Programming Exercise 1: Linear Regression

Carson Slater
Deep Learning - Spring 2026

February 5, 2026

1 Introduction

The primary objective of this exercise is to implement linear regression to predict profits for a food truck based on city population (univariate) and house prices based on size and number of bedrooms (multivariate).

2 Univariate Linear Regression

2.1 Warm-up Exercise

The first task involves implementing a simple function that returns a 5x5 identity matrix.

```
function A = warmUpExercise()
%WARMUPEXERCISE Returns the 5x5 identity matrix
  A = eye(5);
end
```

Listing 1: warmUpExercise.m

2.2 Plotting the Data and Linear Fit

Visualizing the dataset along with the regression line allows for a qualitative assessment of the model's accuracy. Figure 1 shows the scatter plot of city population and profit, overlaid with the final linear fit.

2.3 Cost Function and Gradient Descent

The cost function $J(\theta)$ is computed as the mean squared error. Gradient descent is then used to minimize $J(\theta)$.

```
function J = computeCost(X, y, theta)
%COMPUTECOST Compute cost for linear regression
  m = length(y); % number of training examples
  predictions = X * theta;
  sqErrors = (predictions - y) .^ 2;
  J = 1 / (2 * m) * sum(sqErrors);
end
```

Listing 2: computeCost.m

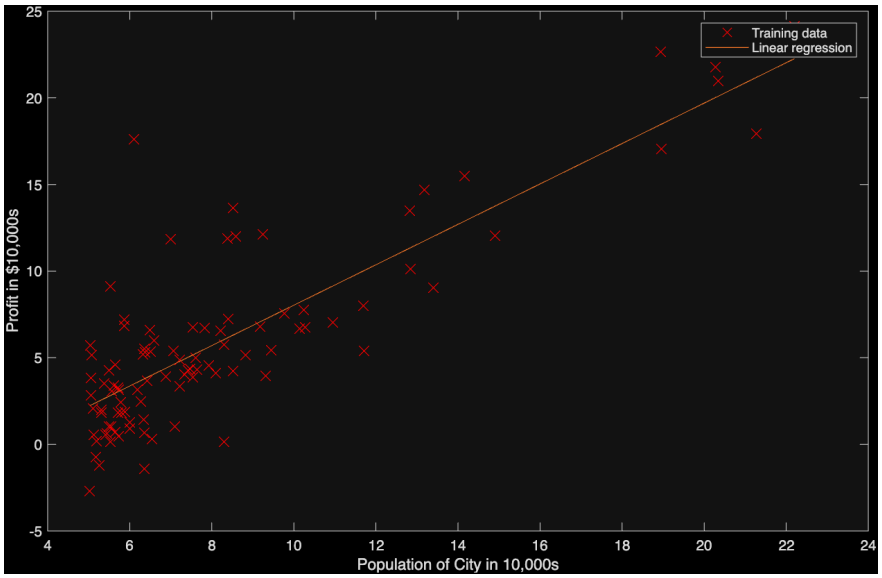

Figure 1: Scatter plot of training data with the linear regression fit.

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha,
    num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
  m = length(y);
  J_history = zeros(num_iters, 1);

  for iter = 1:num_iters
      errors = X * theta - y;
      gradient = (1 / m) * (X' * errors);
      theta = theta - alpha * gradient;
      J_history(iter) = computeCost(X, y, theta);
  end
end
```

Listing 3: gradientDescent.m

Running the algorithm with θ initialized to zeros and a learning rate $\alpha = 0.01$ yields:

- Initial Cost: 32.0727
- Optimized θ: $[-3.6303, 1.1664]$

2.4 Visualizing the Cost Function

Figures 3 and 4 show the surface and contour plots of the cost function $J(\theta)$ over a range of parameter values. The surface plot provides a 3D view of the error landscape, while the contour plot highlights the global minimum.

3 Multivariate Linear Regression

In this section, we predict house prices using two features: house size and number of bedrooms.

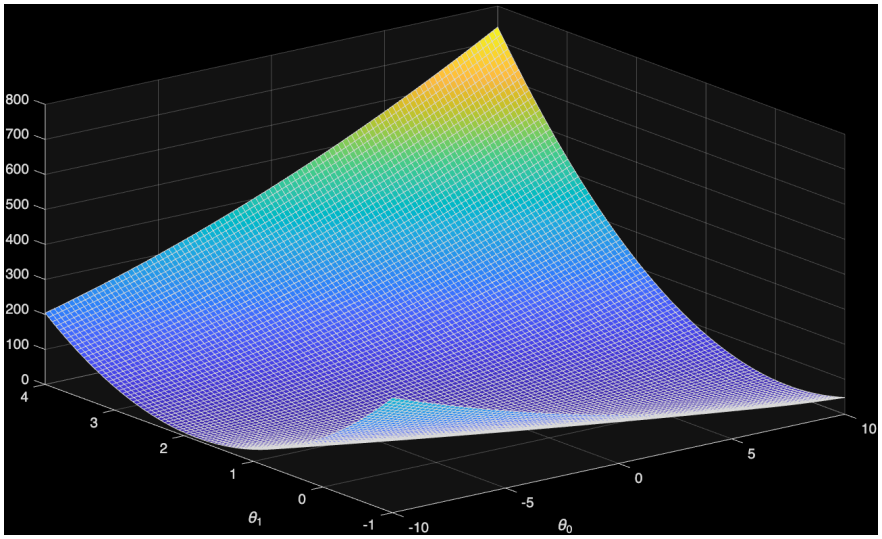

Figure 2: Surface plot of the cost function $J(\theta_0, \theta_1)$.

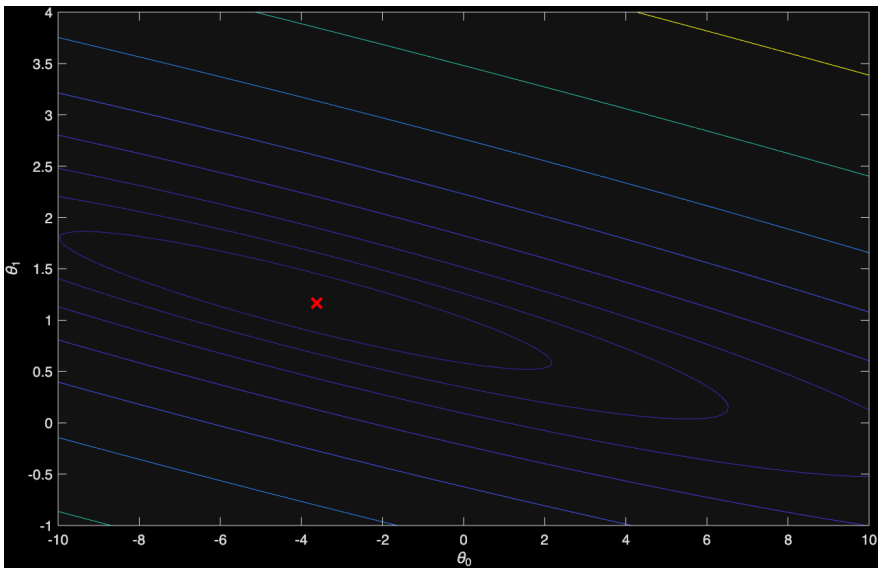

Figure 3: Contour plot of $J(\theta_0, \theta_1)$. The 'x' marks the global minimum found by gradient descent.

3.1 Feature Normalization

Since the features differ significantly in scale (house size in sq-ft vs. number of bedrooms), normalization is applied to ensure gradient descent converges efficiently.

```
function [X_norm, mu, sigma] = featureNormalize(X)
  mu = mean(X, 1);
  sigma = std(X, 0, 1);
  X_norm = (X - mu) ./ sigma;
end
```

Listing 4: featureNormalize.m

3.2 Multi-variable Gradient Descent

The multivariate cost function and gradient descent are implemented below. These functions handle an arbitrary number of features using vectorized operations.

```
function J = computeCostMulti(X, y, theta)
%COMPUTECOSTMULTI Compute cost for linear regression with multiple
   variables
  m = length(y);
  predictions = X * theta;
  sqErrors = (predictions - y) .^ 2;
  J = 1 / (2 * m) * sum(sqErrors);
end
```

Listing 5: computeCostMulti.m

```
function [theta, J_history] = gradientDescentMulti(X, y, theta, alpha,
   num_iters)
%GRADIENTDESCENTMULTI Performs gradient descent to learn theta
  m = length(y);
  J_history = zeros(num_iters, 1);

  for iter = 1:num_iters
      errors = X * theta - y;
      gradient = (1 / m) * (X' * errors);
      theta = theta - alpha * gradient;
      J_history(iter) = computeCostMulti(X, y, theta);
  end
end
```

Listing 6: gradientDescentMulti.m

3.3 Gradient Descent Convergence

The convergence of gradient descent for the multivariate case is tracked via the history of the cost function $J(\theta)$, ensuring that the objective value decreases steadily with each iteration.

3.4 Normal Equations

The normal equation provides a closed-form solution to minimizing $J(\theta)$, which does not require feature scaling.

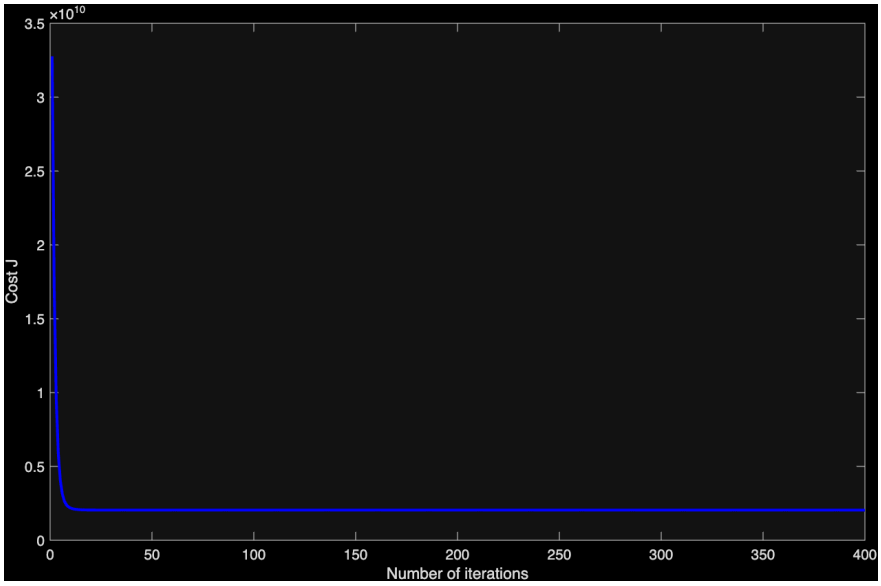

Figure 4: Convergence of the cost function J over 400 iterations for the multivariate case.

```
function [theta] = normalEqn(X, y)
  theta = pinv(X' * X) * X' * y;
end
```

Listing 7: normalEqn.m

Using both methods, we predict the price for a 1650 sq-ft house with 3 bedrooms to be approximately **$293,081.46**.

4 Main Execution Scripts

The following scripts drive the execution of the functions implemented above.

```
%% Machine Learning Online Class - Exercise 1: Linear Regression
% Initialization
clear ; close all; clc

%% Part 1: Basic Function
fprintf('Running warmUpExercise ... \n');
warmUpExercise()

%% Part 2: Plotting
data = load('ex1data1.txt');
X = data(:, 1); y = data(:, 2);
m = length(y);
plotData(X, y);

%% Part 3: Cost and Gradient descent
X = [ones(m, 1), data(:,1)];
theta = zeros(2, 1);
iterations = 1500;
alpha = 0.01;
J = computeCost(X, y, theta);
```

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

% Plot the linear fit
hold on;
plot(X(:,2), X*theta, '-')
legend('Training data', 'Linear regression')
hold off
```

Listing 8: ex1.m (Univariate Part)

```
%% Machine Learning Online Class
%  Exercise 1: Linear regression with multiple variables

%% Initialization
clear ; close all; clc

%% Part 1: Feature Normalization
data = load('ex1data2.txt');
X = data(:, 1:2);
y = data(:, 3);
m = length(y);

[X mu sigma] = featureNormalize(X);
X = [ones(m, 1) X];

%% Part 2: Gradient Descent
alpha = 0.3;
num_iters = 400;
theta = zeros(3, 1);
[theta, J_history] = gradientDescentMulti(X, y, theta, alpha, num_iters);

%% Part 3: Normal Equations
data = csvread('ex1data2.txt');
X = data(:, 1:2);
y = data(:, 3);
X = [ones(m, 1) X];
theta = normalEqn(X, y);
```

Listing 9: ex1_multi.m (Multivariate Part)

5 Execution Output

The following is the terminal output from running the main execution scripts `ex1.m` and `ex1_multi.m`, demonstrating successful completion of both parts of the exercise.

```
Running warmUpExercise ...
5x5 Identity Matrix:
Program paused. Press enter to continue.
Plotting Data ...
Program paused. Press enter to continue.

Testing the cost function ...
With theta = [0 ; 0]
Cost computed = 32.072734
```

```
Expected cost value (approx) 32.07

With theta = [-1 ; 2]
Cost computed = 54.242455
Expected cost value (approx) 54.24
Program paused. Press enter to continue.

Running Gradient Descent ...
Theta found by gradient descent:
-3.630291
1.166362
Expected theta values (approx)
 -3.6303
  1.1664

For population = 35,000, we predict a profit of 4519.767868
For population = 70,000, we predict a profit of 45342.450129
Program paused. Press enter to continue.
Visualizing J(theta_0, theta_1) ...
```

Listing 10: ex1.m Output

```
Loading data ...
First 10 examples from the dataset:
 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
Program paused. Press enter to continue.
Normalizing Features ...
Running gradient descent ...
Theta computed from gradient descent:
 340412.659574
 110631.050279
 -6649.474271

Predicted price of a 1650 sq-ft, 3 br house (using gradient descent):
 $293081.464335
Program paused. Press enter to continue.
Solving with normal equations...
Theta computed from the normal equations:
 89597.909543
 139.210674
 -8738.019112

Predicted price of a 1650 sq-ft, 3 br house (using normal equations):
 $293081.464335
```

Listing 11: ex1_multi.m Output

6 Conclusion

The implementation successfully demonstrates the application of linear regression. Gradient descent and the normal equations provide consistent results, confirming the correctness of the vectorized MATLAB implementation.