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**

Chapter 1 Introduction **1**

1.1 linear function **1**

1.2 The application exercise **1**

Chapter 2 Solutions **2**

2.1 Plotting the Data **2**

2.2 Gradient Descent **3**

2.2.1 Update Equations **3**

2.2.2 Implementation **3**

2.2.3 Computing the cost $J_{(\theta)}$ **3**

2.2.4 Gradient descent **3**

2.3 Visualizing $J_{(\theta)}$ **4**

Chapter 3 Results and Conclusion **5**

Bibliography **7**

Appendix **7**

Chapter 1 Introduction

1.1 Linear functions

The function f satisfies the property：

$$f(\alpha \boldsymbol{x} + \beta \boldsymbol{y}) = aT(\alpha \boldsymbol{x} + \beta \boldsymbol{y}) = aT(\alpha \boldsymbol{x}) + aT(\beta \boldsymbol{y}) = \alpha(aT\boldsymbol{x}) + \beta(aT\boldsymbol{y}) = \alpha f(\boldsymbol{x}) + \beta f(\boldsymbol{y})$$

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

$f(\alpha_1 \boldsymbol{x}_1 + \cdots + \alpha_k \boldsymbol{x}_k) = \alpha 1 f(\boldsymbol{x}_1) + \cdots + \alpha_k f(\boldsymbol{x}_k)$

for any n vectors $x_1,\ldots,x_k$, and any scalars $\alpha_1,\ldots,\alpha_k$. (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

$$\begin{aligned} f(\alpha_1 \boldsymbol{x}_1 + \cdots + \alpha_k \boldsymbol{x}_k) &= \alpha_1 f(\boldsymbol{x}_1) + f(\alpha_2 \boldsymbol{x}_2 + \cdots + \alpha_k \boldsymbol{x}_k) \\ &= \alpha_1 f(\boldsymbol{x}_1) + \alpha_2 f(x_2) + f(\alpha_3 \boldsymbol{x}_3 + \cdots + \alpha_k \boldsymbol{x}_k) \\ &\ \ \vdots \\ &= \alpha_1 f(\boldsymbol{x}_1) + \cdots + \alpha_k f(\boldsymbol{x}_k). \end{aligned}$$

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 $f : Rn \rightarrow R$ is linear if it satisfies the following two properties.

- Homogeneity. For any n-vector $\boldsymbol{x}$ and any scalar α, $f(\alpha \boldsymbol{x}) = \alpha f(\boldsymbol{x})$.
- Additivity. For any n-vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, $f(\boldsymbol{x} + \boldsymbol{y}) = f(\boldsymbol{x}) + f(\boldsymbol{y})$.

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:

```
fprintf('Plotting Data ...\n')
data = load('ex1data1.txt');
X = data(:, 1); y = data(:, 2);
m = length(y); % number of training examples
```

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

```
% Plot Data
% Note: You have to complete the code in plotData.m
plotData(X, y);
```

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.

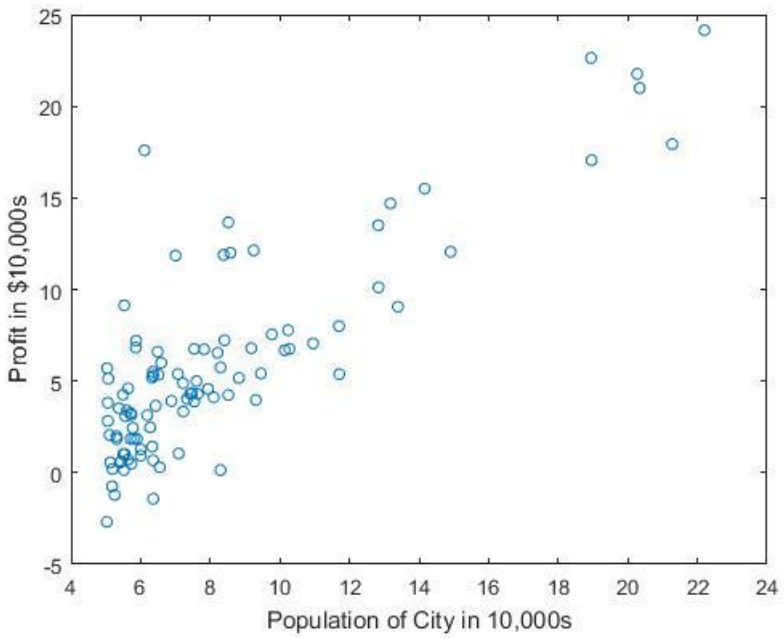


Figure 1: Scatter plot of training data

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

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

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

$$h_{\theta(x)}=\theta_{Tx}=\theta_0+\theta_1 x_1$$

The parameters of our model are the θ_j values. These are the values we 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

With each step of gradient descent, your parameters θ_j come closer to the optimal values that will achieve the lowest cost $J_{(\theta)}$.

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

```
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;
```

2.2.3 Computing the cost $J_{(\theta)}$

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 $J_{(\theta)}$ 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 $J_{(\theta)}$ 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:

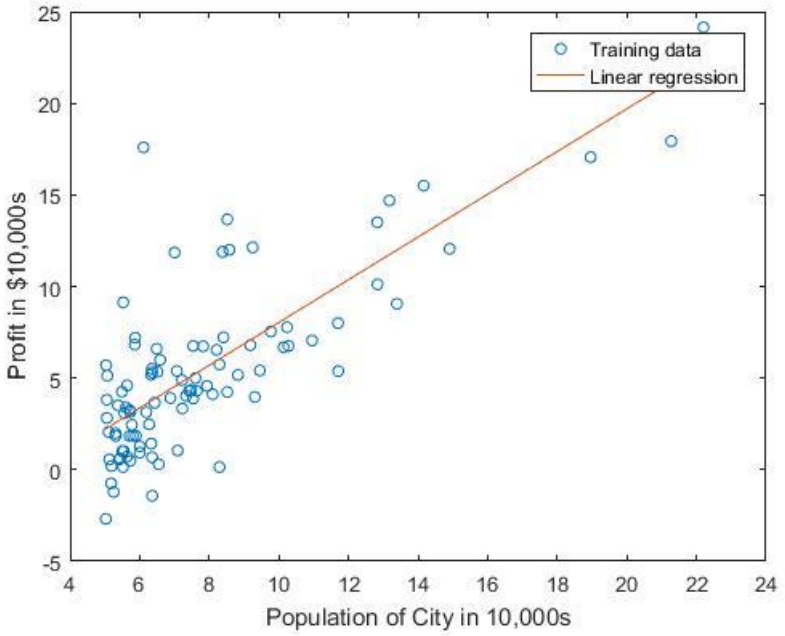


Figure 2: Training data with linear regression fit

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

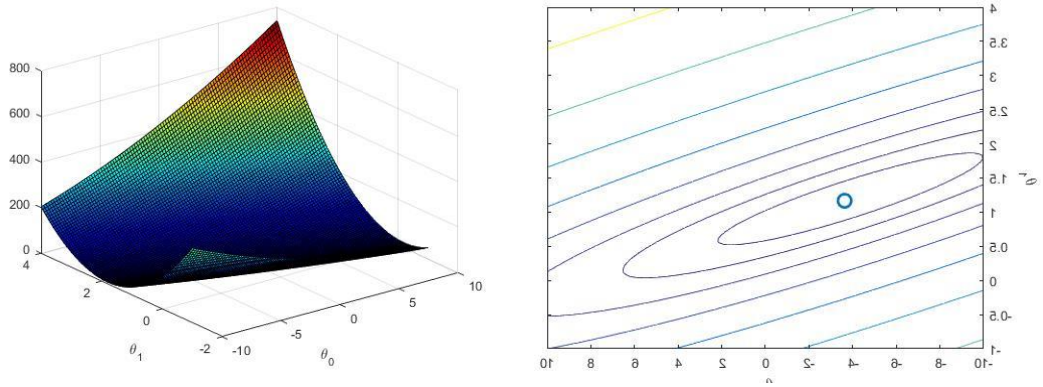
2.3 Visualizing *J(θ)*

To understand the cost function $J_{(\theta)}$ better, we will now plot the cost over a 2-dimensional grid of θ_0 and θ_1 values. 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.

```
% 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(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 $J_{(\theta)}$

The purpose of these graphs is to show 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 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.

Chapter 3 Results and Conclusion

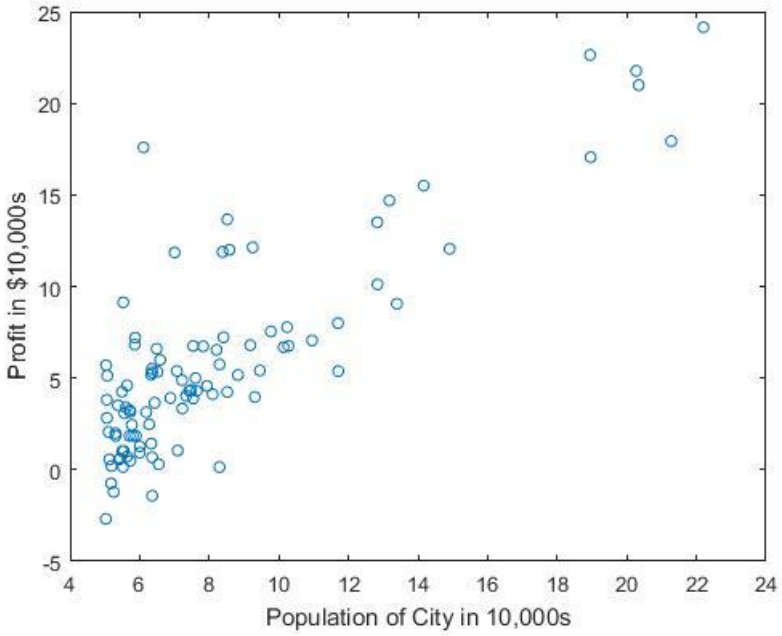


Figure 1: Scatter plot of training data

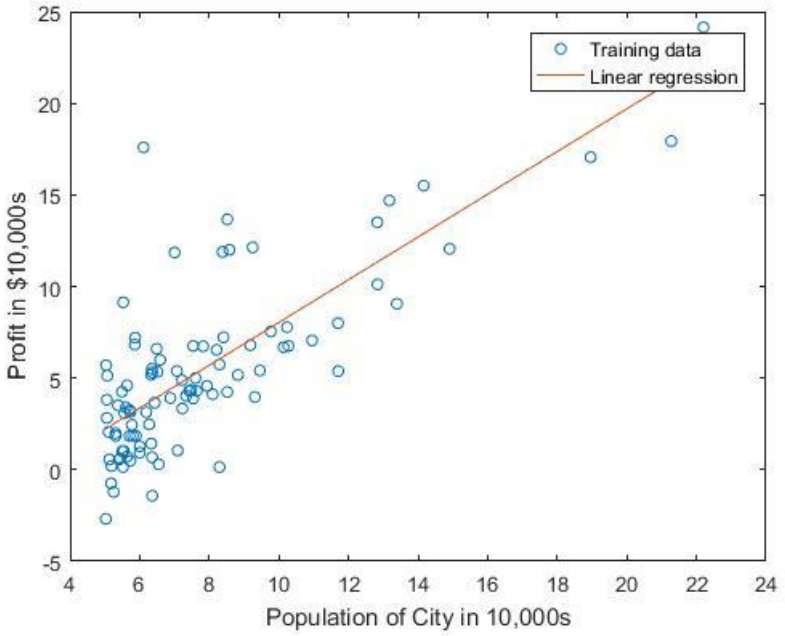


Figure 2: Training data with linear regression fit

The linear fit result just like Figure 2.

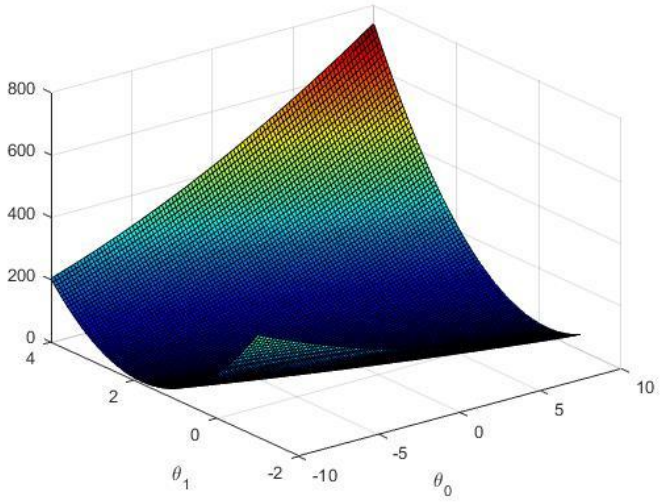


Figure3 Surface

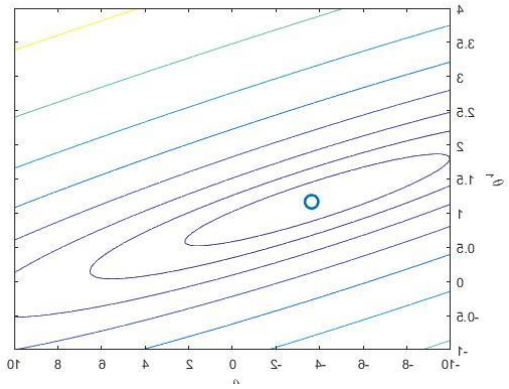


Figure4 Contour, showing minimum

The purpose of these graphs is to show that how $J_{(\theta)}$ 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).

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);
```