3 Simple Linear Regression (Exercise 2)

3.1 Short theory

This first exercise will give you practice with simple linear regression and gradient descent algorithm. At a theoretical level, **Gradient Descent** is an algorithm that minimizes functions. Given a function defined by a set of parameters \( \theta \), gradient descent starts with an initial set of parameter values and iteratively moves toward a set of parameter values that minimize the cost function. This iterative minimization is achieved using calculus, taking steps in the negative direction of the function gradient. We define the hypothesis function as

\[
h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i.
\]

Given the above hypothesis, let us try to figure out the parameters \( \theta \), which minimizes the square of the error between the predicted value \( h_\theta(x) \) and the actual output \( y \) for all values \( i \) in the training set. For that reason, let us define the cost function as

\[
J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x) - y^{(i)} \right)^2.
\]

where \( m \) is the number of training set. The scaling by fraction \( \frac{1}{2m} \) is just for notational convenience. Let us start with some parameter vector \( \theta = 0 \), and keep changing the \( \theta \) to reduce the cost function \( J(\theta) \), i.e.,

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

3.2 Matlab exercise

In this part we proceed with the implementation of the exercise. The following steps provide a guidance for the code implementation.

**Step 1: Data**  
Download ex2Data.zip, and extract the files from the zip file. The files contain some example measurements of heights for various boys between the ages of two and eight. The \( y \)-values are the heights measured in meters, and the \( x \)-values are the ages of the boys corresponding to the heights.

Each height and age tuple constitutes one training example \((x^{(i)}, y^{(i)})\) in our data set. There are \( m = 50 \) training examples, and you will use them to develop a linear regression model.

**Step 2: Supervised Learning Problem**  
In this problem, you’ll implement linear regression using gradient descent. In Matlab, you can load the training set using the function `load`. This will be our training set for a supervised learning problem with \( n = 1 \) features (in addition to the usual \( x_0 = 1 \), so \( x \in \mathbb{R}^2 \) ). Using the function `plot` try to plot your training set and label the axes.

**Step 3: Intercept point.**  
Before starting gradient descent, we need to add the \( x_0 = 1 \) intercept term to every example. To do this in Matlab, first store the number of the training samples using the function `length` and then add a column vector filled with 1’s to vector \( x \). From this point on, you will need to remember that the age values from your training data are actually in the second column of \( x \). This will be important when plotting your results later.
Step 4: Implement Gradient Descent. Recall that the linear regression model is

\[ h_\theta(x) = \theta^T x = \sum_{i=0}^{n} \theta_i x_i, \]

and the batch gradient descent update rule is

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

Implement gradient descent using a learning rate of \( \alpha = 0.07 \). Since Matlab index vectors starting from 1 rather than 0, you’ll probably use theta(1) and theta(2) in Matlab to represent \( \theta_0 \) and \( \theta_1 \). Initialize the parameters to \( \theta = 0 \) (i.e., \( \theta_0 = \theta_1 = 0 \)), and run one iteration of gradient descent from this initial starting point. Record the value of \( \theta_0 \) and \( \theta_1 \) that you get after this first iteration.

Step 5: Iterations. Continue running gradient descent for more iterations until \( \theta \) converges. (this will take a total of about 1500 iterations). After convergence, record the final values of \( \theta_0 \) and \( \theta_1 \) that you get.

When you have found \( \theta \), plot the straight line fit from your algorithm on the same graph as your training data. The plotting commands will look something like this:

```matlab
hold on;
plot(x(:,2), x*theta, '-');
legend('Training data', 'Linear regression')
```

Note that for most machine learning problems, \( x \) is very high dimensional, so we don’t be able to plot \( h_\theta(x) \). But since in this example we have only one feature, being able to plot this gives a nice sanity-check on our result.

Step 6: Finally, we’d like to make some predictions using the learned hypothesis. Use your model to predict the height for a two boys of age 3.5 and age 7.  

Debugging If you are using Matlab and seeing many errors at runtime, try inspecting your matrix operations to check that you are multiplying and adding matrices in ways that their dimensions would allow. Remember that Matlab by default interprets an operation as a matrix operation. In cases where you don’t intend to use the matrix definition of an operator but your expression is ambiguous to Matlab, you will have to use the ‘dot’ operator to specify your command. Additionally, you can try printing \( x \), \( y \), and \( \theta \) to make sure their dimensions are correct.

3.3 Optional step: Understanding the Gradient Descent

We’d like to understand better what gradient descent has done, and visualize the relationship between the parameters \( \theta \in \mathbb{R}^2 \) and \( J(\theta) \). In this problem, we’ll plot \( J(\theta) \) as a 3D surface plot. (When applying learning algorithms, we don’t usually try to plot \( J(\theta) \) since usually \( \theta \in \mathbb{R}^n \) is very high-dimensional so that we don’t have any simple way to plot or visualize \( J(\theta) \). But because the example here uses a very low dimensional \( \theta \in \mathbb{R}^2 \), we’ll plot \( J(\theta) \) to gain more intuition about linear regression). Recall that the formula for \( J(\theta) \) is
\[ J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\theta}(x^{(i)}) - y^{(i)})^2 \]

To get the best viewing results on your surface plot, use the range of theta values that we suggest in the code skeleton below.

```matlab
J_vals = zeros(100, 100); % initialize J_vals to 100x100 matrix of 0's
theta0_vals = linspace(-3, 3, 100);
theta1_vals = linspace(-1, 1, 100);
for i = 1:length(theta0_vals)
    for j = 1:length(theta1_vals)
        t = [theta0_vals(i); theta1_vals(j)];
        J_vals(i,j) = % YOUR CODE HERE %
    end
end

% Plot the surface plot
% 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';
surf(theta0_vals, theta1_vals, J_vals)
xlabel('\theta_0');
ylabel('\theta_1');
```