```
clear;
clc;

m=500; % Number of observations, feel free to change around
n=3; % Number of features, feel free to change around
Feature_scaling = 1; % 0 for NO feature scaling, 1 for feature scaling

iter_method = 'Tolerance'; % Could be either 'Tolerance' or 'FixedIter'

% Tolerance value (norm of gradient matrix);
tol = 1e-5;

% Number of iterations for 'FixedIter' method
num_iter = 10000;

% LEARNING RATE, Feel free to experiment with this rate
alpha = 0.01;
```

```
% Fixed offset (random), scaling the cofficients by 10, feel free to change
beta0=10*rand();

% Coefficients of functional relationship, randomly generated with random + or - sign
beta_coeff = 10*rand(n,1); % Scaling the cofficients by 10, feel free to change
beta_sign=-1+2*round(rand(n,1));
beta_coeff = beta_coeff.*beta_sign;

% Random initiation of x-matrix (m observations with n features);
x_multiplier = 20;
x = x_multiplier*rand(m,n);

% Magnitudes of scaled x-matrix affect the convergence of algorithm
% Feature scaling may have to be enabled for large x-multipler
if (Feature_scaling==1)
    x = x/max(max(x));
end;
```

```
% Functional relationship between y and x;
% The exact exponents of x's are randomly generated i.e. p is a random variable
% Random noise added to the y vector;
```

$$y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i^p + \varepsilon\left(noise\right)$$

```
y = beta0+(x.^(0.8+0.9*rand()))*beta_coeff+10*rand();
```

```
% Define X and theta (linear regression coefficient) vectors;
% X is just x matrix appended with a first column of 1's for gradient descient run;
% Theta is (n+1) vector with a theta0 at the beginning
X = [ones(m,1) x];
theta = ones(n+1,1);
```

```
if (strcmp(iter_method,'FixedIter'))
    J_history = zeros(num_iter,1); % Error vector initialization;
    gnorm = zeros(num_iter,1); % Norm vector initialization;

    % GradientDescent loop (and computing the cost function)
    %---------------------------------------------------------
    for i = 1:num_iter
        h = X*theta;          % Hypothesis function, inner product of X and theta;
        er = h-y;             % error (difference of hypothesis and actual observation);
        er_sqr = er.^2;       % error squared
```

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

```
        J = (1/(2*m))*sum(er_sqr); % mean-squared-error (with a 1/2 factor)
```

$$\theta_j := \theta_j - \alpha\,\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).\,x_j^{(i)}$$

```
        % Partial derivative of J(theta) with respect to theta
        theta_change = (alpha/m)*(X'*(h-y));
        theta = theta-theta_change; % Update theta vector

        %Book-keeping of errors for plotting
        iter = i;
        J_history(iter) = J;
        gnorm(iter) = norm(theta_change);
        current_norm = norm(theta_change);
    end;
elseif (strcmp(iter_method,'Tolerance'))
```

```
    % GradientDescent loop (and computing the cost function)
    %---------------------------------------------------------
    current_norm = 1;
    i=1;
    J_history=[];
    while (current_norm > tol)
        h = X*theta;      % Hypothesis function, inner product of X and theta;
        er = h-y;         % error (difference of hypothesis and actual observation);
        er_sqr = er.^2;   % error squared
```

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

```
        J = (1/(2*m))*sum(er_sqr); % mean-squared-error (with a 1/2 factor)
```

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).\, x_j^{(i)}$$

```
        % Partial derivative of J(theta) with respect to theta
        theta_change = (alpha/m)*(X'*(h-y));
        theta = theta-theta_change; % Update theta vector

        %Book-keeping of errors for plotting
        iter = i;
        J_history(iter)=J;
        i=i+1;
        current_norm = norm(theta_change);
    end;
end;
```

```
% Generate predicted values from the final theta vector and compute R^2-statistic
y_hat = theta(1)+x*(theta(2:n+1));
SSE = sum((y-y_hat).^2);
SSTO = sum((y-mean(y)).^2);
r_squared = 1 - (SSE/SSTO);
```

```
% Result and comparison
beta0;              % Actual functional offset
beta_coeff;         % Actual functional coefficients
theta;              % Final linear regression coefficients
J_history(iter-1); % Show the last element of the MSE vector
regression_coeff = theta(2:n+1);
```

```
% Displaying some final results;
% Table of actual functional coefficients and regression coefficients, side-by-side
t_coeff = table(beta_coeff, regression_coeff);
msg1 = ['This was a linear regression fit with '];
msg1= [msg1, num2str(n), ' variables, and ', num2str(m), ' observations.'];
disp(msg1)
msg2 = ['Algorithm followed ',iter_method,' method and took ',num2str(iter),' steps.'];
disp(msg2)
disp(' ')
display ('-------------------------------------------')
isplay ('Original and regression coefficients Table')
display ('-------------------------------------------')
disp(t_coeff)
display ('-------------------------------------------');
r_sq_disp=['   R-squared value: ', num2str(r_squared)];
disp(r_sq_disp)
display ('-------------------------------------------');
```

```
% Plots (this section will be totally commented out)
%-----------------------------------------------------------------------------
% Scatter plot of y-actual and y-predicted;
% works for x-dimensions > 1 since it will not be possible
% to plot standard x-y scatter and linear regression line for x > 1 dimension
```

```
%scatter (1:length(x),y); hold on; scatter(1:length(x),y_hat, 'filled');
%hold off;
%hist(y-y_hat,50); % Residuals histogram, adjust number of bins for a decent plot

%scatter(x,y); hold on; plot(x, y_hat); % this is for 1-dimensional x vector only
```