Programming Exercise 1: Linear Regression

Machine Learning

# Introduction

In this exercise, you will implement linear regression and get to see it work on data. Before starting on this programming exercise, we strongly recom- mend watching the video lectures and completing the review questions for the associated topics.

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.

You can also find instructions for installing Octave/MATLAB in the “En- vironment Setup Instructions” of the course website.

## Files included in this exercise

ex1.m - Octave/MATLAB script that steps you through the exercise ex1 multi.m - Octave/MATLAB script for the later parts of the exercise ex1data1.txt - Dataset for linear regression with one variable ex1data2.txt - Dataset for linear regression with multiple variables submit.m - Submission script that sends your solutions to our servers

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

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

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

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

[*†*] computeCostMulti.m - Cost function for multiple variables

[*†*] gradientDescentMulti.m - Gradient descent for multiple variables

[*†*] featureNormalize.m - Function to normalize features

[*†*] normalEqn.m - Function to compute the normal equations

*y* indicates files you will need to complete

*†* indicates optional exercises

Throughout the exercise, you will be using the scripts ex1.m and ex1 multi.m.

These scripts set up the dataset for the problems and make calls to functions that you will write. You do not need to modify either of them. 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 complete the first part of the exercise to implement linear regression with one variable. The second part of the exercise, which is optional, covers linear regression with multiple variables.

## Where to get help

The exercises in this course use Octave[1](#_bookmark0)or MATLAB, a high-level program- ming language well-suited for numerical computations. If you do not have Octave or MATLAB installed, please refer to the installation instructions in the “Environment Setup Instructions” of the course website.

At the Octave/MATLAB command line, typing help followed by a func- tion 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.](http://www.gnu.org/software/octave/doc/interpreter/) MAT- LAB documentation can be found at the[MATLAB documentation pages.](http://www.mathworks.com/help/matlab/?refresh=true)

We also strongly encourage using the online **Discussions** to discuss ex- ercises with other students. However, do not look at any source code written by others or share your source code with others.

# Simple Octave/MATLAB function

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

A = eye(5);

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

When you are finished, run ex1.m (assuming you are in the correct di- rectory, 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.

## Submitting Solutions

After completing a part of the exercise, you can submit your solutions for grading by typing submit at the Octave/MATLAB command line. The sub- mission script will prompt you for your login e-mail and submission token and ask you which files you want to submit. You can obtain a submission token from the web page for the assignment.

*You should now submit your solutions.*

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

# 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 prob- lem. 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.

## 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');%readcommaseparateddata X = data(:, 1); y = data(:, 2);

m = length(y);%numberoftrainingexamples

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);%Plotthedata ylabel('Profitin$10,000s');%Setthey xlabel('PopulationofCityin10,000s');%Setthex

*−*axislabel

*−*axislabel

Now, when you continue to run ex1.m, our end result should look like Figure [1,](#_bookmark1) 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 doc- umentation. (To change the markers to red “x”, we used the option ‘rx’ together with the plot command, i.e., plot(..,[your options here],.., ‘rx’); )

25

20

15

10

Profit in $10,000s

5

0

−5

4 6 8 10 12 14 16 18 20 22 24

Population of City in 10,000s

Figure 1: Scatter plot of training data

## Gradient Descent

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

### Update Equations

The objective of linear regression is to minimize the cost function

*J* (*θ*) = 1 Σ .*h* (*x*(*i*)) *− y*(*i*)Σ2

*i*=1

*m*

2*m*

*θ*

where the hypothesis *hθ*(*x*) is given by the linear model

*hθ*(*x*) = *θT x* = *θ*0 + *θ*1*x*1

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

1 Σ (*i*) (*i*) (*i*)

*m*

*θ* := *θ − α* (*h* (*x* ) *− y* )*x* (simultaneously update *θ* for all *j*)*.*

*i*=1

*j*

*j*

*m*

*θ*

*j*

*j*

With each step of gradient descent, your parameters *θj* 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 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’.

### 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)];%Addacolumnofonestox theta = zeros(2, 1);%initializefittingparameters

iterations = 1500;

alpha = 0.01;

* + 1. **Computing the cost** *J* (*θ*)

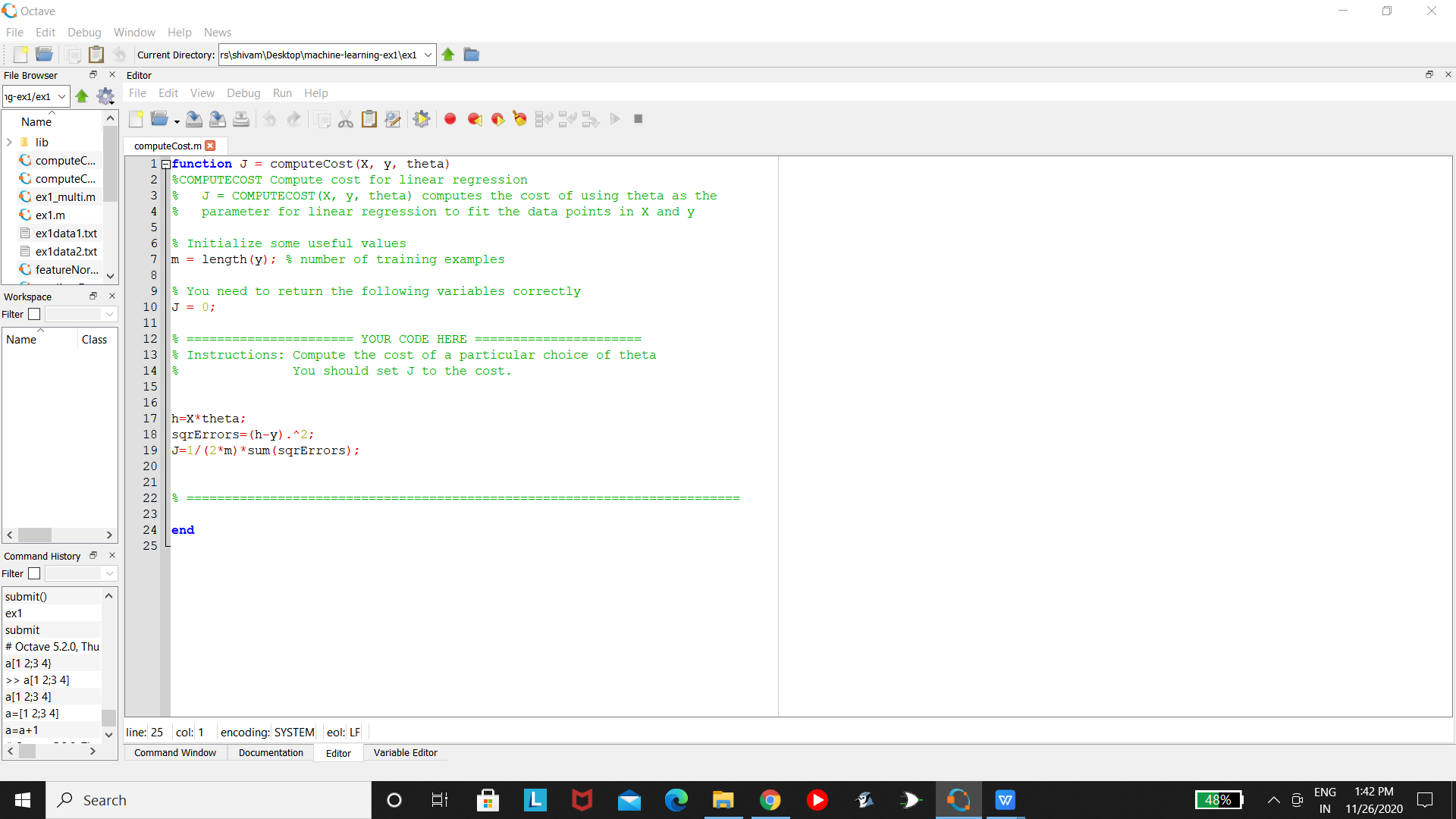
As you perform gradient descent to learn minimize the cost function *J* (*θ*), it is helpful to monitor the convergence by computing the cost. In this section, you will implement a function to calculate *J* (*θ*) so you can check the convergence of your gradient descent implementation.

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 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.

*You should now submit your solutions.*



### 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 opti- mize and what is being updated. Keep in mind that the cost *J* (*θ*) is parame- terized 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*. Refer to the equations in this handout 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* (*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, ex1.m will use your final parameters to plot the linear fit. The result should look something like Figure [2:](#_bookmark2)

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 loop- ing, to calculate the predictions. This is an example of code vectorization in Octave/MATLAB.

*You should now submit your solutions.*

predict1 = [1, 3.5] \* theta;

predict2 = [1, 7] \* theta;

## 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 stor- ing *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 compat- ible dimensions. Printing the dimensions of variables with the size command will help you debug.

25

Training data Linear regression

20

15

10

Profit in $10,000s

5

0

−5

4 6 8 10 12 14 16 18 20 22 24

Population of City in 10,000s

Figure 2: Training data with linear regression fit

* 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.
  1. **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 the next step of ex1.m, there is code set up to calculate *J* (*θ*) over a grid of values using the computeCost function that you wrote.

%initializeJ valstoamatrixof0's

J vals = zeros(length(theta0 vals), length(theta1 vals));

%FilloutJ

vals

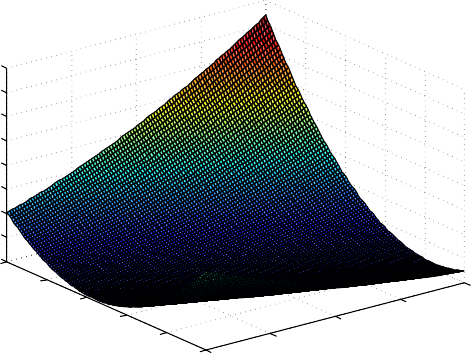
fori = 1:length(theta0 vals) forj = 1:length(theta1 vals)

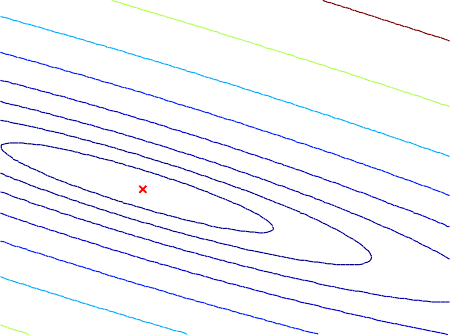
t = [theta0 vals(i); theta1 vals(j)]; J vals(i,j) = computeCost(x, y, t);

end

end

After these lines are executed, you will have a 2-D array of *J* (*θ*) values. The script ex1.m will then use these values to produce surface and contour plots of *J* (*θ*) using the surf and contour commands. The plots should look something like Figure [3:](#_bookmark3)

4



800

700

600

500

400

300

200

100

0

4

3

2 5

1 0

0 −5

−1 −10



1 0

3.5

3

2

1.5

1

1

0.5

0

10

−0.5

−1

−10 −8 −6 −4 −2 0 2 4 6 8 10

0

(a) Surface (b) Contour, showing minimum

Figure 3: Cost function *J* (*θ*)

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 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.

