**LINEAR FUNCTIONS REPORT**



**School or Department:** Wu Han University

**Grade and Specialty:** Excellent engineer 2016

**Name:** Guo Yang, Kong Chuishun, Tang Rui

**Advisor:** Fangling Pu

March 18 2018

【**Abstract**】In this lab, we purpose to review the linear functions and learn how to apply it to a engineering problem with MATLAB. And we will introduce linear and aﬃne functions, and describe some common settings where they arise, including regression models. In the application example, we will implement linear regression with one variable to predict proﬁts for a food truck and select the next city for opening a new outlet. With the data for proﬁts and populations from the cities, we will construct a linear predicted model to help us make the best choice.

**【Key words】** MATLAB; linear functions; predicted model

**Table of Contents**

[Abstract 1](#_Toc324279571)

[Chapter 1 Introduction 1](#_Toc324279574)

[1.1 linear function 1](#_Toc324279574)

[1.2 The application exercise 1](#_Toc324279574)

[Chapter 2 Solutions 2](#_Toc324279575)

[2.1 Plotting the Data 2](#_Toc324279575)

[2.2 Gradient Descent 3](#_Toc324279575)

[2.2.1 Update Equations 3](#_Toc324279575)

[2.2.2 Implementation 3](#_Toc324279575)

[2.2.3 Computing the cost *J(θ)* 3](#_Toc324279575)

[2.2.4 Gradient descent 3](#_Toc324279575)

[2.3 Visualizing *J(θ)* 4](#_Toc324279575)

[Chapter 3 Results and Conclusion 5](#_Toc324279576)

[Bibliography 7](#_Toc324279590)

[Appendix 7](#_Toc324279591)

**Chapter 1 Introduction**

**1.1 Linear functions**

The function f satisﬁes the property：

*f(α****x*** *+ β****y****) =* a*T(*α***x*** *+ β****y****) = aT(α****x****) + aT(β****y****) = α(aT****x****) + β(aT****y****) = αf(****x****) + βf(****y****)*

for all n-vectors x, y, and all scalars α, β. This property is called superposition. A function that satisﬁes the superposition property is called linear

If a function *f* is linear, superposition extends to linear combinations of any number of vectors, and not just linear combinations of two vectors: We have

*f(α1****x1*** *+···+ αk****xk****) = α1f(****x1****) +···+ αkf(****xk****)*

for any n vectors *x1,...,xk,* and any scalars α1,...,αk. (This more general k-term form of superposition reduces to the two-term form given above when k = 2.) To see this, we note that

*f(α1****x1*** *+···+ αk****xk****) = α1f(****x1****) + f(α2****x2*** *+···+ αk****xk****)*

*= α1f(****x1****) + α2f(x2) + f(α3****x3*** *+···+ αk****xk****)*

. . .

*= α1f(****x1****) +···+ αkf(****xk****).*

The superposition equality is sometimes broken down into two properties, one involving the scalar-vector product and one involving vector addition in the argument. A function *f : Rn → R* is linear if it satisﬁes the following two properties.

• Homogeneity. For any n-vector ***x*** and any scalar α, *f(α****x****) = αf(****x****).*

• Additivity. For any n-vectors ***x*** and **y**, *f(****x*** *+* ***y****) = f(****x****) + f(****y****).*

Homogeneity states that scaling the (vector) argument is the same as scaling the function value; additivity says that adding (vector) arguments is the same as adding the function values.

**1.2 The application exercise**

Suppose we 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 I have data for proﬁts and populations from the cities.

We will implement linear regression with one variable to predict proﬁts for a food truck and use this data to help me 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.

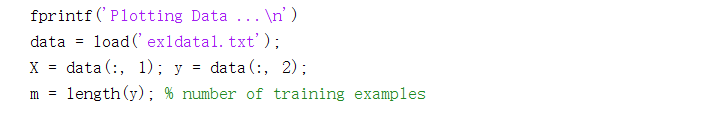
The ex1.m script has already been set up to load this data for us.

**Chapter 2 Solutions**

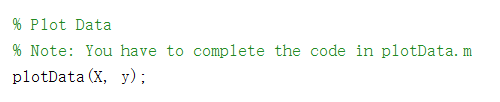
**2.1** **Plotting the Data**

For this dataset, we use a scatter plot to visualize the data, since it has only two properties to plot (proﬁt and population).

In ex1.m, the dataset is loaded from the data ﬁle into the variables ***X*** and ***y***:



Next, the script calls the plotData function to create a scatter plot of the data. Our job is to complete plotData.m to draw the plot; modify the ﬁle and ﬁll in the following code



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

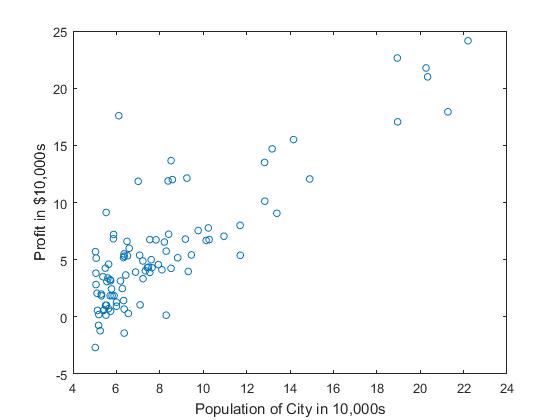


Figure 1: Scatter plot of training data

**2.2 Gradient Descent**

We 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(θ)*=*2*

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

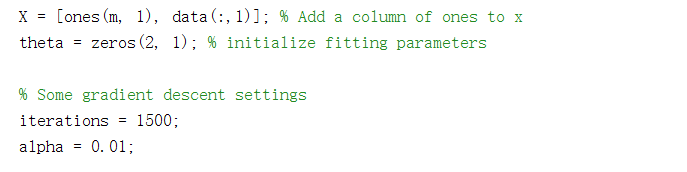
*hθ(x) = θTx = θ0 + θ1****x1***

The parameters of our model are the *θj* values. These are the values we 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

With each step of gradient descent, your parameters *θj*come closer to the optimal values that will achieve the lowest cost *J(θ)*.

**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 θ0 intercept term. We also initialize the initial parameters to 0 and the learning rate alpha to 0.01



**2.2.3 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.

**2.2.4 Gradient descent**

Next, we implement gradient descent in the ﬁle gradientDescent.m and need to supply the updates to *θ* within each iteration. 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. After ﬁnished, ex1.m will use your ﬁnal parameters to plot the linear ﬁt. The result like Figure 2:

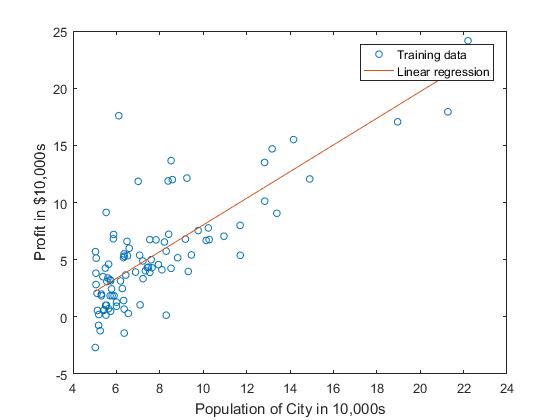
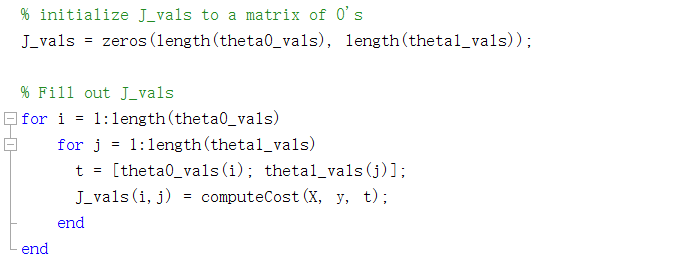


Figure 2: Training data with linear regression ﬁt

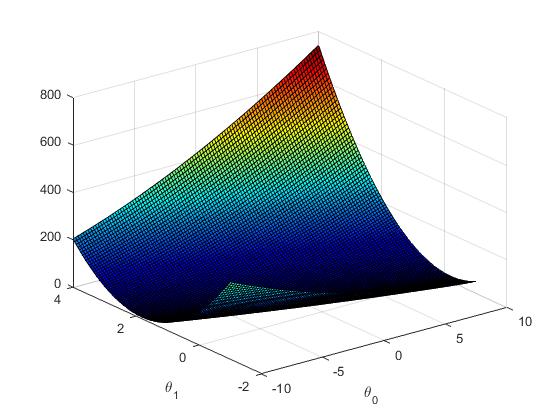
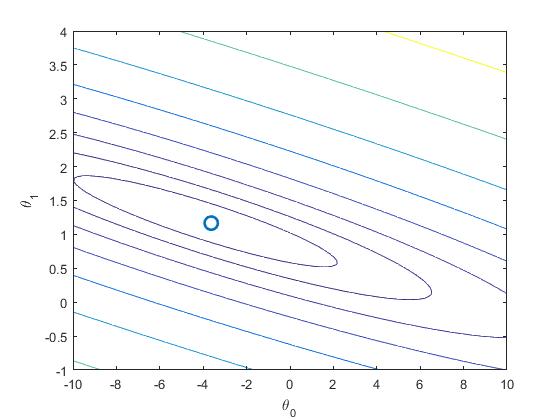
Our ﬁnal values for *θ* will also be used to make predictions on proﬁts in areas of 35,000 and 70,000 people.

**2.3 Visualizing *J(θ)***

To understand the cost function *J(θ)* better, we will now plot the cost over a 2-dimensional grid of *θ0* and *θ1* values. 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.



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:

**** 

(a) Surface (b) Contour, showing minimum

Figure 3: Cost function *J(θ)*

The purpose of these graphs is to show that how *J(θ)* varies with changes in *θ0* and *θ1*. The cost function *J(θ)* 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 *θ0* and *θ1*, and each step of gradient descent moves closer to this point.

**Chapter 3 Results and** **Conclusion**

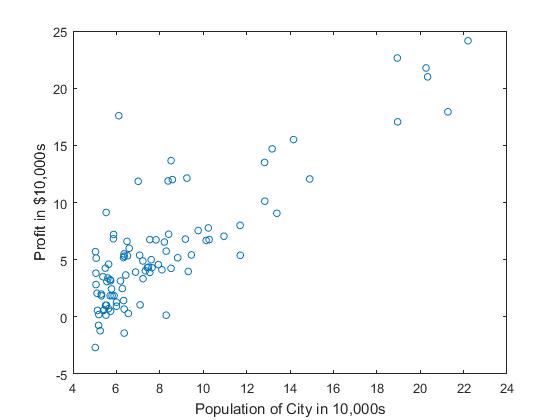


Figure 1: Scatter plot of training data

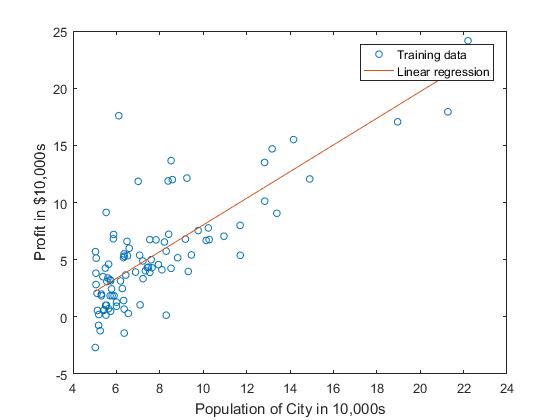


Figure 2: Training data with linear regression ﬁt

The linear ﬁt result just like Figure 2.

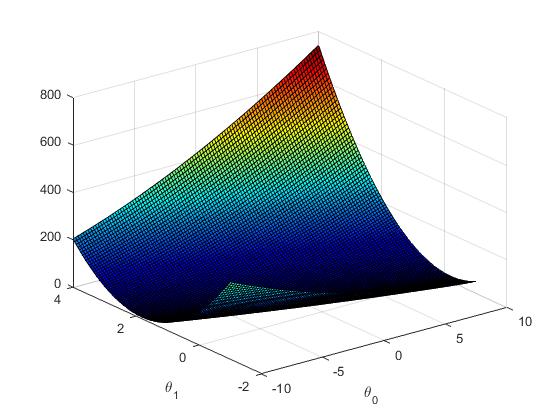
****

Figure3 Surface

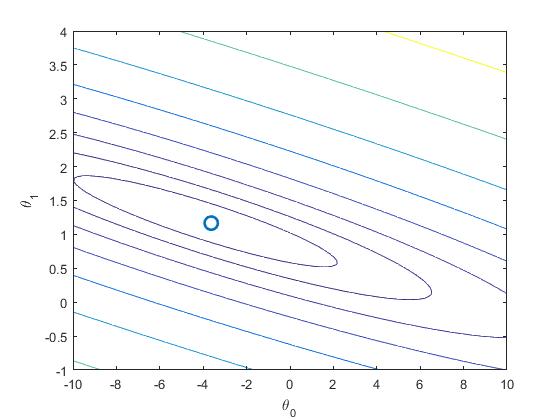


Figure4 Contour, showing minimum

The purpose of these graphs is to show that how *J(θ)* varies with changes in *θ0* and *θ1*. The cost function *J(θ)* is bowl-shaped and has a global minimum. (This is easier to see in the contour plot than in the 3D surface plot).

**Bibliography**

[1] Introduction to Applied Linear Algebra (Vectors, Matrices, and Least Squares), Stephen Boyd, Department of Electrical Engineering Stanford University Lieven Vandenberghe, Department of Electrical and Computer Engineering University of California, Los Angeles

**Appendix**

Ex1.m:

%% Machine Learning Online Class - Exercise 1: Linear Regression

% Instructions

% ------------

%

% This file contains code that helps you get started on the

% linear exercise. You will need to complete the following functions

% in this exericse:

%

% warmUpExercise.m

% plotData.m

% gradientDescent.m

% computeCost.m

% gradientDescentMulti.m

% computeCostMulti.m

% featureNormalize.m

% normalEqn.m

%

% For this exercise, you will not need to change any code in this file,

% or any other files other than those mentioned above.

%

% x refers to the population size in 10,000s

% y refers to the profit in $10,000s

%

%% Initialization

clear ; close all; clc

%% ==================== Part 1: Basic Function ====================

% Complete warmUpExercise.m

fprintf('Running warmUpExercise ... \n');

fprintf('5x5 Identity Matrix: \n');

warmUpExercise()

fprintf('Program paused. Press enter to continue.\n');

pause;

%% ======================= Part 2: Plotting =======================

fprintf('Plotting Data ...\n')

data = load('ex1data1.txt');

X = data(:, 1); y = data(:, 2);

m = length(y); % number of training examples

% Plot Data

% Note: You have to complete the code in plotData.m

plotData(X, y);

fprintf('Program paused. Press enter to continue.\n');

pause;

%% =================== Part 3: Cost and Gradient descent ===================

X = [ones(m, 1), data(:,1)]; % Add a column of ones to x

theta = zeros(2, 1); % initialize fitting parameters

% Some gradient descent settings

iterations = 1500;

alpha = 0.01;

fprintf('\nTesting the cost function ...\n')

% compute and display initial cost

J = computeCost(X, y, theta);

fprintf('With theta = [0 ; 0]\nCost computed = %f\n', J);

fprintf('Expected cost value (approx) 32.07\n');

% further testing of the cost function

J = computeCost(X, y, [-1 ; 2]);

fprintf('\nWith theta = [-1 ; 2]\nCost computed = %f\n', J);

fprintf('Expected cost value (approx) 54.24\n');

fprintf('Program paused. Press enter to continue.\n');

pause;

fprintf('\nRunning Gradient Descent ...\n')

% run gradient descent

theta = gradientDescent(X, y, theta, alpha, iterations);

% print theta to screen

fprintf('Theta found by gradient descent:\n');

fprintf('%f\n', theta);

fprintf('Expected theta values (approx)\n');

fprintf(' -3.6303\n 1.1664\n\n');

% 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);

predict2 = [1, 7] \* theta;

fprintf('For population = 70,000, we predict a profit of %f\n',...

predict2\*10000);

fprintf('Program paused. Press enter to continue.\n');

pause;

%% ============= Part 4: Visualizing J(theta\_0, theta\_1) =============

fprintf('Visualizing J(theta\_0, theta\_1) ...\n')

% 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

% 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);