.5660
ans = 4.5657
ans = 4.5654
ans = 4.5651
ans = 4.5648
ans = 4.5645
ans = 4.5641
ans = 4.5638
ans = 4.5635
ans = 4.5632
ans = 4.5629
ans = 4.5626
ans = 4.5623
ans = 4.5620
ans = 4.5617
ans = 4.5614
ans = 4.5611
ans = 4.5607
ans = 4.5604
ans = 4.5601
ans = 4.5598
ans = 4.5595
ans = 4.5592
ans = 4.5590
ans = 4.5587
ans = 4.5584
ans = 4.5581
ans = 4.5578
ans = 4.5575
ans = 4.5572
ans = 4.5569
ans = 4.5566
ans = 4.5563
ans = 4.5560
ans = 4.5558
ans = 4.5555
ans = 4.5552
ans = 4.5549
ans = 4.5546
ans = 4.5544
ans = 4.5541
ans = 4.5538
ans = 4.5535
ans = 4.5532
ans = 4.5530
ans = 4.5527
ans = 4.5524
ans = 4.5522
ans = 4.5519
ans = 4.5516
ans = 4.5513
ans = 4.5511
ans = 4.5508
ans = 4.5505
ans = 4.5503
ans = 4.5500
ans = 4.5497
ans = 4.5495
ans = 4.5492
ans = 4.5490
ans = 4.5487
ans = 4.5484
ans = 4.5482
ans = 4.5479
ans = 4.5477
ans = 4.5474
ans = 4.5472
ans = 4.5469
ans = 4.5467
ans = 4.5464
ans = 4.5462
ans = 4.5459
ans = 4.5457
ans = 4.5454
ans = 4.5452
ans = 4.5449
ans = 4.5447
ans = 4.5444
ans = 4.5442
ans = 4.5440
ans = 4.5437
ans = 4.5435
ans = 4.5432
ans = 4.5430
ans = 4.5428
ans = 4.5425
ans = 4.5423
ans = 4.5420
ans = 4.5418
ans = 4.5416
ans = 4.5413
ans = 4.5411
ans = 4.5409
ans = 4.5407
ans = 4.5404
ans = 4.5402
ans = 4.5400
ans = 4.5397
ans = 4.5395
ans = 4.5393
ans = 4.5391
ans = 4.5388
ans = 4.5386
ans = 4.5384
ans = 4.5382
ans = 4.5380
ans = 4.5377
ans = 4.5375
ans = 4.5373
ans = 4.5371
ans = 4.5369
ans = 4.5366
ans = 4.5364
ans = 4.5362
ans = 4.5360
ans = 4.5358
ans = 4.5356
ans = 4.5354
ans = 4.5352
ans = 4.5350
ans = 4.5347
ans = 4.5345
ans = 4.5343
ans = 4.5341
ans = 4.5339
ans = 4.5337
ans = 4.5335
ans = 4.5333
ans = 4.5331
ans = 4.5329
ans = 4.5327
ans = 4.5325
ans = 4.5323
ans = 4.5321
ans = 4.5319
ans = 4.5317
ans = 4.5315
ans = 4.5313
ans = 4.5311
ans = 4.5309
ans = 4.5307
ans = 4.5305
ans = 4.5303
ans = 4.5301
ans = 4.5299
ans = 4.5298
ans = 4.5296
ans = 4.5294
ans = 4.5292
ans = 4.5290
ans = 4.5288
ans = 4.5286
ans = 4.5284
ans = 4.5283
ans = 4.5281
ans = 4.5279
ans = 4.5277
ans = 4.5275
ans = 4.5273
ans = 4.5272
ans = 4.5270
ans = 4.5268
ans = 4.5266
ans = 4.5264
ans = 4.5263
ans = 4.5261
ans = 4.5259
ans = 4.5257
ans = 4.5256
ans = 4.5254
ans = 4.5252
ans = 4.5250
ans = 4.5249
ans = 4.5247
ans = 4.5245
ans = 4.5243
ans = 4.5242
ans = 4.5240
ans = 4.5238
ans = 4.5237
ans = 4.5235
ans = 4.5233
ans = 4.5232
ans = 4.5230
ans = 4.5228
ans = 4.5227
ans = 4.5225
ans = 4.5223
ans = 4.5222
ans = 4.5220
ans = 4.5218
ans = 4.5217
ans = 4.5215
ans = 4.5214
ans = 4.5212
ans = 4.5210
ans = 4.5209
ans = 4.5207
ans = 4.5206
ans = 4.5204
ans = 4.5203
ans = 4.5201
ans = 4.5199
ans = 4.5198
ans = 4.5196
ans = 4.5195
ans = 4.5193
ans = 4.5192
ans = 4.5190
ans = 4.5189
ans = 4.5187
ans = 4.5186
ans = 4.5184
ans = 4.5183
ans = 4.5181
ans = 4.5180
ans = 4.5178
ans = 4.5177
ans = 4.5175
ans = 4.5174
ans = 4.5172
ans = 4.5171
ans = 4.5170
ans = 4.5168
ans = 4.5167
ans = 4.5165
ans = 4.5164
ans = 4.5162
ans = 4.5161
ans = 4.5160
ans = 4.5158
ans = 4.5157
ans = 4.5155
ans = 4.5154
ans = 4.5153
ans = 4.5151
ans = 4.5150
ans = 4.5148
ans = 4.5147
ans = 4.5146
ans = 4.5144
ans = 4.5143
ans = 4.5142
ans = 4.5140
ans = 4.5139
ans = 4.5138
ans = 4.5136
ans = 4.5135
ans = 4.5134
ans = 4.5132
ans = 4.5131
ans = 4.5130
ans = 4.5129
ans = 4.5127
ans = 4.5126
ans = 4.5125
ans = 4.5123
ans = 4.5122
ans = 4.5121
ans = 4.5120
ans = 4.5118
ans = 4.5117
ans = 4.5116
ans = 4.5115
ans = 4.5113
ans = 4.5112
ans = 4.5111
ans = 4.5110
ans = 4.5108
ans = 4.5107
ans = 4.5106
ans = 4.5105
ans = 4.5104
ans = 4.5102
ans = 4.5101
ans = 4.5100
ans = 4.5099
ans = 4.5098
ans = 4.5096
ans = 4.5095
ans = 4.5094
ans = 4.5093
ans = 4.5092
ans = 4.5091
ans = 4.5089
ans = 4.5088
ans = 4.5087
ans = 4.5086
ans = 4.5085
ans = 4.5084
ans = 4.5083
ans = 4.5081
ans = 4.5080
ans = 4.5079
ans = 4.5078
ans = 4.5077
ans = 4.5076
ans = 4.5075
ans = 4.5074
ans = 4.5073
ans = 4.5071
ans = 4.5070
ans = 4.5069
ans = 4.5068
ans = 4.5067
ans = 4.5066
ans = 4.5065
ans = 4.5064
ans = 4.5063
ans = 4.5062
ans = 4.5061
ans = 4.5060
ans = 4.5059
ans = 4.5058
ans = 4.5057
ans = 4.5056
ans = 4.5055
ans = 4.5053
ans = 4.5052
ans = 4.5051
ans = 4.5050
ans = 4.5049
ans = 4.5048
ans = 4.5047
ans = 4.5046
ans = 4.5045
ans = 4.5044
ans = 4.5043
ans = 4.5042
ans = 4.5041
ans = 4.5040
ans = 4.5040
ans = 4.5039
ans = 4.5038
ans = 4.5037
ans = 4.5036
ans = 4.5035
ans = 4.5034
ans = 4.5033
ans = 4.5032
ans = 4.5031
ans = 4.5030
ans = 4.5029
ans = 4.5028
ans = 4.5027
ans = 4.5026
ans = 4.5025
ans = 4.5024
ans = 4.5023
ans = 4.5023
ans = 4.5022
ans = 4.5021
ans = 4.5020
ans = 4.5019
ans = 4.5018
ans = 4.5017
ans = 4.5016
ans = 4.5015
ans = 4.5014
ans = 4.5014
ans = 4.5013
ans = 4.5012
ans = 4.5011
ans = 4.5010
ans = 4.5009
ans = 4.5008
ans = 4.5008
ans = 4.5007
ans = 4.5006
ans = 4.5005
ans = 4.5004
ans = 4.5003
ans = 4.5002
ans = 4.5002
ans = 4.5001
ans = 4.5000
ans = 4.4999
ans = 4.4998
ans = 4.4997
ans = 4.4997
ans = 4.4996
ans = 4.4995
ans = 4.4994
ans = 4.4993
ans = 4.4993
ans = 4.4992
ans = 4.4991
ans = 4.4990
ans = 4.4989
ans = 4.4989
ans = 4.4988
ans = 4.4987
ans = 4.4986
ans = 4.4985
ans = 4.4985
ans = 4.4984
ans = 4.4983
ans = 4.4982
ans = 4.4982
ans = 4.4981
ans = 4.4980
ans = 4.4979
ans = 4.4979
ans = 4.4978
ans = 4.4977
ans = 4.4976
ans = 4.4976
ans = 4.4975
ans = 4.4974
ans = 4.4973
ans = 4.4973
ans = 4.4972
ans = 4.4971
ans = 4.4970
ans = 4.4970
ans = 4.4969
ans = 4.4968
ans = 4.4968
ans = 4.4967
ans = 4.4966
ans = 4.4965
ans = 4.4965
ans = 4.4964
ans = 4.4963
ans = 4.4963
ans = 4.4962
ans = 4.4961
ans = 4.4961
ans = 4.4960
ans = 4.4959
ans = 4.4958
ans = 4.4958
ans = 4.4957
ans = 4.4956
ans = 4.4956
ans = 4.4955
ans = 4.4954
ans = 4.4954
ans = 4.4953
ans = 4.4952
ans = 4.4952
ans = 4.4951
ans = 4.4950
ans = 4.4950
ans = 4.4949
ans = 4.4949
ans = 4.4948
ans = 4.4947
ans = 4.4947
ans = 4.4946
ans = 4.4945
ans = 4.4945
ans = 4.4944
ans = 4.4943
ans = 4.4943
ans = 4.4942
ans = 4.4942
ans = 4.4941
ans = 4.4940
ans = 4.4940
ans = 4.4939
ans = 4.4938
ans = 4.4938
ans = 4.4937
ans = 4.4937
ans = 4.4936
ans = 4.4935
ans = 4.4935
ans = 4.4934
ans = 4.4934
ans = 4.4933
ans = 4.4933
ans = 4.4932
ans = 4.4931
ans = 4.4931
ans = 4.4930
ans = 4.4930
ans = 4.4929
ans = 4.4928
ans = 4.4928
ans = 4.4927
ans = 4.4927
ans = 4.4926
ans = 4.4926
ans = 4.4925
ans = 4.4924
ans = 4.4924
ans = 4.4923
ans = 4.4923
ans = 4.4922
ans = 4.4922
ans = 4.4921
ans = 4.4921
ans = 4.4920
ans = 4.4920
ans = 4.4919
ans = 4.4918
ans = 4.4918
ans = 4.4917
ans = 4.4917
ans = 4.4916
ans = 4.4916
ans = 4.4915
ans = 4.4915
ans = 4.4914
ans = 4.4914
ans = 4.4913
ans = 4.4913
ans = 4.4912
ans = 4.4912
ans = 4.4911
ans = 4.4911
ans = 4.4910
ans = 4.4910
ans = 4.4909
ans = 4.4909
ans = 4.4908
ans = 4.4908
ans = 4.4907
ans = 4.4907
ans = 4.4906
ans = 4.4906
ans = 4.4905
ans = 4.4905
ans = 4.4904
ans = 4.4904
ans = 4.4903
ans = 4.4903
ans = 4.4902
ans = 4.4902
ans = 4.4901
ans = 4.4901
ans = 4.4900
ans = 4.4900
ans = 4.4899
ans = 4.4899
ans = 4.4898
ans = 4.4898
ans = 4.4898
ans = 4.4897
ans = 4.4897
ans = 4.4896
ans = 4.4896
ans = 4.4895
ans = 4.4895
ans = 4.4894
ans = 4.4894
ans = 4.4893
ans = 4.4893
ans = 4.4893
ans = 4.4892
ans = 4.4892
ans = 4.4891
ans = 4.4891
ans = 4.4890
ans = 4.4890
ans = 4.4890
ans = 4.4889
ans = 4.4889
ans = 4.4888
ans = 4.4888
ans = 4.4887
ans = 4.4887
ans = 4.4887
ans = 4.4886
ans = 4.4886
ans = 4.4885
ans = 4.4885
ans = 4.4884
ans = 4.4884
ans = 4.4884
ans = 4.4883
ans = 4.4883
ans = 4.4882
ans = 4.4882
ans = 4.4882
ans = 4.4881
ans = 4.4881
ans = 4.4880
ans = 4.4880
ans = 4.4880
ans = 4.4879
ans = 4.4879
ans = 4.4878
ans = 4.4878
ans = 4.4878
ans = 4.4877
ans = 4.4877
ans = 4.4876
ans = 4.4876
ans = 4.4876
ans = 4.4875
ans = 4.4875
ans = 4.4875
ans = 4.4874
ans = 4.4874
ans = 4.4873
ans = 4.4873
ans = 4.4873
ans = 4.4872
ans = 4.4872
ans = 4.4872
ans = 4.4871
ans = 4.4871
ans = 4.4870
ans = 4.4870
ans = 4.4870
ans = 4.4869
ans = 4.4869
ans = 4.4869
ans = 4.4868
ans = 4.4868
ans = 4.4868
ans = 4.4867
ans = 4.4867
ans = 4.4867
ans = 4.4866
ans = 4.4866
ans = 4.4865
ans = 4.4865
ans = 4.4865
ans = 4.4864
ans = 4.4864
ans = 4.4864
ans = 4.4863
ans = 4.4863
ans = 4.4863
ans = 4.4862
ans = 4.4862
ans = 4.4862
ans = 4.4861
ans = 4.4861
ans = 4.4861
ans = 4.4860
ans = 4.4860
ans = 4.4860
ans = 4.4859
ans = 4.4859
ans = 4.4859
ans = 4.4859
ans = 4.4858
ans = 4.4858
ans = 4.4858
ans = 4.4857
ans = 4.4857
ans = 4.4857
ans = 4.4856
ans = 4.4856
ans = 4.4856
ans = 4.4855
ans = 4.4855
ans = 4.4855
ans = 4.4854
ans = 4.4854
ans = 4.4854
ans = 4.4854
ans = 4.4853
ans = 4.4853
ans = 4.4853
ans = 4.4852
ans = 4.4852
ans = 4.4852
ans = 4.4851
ans = 4.4851
ans = 4.4851
ans = 4.4851
ans = 4.4850
ans = 4.4850
ans = 4.4850
ans = 4.4849
ans = 4.4849
ans = 4.4849
ans = 4.4849
ans = 4.4848
ans = 4.4848
ans = 4.4848
ans = 4.4847
ans = 4.4847
ans = 4.4847
ans = 4.4847
ans = 4.4846
ans = 4.4846
ans = 4.4846
ans = 4.4845
ans = 4.4845
ans = 4.4845
ans = 4.4845
ans = 4.4844
ans = 4.4844
ans = 4.4844
ans = 4.4844
ans = 4.4843
ans = 4.4843
ans = 4.4843
ans = 4.4843
ans = 4.4842
ans = 4.4842
ans = 4.4842
ans = 4.4841
ans = 4.4841
ans = 4.4841
ans = 4.4841
ans = 4.4840
ans = 4.4840
ans = 4.4840
ans = 4.4840
ans = 4.4839
ans = 4.4839
ans = 4.4839
ans = 4.4839
ans = 4.4838
ans = 4.4838
ans = 4.4838
ans = 4.4838
ans = 4.4837
ans = 4.4837
ans = 4.4837
ans = 4.4837
ans = 4.4836
ans = 4.4836
ans = 4.4836
ans = 4.4836
ans = 4.4836
ans = 4.4835
ans = 4.4835
ans = 4.4835
ans = 4.4835
ans = 4.4834
ans = 4.4834
ans = 4.4834
```

```
% Print theta to screen
% Display gradient descent's result
fprintf('Theta computed from gradient descent:\n%f,\n%f',theta(1),theta(2))
```

```
Theta computed from gradient descent:
-3.630291,
1.166362
```

```
% Plot the linear fit
hold on; % keep previous plot visible
plot(X(:,2), X*theta, '-')
legend('Training data', 'Linear regression')
hold off % don't overlay any more plots on this figure
```



```
% Predict values for population sizes of 35,000 and 70,000
predict1 = [1, 3.5] *theta;
fprintf('For population = 35,000, we predict a profit of %f\n', predict1*10000);
```

```
For population = 35,000, we predict a profit of 4519.767868
```

```
predict2 = [1, 7] * theta;
fprintf('For population = 70,000, we predict a profit of %f\n', predict2*10000);
```

```
For population = 70,000, we predict a profit of 45342.450129
```

Note the way that the lines above use matrix multiplication, rather than explicit summation or looping, to calculate the predictions. This is an example of *code vectorization* in MATLAB.

You should now submit your solutions. Enter **`submit`** *at the command prompt, then enter or confirm your login and token when prompted.*

2.3 Debugging

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

- 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.

2.4 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, there is code set up to calculate $J(\theta)$ over a grid of values using the `computeCost` function that you wrote.

```
% Visualizing J(theta_0, theta_1):
% Grid over which we will calculate J
theta0_vals = linspace(-10, 10, 100);
theta1_vals = linspace(-1, 4, 100);

% 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 the code above is executed, you will have a 2-D array of $J(\theta)$ values. The code below will then use these values to produce surface and contour plots of $J(\theta)$ using the **`surf`** and **`contour`** commands. Run the code in this section now. The resulting plots should look something like the figure below.

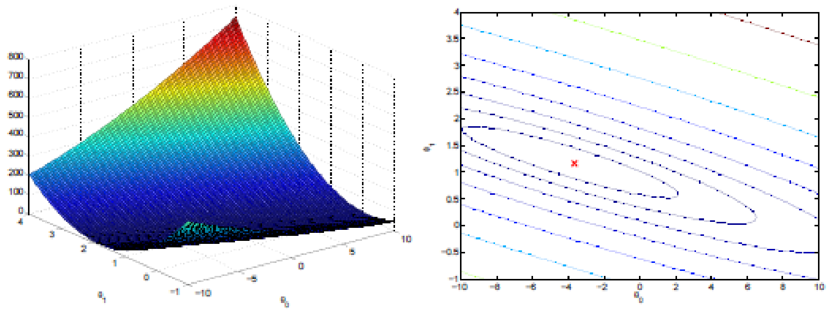
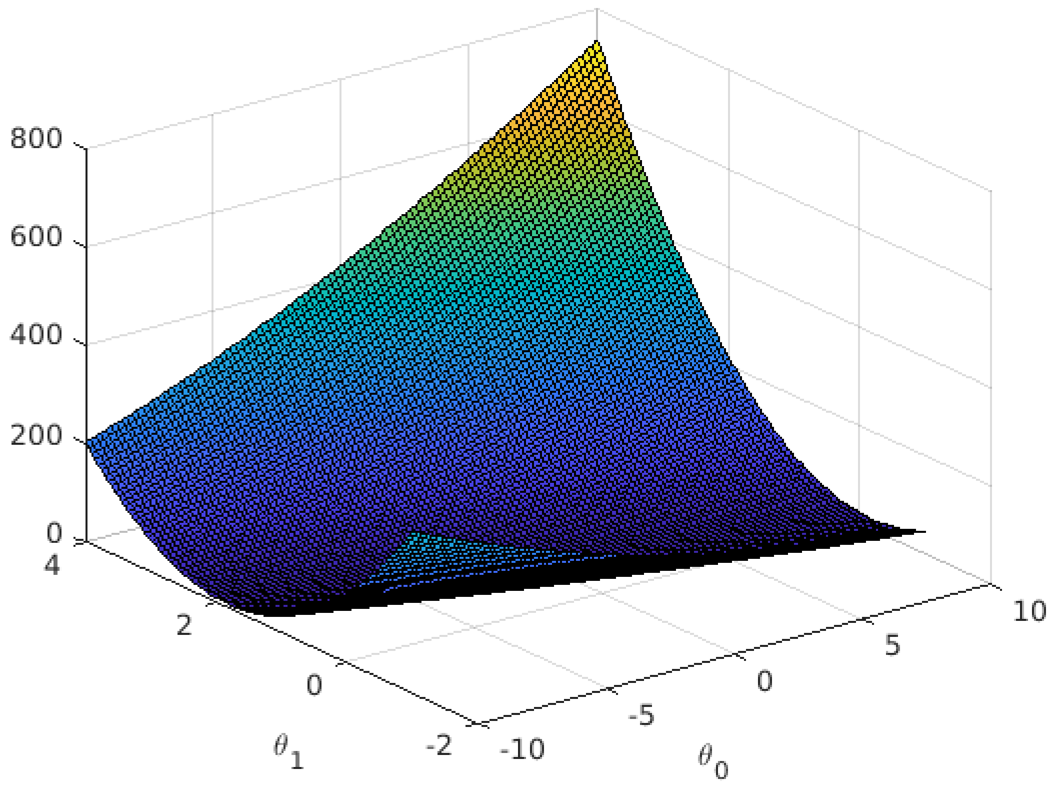


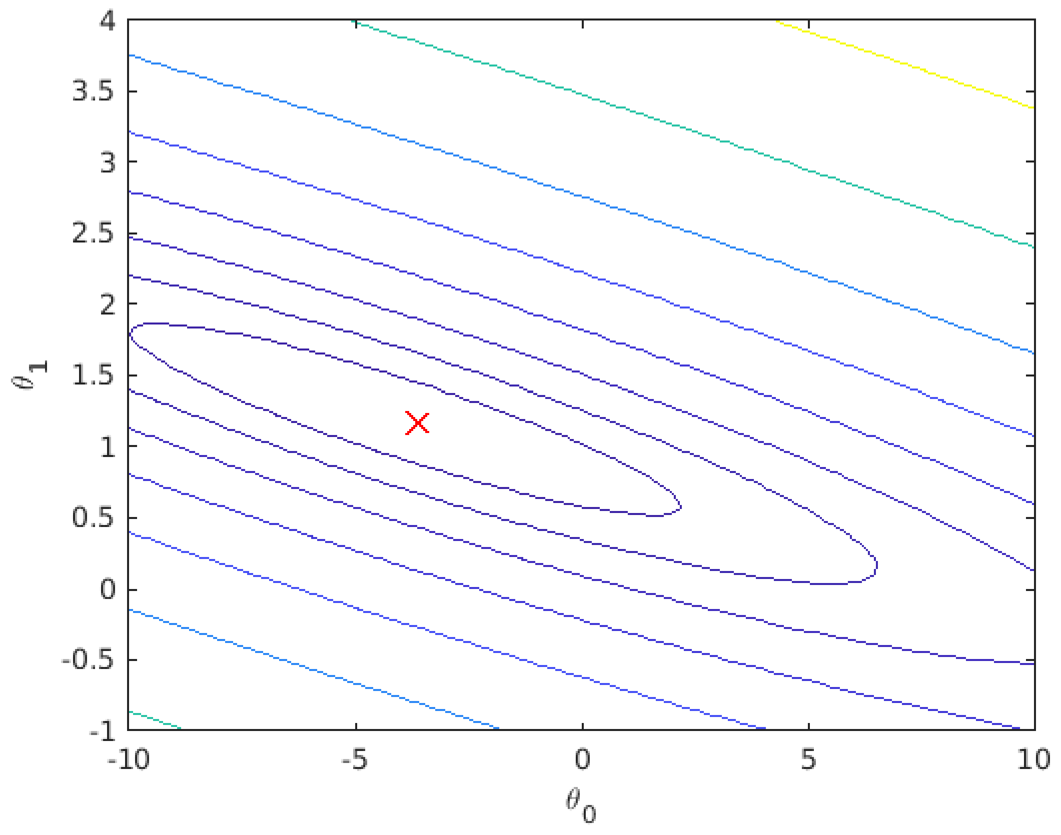
Figure 3: Cost function $J(\theta)$

```
% Because of the way meshgrids work in the surf command, we need to
% transpose J_vals before calling surf, or else the axes will be flipped
J_vals = J_vals';

% Surface plot
figure;
surf(theta0_vals, theta1_vals, J_vals)
xlabel('\theta_0'); ylabel('\theta_1');
```



```
% Contour plot
figure;
% Plot J_vals as 15 contours spaced logarithmically between 0.01 and 100
```

```
contour(theta0_vals, theta1_vals, J_vals, logspace(-2, 3, 20))
xlabel('\theta_0'); ylabel('\theta_1');
hold on;
plot(theta(1), theta(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);
hold off;
```



The purpose of these graphs 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 mininum. (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.

Optional Exercises:

If you have successfully completed the material above, congratulations! You now understand linear regression and should able to start using it on your own datasets. For the rest of this programming exercise, we have included the following optional exercises. These exercises will help you gain a deeper understanding of the material, and if you are able to do so, we encourage you to complete them as well.

3. Linear regression with multiple variables

In this part, you will implement linear regression with multiple variables to predict the prices of houses. Suppose you are selling your house and you want to know what a good market price would be. One way to do this is to first collect information on recent houses sold and make a model of housing prices.

The file `ex1data2.txt` contains a training set of housing prices in Portland, Oregon. The first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house. Run this section now to preview the data.

```
% Load Data
data = load('ex1data2.txt');
X = data(:, 1:2);
y = data(:, 3);
m = length(y);

% Print out some data points
% First 10 examples from the dataset
fprintf(' x = [%.0f %.0f], y = %.0f \n', [X(1:10,:) y(1:10,:)]');
```

```
 x = [2104 3], y = 399900
 x = [1600 3], y = 329900
 x = [2400 3], y = 369000
 x = [1416 2], y = 232000
 x = [3000 4], y = 539900
 x = [1985 4], y = 299900
 x = [1534 3], y = 314900
 x = [1427 3], y = 198999
 x = [1380 3], y = 212000
 x = [1494 3], y = 242500
```

The remainder of this script has been set up to help you step through this exercise.

3.1 Feature Normalization

This section of the script will start by loading and displaying some values from this dataset. By looking at the values, note that house sizes are about 1000 times the number of bedrooms. When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly.

Your task here is to complete the code in `featureNormalize.m` to:

- Subtract the mean value of each feature from the dataset.
- After subtracting the mean, additionally scale (divide) the feature values by their respective "standard deviations".

The standard deviation is a way of measuring how much variation there is in the range of values of a particular feature (most data points will lie within ± 2 standard deviations of the mean); this is an alternative to taking the range of values (*max* - *min*). In MATLAB, you can use the `std` function to compute the standard deviation. For example, inside `featureNormalize.m`, the quantity `X(:,1)` contains all the values of x_1 (house sizes) in the training set, so `std(X(:,1))` computes the standard deviation of the house sizes. At the time that `featureNormalize.m` is called, the extra column of 1's corresponding to $x_0 = 1$ has not yet been added to X (see the code below for details).

You will do this for all the features and your code should work with datasets of all sizes (any number of features / examples). Note that each column of the matrix **X** corresponds to one feature. When you are finished with `featureNormailize.m`, run this section to normailze the features of the housing dataset.

```
% Scale features and set them to zero mean
[X, mu, sigma] = featureNormalize(X);
```

Implementation Note: When normalizing the features, it is important to store the values used for normalization - the mean value and the standard deviation used for the computations. After learning the parameters from the model, we often want to predict the prices of houses we have not seen before. Given a new `x` value (living room area and number of bedrooms), we must first normalize `x` using the mean and standard deviation that we had previously computed from the training set.

Now that we have normailzed the features, we again add a column of ones corresponding to θ_0 to the data matrix `X`.

```
% Add intercept term to X
X = [ones(m, 1) X];
```

3.2 Gradient Descent

Previously, you implemented gradient descent on a univariate regression problem. The only difference now is that there is one more feature in the matrix `X`. The hypothesis function and the batch gradient descent update rule remain unchanged.

You should complete the code in `computeCostMulti.m` and `gradientDescentMulti.m` to implement the cost function and gradient descent for linear regression *with multiple variables*. If your code in the previous part (single variable) already supports multiple variables, you can use it here too.

Make sure your code supports any number of features and is well-vectorized. You can use the command `size(X,2)` to find out how many features are present in the dataset.

We have provided you with the following starter code below that runs gradient descent with a particular learning rate (`alpha`). Your task is to first make sure that your functions `computeCost` and `gradientDescent` already work with this starter code and support multiple variables.

Implementation Note: In the multivariate case, the cost function can also be written in the following vectorized form:

$$J(\theta) = \frac{1}{2m}\left(X\theta - \vec{y}\right)^T\left(X\theta - \vec{y}\right)$$

where

$$X = \begin{bmatrix} -\ (x^{(1)})^T\ - \\ -\ (x^{(2)})^T\ - \\ \vdots \\ -\ (x^{(m)})^T\ - \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

The vectorized version is efficient when you're working with numerical computing tools like MATLAB. If you are an expert with matrix operations, you can prove to yourself that the two forms are equivalent.

```
% Run gradient descent
% Choose some alpha value
alpha = 0.1;
num_iters = 400;

% Init Theta and Run Gradient Descent
theta = zeros(3, 1);
[theta, ~] = gradientDescentMulti(X, y, theta, alpha, num_iters);

% Display gradient descent's result
fprintf('Theta computed from gradient descent:\n%f,\n%f',theta(1),theta(2))
```

```
Theta computed from gradient descent:
340412.659574,
110631.048958
```

Finally, you should complete and run the code below to predict the price of a 1650 sq-ft, 3 br house using the value of `theta` obtained above.

Hint: At prediction, make sure you do the same feature normalization. Recall that the first column of X is all ones. Thus, it does not need to be normalized.

```
% Estimate the price of a 1650 sq-ft, 3 br house
% ====================== YOUR CODE HERE ======================

price = [] ; % Enter your price formula here

% ============================================================

fprintf('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):\n $%f',
```

```
Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):
 $
```

3.2.1 Optional (ungraded) exercise: Selecting learning rates

In this part of the exercise, you will get to try out dierent learning rates for the dataset and find a learning rate that converges quickly. You can change the learning rate by modifying the code below and changing the part of the code that sets the learning rate.

The code below will call your `gradientDescent` function and run gradient descent for about 50 iterations at the chosen learning rate. The function should also return the history of $J(\theta)$ values in a vector J. After the last iteration, the code plots the **J** values against the number of the iterations. If you picked a learning rate within a good range, your plot should look similar Figure 4 below.

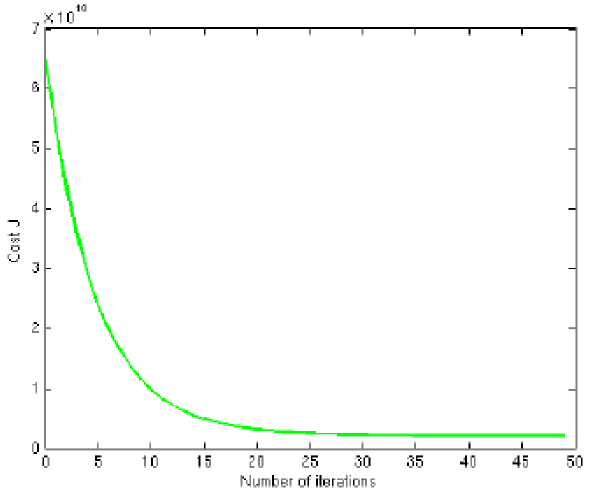


Figure 4: Convergence of gradient descent with an appropriate learning rate

If your graph looks very different, especially if your value of $J(\theta)$ increases or even blows up, use the control to adjust your learning rate and try again. We recommend trying values of the learning rate on a log-scale, at multiplicative steps of about 3 times the previous value (i.e., 0.3, 0.1, 0.03, 0.01 and so on). You may also want to adjust the number of iterations you are running if that will help you see the overall trend in the curve.

Implementation Note: If your learning rate is too large, $J(\theta)$ can diverge and 'blow up', resulting in values which are too large for computer calculations. In these situations, MATLAB will tend to return `NaN`s. `NaN` stands for 'not a number' and is often caused by undefined operations that involve $\pm\infty$.

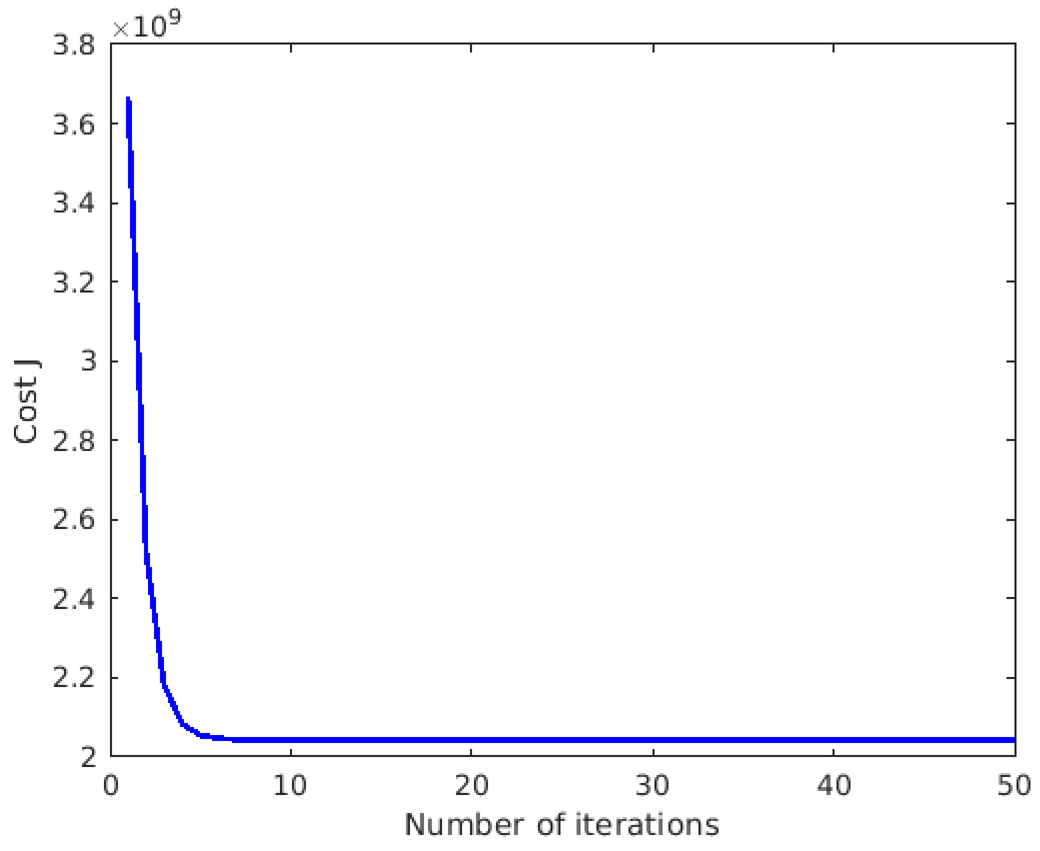
MATLAB Tip: To compare how different learning learning rates affect convergence, it's helpful to plot `J` for several learning rates on the same figure. In MATLAB, this can be done by performing gradient descent multiple times with a `hold on` command between plots. Make sure to use the `hold off` command when you are done plotting in that figure. Concretely, if you've tried three different values of `alpha` (you should probably try more values than this) and stored the costs in `J1`, `J2` and `J3`, you can use the following commands to plot them on the same figure:

```
plot(1:50, J1(1:50), 'b');
hold on
plot(1:50, J2(1:50), 'r');
plot(1:50, J3(1:50), 'k');
hold off
```

The final arguments `'b'`,`'r'`, and `'k'` specify different colors for the plots. If desired, you can use this technique and adapt the code below to plot multiple convergence histories in the same plot.

```
% Run gradient descent:
% Choose some alpha value
alpha = 1;
num_iters = 50;

% Init Theta and Run Gradient Descent
theta = zeros(3, 1);
[~, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters);
```

```
% Plot the convergence graph
plot(1:num_iters, J_history, '-b', 'LineWidth', 2);
xlabel('Number of iterations');
ylabel('Cost J');
```



Notice the changes in the convergence curves as the learning rate changes. With a small learning rate, you should find that gradient descent takes a very long time to converge to the optimal value. Conversely, with a large learning rate, gradient descent might not converge or might even diverge!

Using the best learning rate that you found, run the section of code below, which will run gradient descent until convergence to find the final values of θ. Next, use this value of θ to predict the price of a house with 1650 square feet and 3 bedrooms. You will use value later to check your implementation of the normal equations. Don't forget to normalize your features when you make this prediction!

```
% Run gradient descent
% Replace the value of alpha below best alpha value you found above
alpha = 0.1;
num_iters = 400;

% Init Theta and Run Gradient Descent
theta = zeros(3, 1);
[theta, ~] = gradientDescentMulti(X, y, theta, alpha, num_iters);

% Display gradient descent's result
```

```
fprintf('Theta computed from gradient descent:\n%f\n%f',theta(1),theta(2))
```

```
Theta computed from gradient descent:
340412.659574
110631.048958
```

```
% Estimate the price of a 1650 sq-ft, 3 br house. You can use the same
% code you entered ealier to predict the price
% ====================== YOUR CODE HERE ======================

price = []; % Enter your price formula here

% ============================================================

fprintf('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):\n $%f',
```

```
Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):
 $
```

3.3 Normal Equations

In the lecture videos, you learned that the closed-form solution to linear regression is

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

Using this formula does not require any feature scaling, and you will get an exact solution in one calculation: there is no "loop until convergence" like in gradient descent.

Complete the code in `normalEqn.m` to use the formula above to calculate θ, then run the code in this section. Remember that while you don't need to scale your features, we still need to add a column of 1's to the X matrix to have an intercept term (θ_0) . Note that the code below will add the column of 1's to X for you.

```
% Solve with normal equations:
% Load Data
data = csvread('ex1data2.txt');
X = data(:, 1:2);
y = data(:, 3);
m = length(y);

% Add intercept term to X
X = [ones(m, 1) X];

% Calculate the parameters from the normal equation
theta = normalEqn(X, y);

% Display normal equation's result
fprintf('Theta computed from the normal equations:\n%f\n%f', theta(1),theta(2));
```

```
Theta computed from the normal equations:
0.000000
0.000000
```

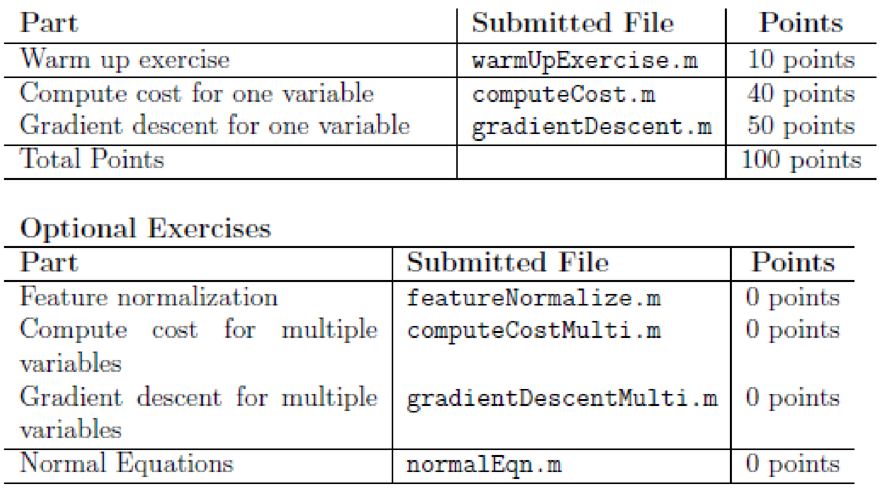
Optional (ungraded) exercise: Now, once you have found θ using this method, use it to make a price prediction for a 1650-square-foot house with 3 bedrooms. You should find that gives the same predicted price as the value you obtained using the model fit with gradient descent (in Section 3.2.1).

```
% Estimate the price of a 1650 sq-ft, 3 br house.
% ====================== YOUR CODE HERE ======================

price = []; % Enter your price forumla here

% ============================================================

fprintf('Predicted price of a 1650 sq-ft, 3 br house (using normal equations):\n $%f',
```

```
Predicted price of a 1650 sq-ft, 3 br house (using normal equations):
 $
```

Submission and Grading

After completing various parts of the assignment, be sure to use the submit function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.



Part	Submitted File	Points
Warm up exercise	`warmUpExercise.m`	10 points
Compute cost for one variable	`computeCost.m`	40 points
Gradient descent for one variable	`gradientDescent.m`	50 points
Total Points		100 points

Optional Exercises

Part	Submitted File	Points
Feature normalization	`featureNormalize.m`	0 points
Compute cost for multiple variables	`computeCostMulti.m`	0 points
Gradient descent for multiple variables	`gradientDescentMulti.m`	0 points
Normal Equations	`normalEqn.m`	0 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.