Compute J(\theta):

```
function J = computeCostMulti(X, y, theta)
%COMPUTECOSTMULTI Compute cost for linear regression
with multiple variables
%   J = COMPUTECOSTMULTI(X, y, theta) computes the cost
of using theta as the
%   parameter for linear regression to fit the data
points in X and y

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = (0.5/m) * transpose(y-X*theta)*(y-X*theta);

% ====================== YOUR CODE HERE
======================
% Instructions: Compute the cost of a particular choice
of theta
%               You should set J to the cost.
%
=======================================================
==================
end
```

Compute \theta^{n+1}:

```
function [theta, J_history] = gradientDescentMulti(X,
y, theta, alpha, num_iters)
%GRADIENTDESCENTMULTI Performs gradient descent to
learn theta
%   theta = GRADIENTDESCENTMULTI(x, y, theta, alpha,
num_iters) updates theta by
%   taking num_iters gradient steps with learning rate
alpha
% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);
overallDim = size(theta,1);
```

```
for iter = 1:num_iters
    % ====================== YOUR CODE HERE
======================
    % Instructions: Perform a single gradient step on
the parameter vector
    %               theta.
    %
    % Hint: While debugging, it can be useful to print
out the values
    %       of the cost function (computeCostMulti) and
gradient here.
    %
    newtheta = zeros(overallDim,1);
    for j = 1:overallDim
      pDiff = 1/m *ones(1,m)*((X*theta-y).*X(:,j));
      newtheta(j) = theta(j)-alpha*pDiff;
    end
    theta = newtheta;
    %
========================================================
=====
    % Save the cost J in every iteration
    J_history(iter) = computeCostMulti(X, y, theta);
end
end
```

Compute Normalization:

```
function [X_norm, mu, sigma] = featureNormalize(X)
%FEATURENORMALIZE Normalizes the features in X
%   FEATURENORMALIZE(X) returns a normalized version of
X where
%   the mean value of each feature is 0 and the
standard deviation
%   is 1. This is often a good preprocessing step to do
when
%   working with learning algorithms.
% You need to set these values correctly
m = size(X,1);
d = size(X,2);
X_norm = zeros(m,d);
```

```
mu = mean(X);
Z = X - ones(m,1)*mu;
Sigma = 1/(m-1) * (Z'*Z);
sigma = diag(Sigma)';
for i=1:d
   if sigma(i) == 0
      X_norm(:,i) = Z(:,i);
      continue
   end
   X_norm(:,i) = ( Z(:,i) ./ sqrt(sigma(i)) );
end

% ====================== YOUR CODE HERE
======================
% Instructions: First, for each feature dimension,
compute the mean
%               of the feature and subtract it from the
dataset,
%               storing the mean value in mu. Next,
compute the
%               standard deviation of each feature and
divide
%               each feature by it's standard deviation,
storing
%               the standard deviation in sigma.
%
%               Note that X is a matrix where each column
is a
%               feature and each row is an example. You
need
%               to perform the normalization separately
for
%               each feature.
%
% Hint: You might find the 'mean' and 'std' functions
useful.
%
%
=========================================================
=====
end
```

Compute Plot:

```
plot(x,y,'rx','MarkerSize',10);
```

Compute Normal Equation:

```
function [theta] = normalEqn(X, y)
%NORMALEQN Computes the closed-form solution to linear
regression
%   NORMALEQN(X,y) computes the closed-form solution to
linear
%   regression using the normal equations.


theta = pinv(X'*X)*X'*y;
% ====================== YOUR CODE HERE
======================
% Instructions: Complete the code to compute the closed
form solution
%               to linear regression and put the result
in theta.
%

% ---------------------- Sample Solution --------------
---------
% -------------------------------------------------------
---------
%
=======================================================
=====

end
```