```matlab
%% Machine Learning Online Class
%  Exercise 5 | Regularized Linear Regression and Bias-Variance
%
%  Instructions
%  ------------
%
%  This file contains code that helps you get started on the
%  exercise. You will need to complete the following functions:
%
%     linearRegCostFunction.m
%     learningCurve.m
%     validationCurve.m
%
%  For this exercise, you will not need to change any code in this file,
%  or any other files other than those mentioned above.
%

%% Initialization
clear ; close all; clc

%% =========== Part 1: Loading and Visualizing Data =============
%  We start the exercise by first loading and visualizing the dataset.
%  The following code will load the dataset into your environment and plot
%  the data.
%

% Load Training Data
fprintf('Loading and Visualizing Data ...\n')

% Load from ex5data1:
% You will have X, y, Xval, yval, Xtest, ytest in your environment
load ('ex5data1.mat');

% m = Number of examples
m = size(X, 1);

% Plot training data
plot(X, y, 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);
xlabel('Change in water level (x)');
ylabel('Water flowing out of the dam (y)');

fprintf('Program paused. Press enter to continue.\n');
pause;

%% =========== Part 2: Regularized Linear Regression Cost =============
%  You should now implement the cost function for regularized linear
%  regression.
%

theta = [1 ; 1];
J = linearRegCostFunction([ones(m, 1) X], y, theta, 1);
```

```
J = linearRegCostFunction([ones(m, 1) X], y, theta, 1);

fprintf(['Cost at theta = [1 ; 1]: %f '...
         '\n(this value should be about 303.993192)\n'], J);

fprintf('Program paused. Press enter to continue.\n');
pause;

%% =========== Part 3: Regularized Linear Regression Gradient =============
%  You should now implement the gradient for regularized linear
%  regression.
%

theta = [1 ; 1];
[J, grad] = linearRegCostFunction([ones(m, 1) X], y, theta, 1);

fprintf(['Gradient at theta = [1 ; 1]:  [%f; %f] '...
         '\n(this value should be about [-15.303016; 598.250744])\n'], ...
         grad(1), grad(2));

fprintf('Program paused. Press enter to continue.\n');
pause;


%% =========== Part 4: Train Linear Regression =============
%  Once you have implemented the cost and gradient correctly, the
%  trainLinearReg function will use your cost function to train
%  regularized linear regression.
%
%  Write Up Note: The data is non-linear, so this will not give a great
%                 fit.
%

%  Train linear regression with lambda = 0
lambda = 0;
[theta] = trainLinearReg([ones(m, 1) X], y, lambda);

%  Plot fit over the data
plot(X, y, 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);
xlabel('Change in water level (x)');
ylabel('Water flowing out of the dam (y)');
hold on;
plot(X, [ones(m, 1) X]*theta, '--', 'LineWidth', 2)
hold off;

fprintf('Program paused. Press enter to continue.\n');
pause;


%% =========== Part 5: Learning Curve for Linear Regression
```

```
=============
%  Next, you should implement the learningCurve function.
%
%  Write Up Note: Since the model is underfitting the data, we expect to
%                 see a graph with "high bias" -- Figure 3 in ex5.pdf
%

lambda = 0;
[error_train, error_val] = ...
    learningCurve([ones(m, 1) X], y, ...
                  [ones(size(Xval, 1), 1) Xval], yval, ...
                  lambda);

plot(1:m, error_train, 1:m, error_val);
title('Learning curve for linear regression')
legend('Train', 'Cross Validation')
xlabel('Number of training examples')
ylabel('Error')
axis([0 13 0 150])

fprintf('# Training Examples\tTrain Error\tCross Validation Error\n');
for i = 1:m
    fprintf('  \t%d\t\t%f\t%f\n', i, error_train(i), error_val(i));
end

fprintf('Program paused. Press enter to continue.\n');
pause;

%% =========== Part 6: Feature Mapping for Polynomial Regression =============
%  One solution to this is to use polynomial regression. You should now
%  complete polyFeatures to map each example into its powers
%

p = 8;

% Map X onto Polynomial Features and Normalize
X_poly = polyFeatures(X, p);
[X_poly, mu, sigma] = featureNormalize(X_poly);  % Normalize
X_poly = [ones(m, 1), X_poly];                   % Add Ones

% Map X_poly_test and normalize (using mu and sigma)
X_poly_test = polyFeatures(Xtest, p);
X_poly_test = bsxfun(@minus, X_poly_test, mu);
X_poly_test = bsxfun(@rdivide, X_poly_test, sigma);
X_poly_test = [ones(size(X_poly_test, 1), 1), X_poly_test];         % Add Ones

% Map X_poly_val and normalize (using mu and sigma)
X_poly_val = polyFeatures(Xval, p);
X_poly_val = bsxfun(@minus, X_poly_val, mu);
X_poly_val = bsxfun(@rdivide, X_poly_val, sigma);
```

```
X_poly_val = [ones(size(X_poly_val, 1), 1), X_poly_val];           % Add Ones

fprintf('Normalized Training Example 1:\n');
fprintf('  %f  \n', X_poly(1, :));

fprintf('\nProgram paused. Press enter to continue.\n');
pause;



%% =========== Part 7: Learning Curve for Polynomial Regression =============
%  Now, you will get to experiment with polynomial regression with multiple
%  values of lambda. The code below runs polynomial regression with
%  lambda = 0. You should try running the code with different values of
%  lambda to see how the fit and learning curve change.
%

lambda = 3; %% CAMBIAR ESTE LAMBDA PARA VER SU EFECTO %%%%% 1 ok, 100 chungo
            %% el mejor es 3, como se ve mas adelante
[theta] = trainLinearReg(X_poly, y, lambda);

% Plot training data and fit
figure(1);
plot(X, y, 'rx', 'MarkerSize', 10, 'LineWidth', 1.5);
plotFit(min(X), max(X), mu, sigma, theta, p);
xlabel('Change in water level (x)');
ylabel('Water flowing out of the dam (y)');
title (sprintf('Polynomial Regression Fit (lambda = %f)', lambda));

figure(2);
[error_train, error_val] = ...
    learningCurve(X_poly, y, X_poly_val, yval, lambda);
plot(1:m, error_train, 1:m, error_val);

title(sprintf('Polynomial Regression Learning Curve (lambda = %f)', lambda));
xlabel('Number of training examples')
ylabel('Error')
axis([0 13 0 100])
legend('Train', 'Cross Validation')

fprintf('Polynomial Regression (lambda = %f)\n\n', lambda);
fprintf('# Training Examples\tTrain Error\tCross Validation Error\n');
for i = 1:m
    fprintf('  \t%d\t\t%f\t%f\n', i, error_train(i), error_val(i));
end

```

```
fprintf('Program paused. Press enter to continue.\n');
pause;

%% =========== Part 8: Validation for Selecting Lambda =============
%  You will now implement validationCurve to test various values of
%  lambda on a validation set. You will then use this to select the
%  "best" lambda value.
%

[lambda_vec, error_train, error_val] = ...
    validationCurve(X_poly, y, X_poly_val, yval);

close all;
plot(lambda_vec, error_train, lambda_vec, error_val);
legend('Train', 'Cross Validation');
xlabel('lambda');
ylabel('Error');
```

```
function [J, grad] = linearRegCostFunction(X, y, theta, lambda)
%LINEARREGCOSTFUNCTION Compute cost and gradient for regularized linear
%regression with multiple variables
%   [J, grad] = LINEARREGCOSTFUNCTION(X, y, theta, lambda) computes the
%   cost of using theta as the parameter for linear regression to fit the
%   data points in X and y. Returns the cost in J and the gradient in grad

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

% You need to return the following variables correctly
J = 0;
grad = zeros(size(theta));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the cost and gradient of regularized linear
%               regression for a particular choice of theta.
%
%               You should set J to the cost and grad to the gradient.
%

Reg = lambda / (2 * m) * (theta' * theta - theta(1)^2); % término regulazacion
J = 1 / (2 * m) * sum((X * theta - y) .^2) + Reg;

forma = ones(size(theta));
forma(1) = 0;
grad = 1 / m * ((X * theta - y)' * X)' + lambda / m * (theta .* forma);

% =========================================================================

grad = grad(:);

```

```
function [error_train, error_val] = ...
    learningCurve(X, y, Xval, yval, lambda)
%LEARNINGCURVE Generates the train and cross validation set errors needed
%to plot a learning curve
%   [error_train, error_val] = ...
%       LEARNINGCURVE(X, y, Xval, yval, lambda) returns the train and
%       cross validation set errors for a learning curve. In particular,
%       it returns two vectors of the same length - error_train and
%       error_val. Then, error_train(i) contains the training error for
%       i examples (and similarly for error_val(i)).
%
%   In this function, you will compute the train and test errors for
%   dataset sizes from 1 up to m. In practice, when working with larger
%   datasets, you might want to do this in larger intervals.
%

% Number of training examples
m = size(X, 1);

% You need to return these values correctly
error_train = zeros(m, 1);
error_val   = zeros(m, 1);

% ====================== YOUR CODE HERE ======================
% Instructions: Fill in this function to return training errors in
%               error_train and the cross validation errors in error_val.
%               i.e., error_train(i) and
%               error_val(i) should give you the errors
%               obtained after training on i examples.
%
% Note: You should evaluate the training error on the first i training
%       examples (i.e., X(1:i, :) and y(1:i)).
%
%       For the cross-validation error, you should instead evaluate on
%       the _entire_ cross validation set (Xval and yval).
%
% Note: If you are using your cost function (linearRegCostFunction)
%       to compute the training and cross validation error, you should
%       call the function with the lambda argument set to 0.
%       Do note that you will still need to use lambda when running
%       the training to obtain the theta parameters.
%
% Hint: You can loop over the examples with the following:
%
%       for i = 1:m
%           % Compute train/cross validation errors using training examples
%           % X(1:i, :) and y(1:i), storing the result in
%           % error_train(i) and error_val(i)
%           ....
%
%       end
%
```

```
%

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


for i = 1:m
  % Compute train/cross validation errors using training examples
  X_sample = X(1:i,:);
  y_sample = y(1:i);

  theta = trainLinearReg(X_sample, y_sample, lambda);

  error_train(i) = linearRegCostFunction(X_sample, y_sample, theta, 0);
  error_val(i) = linearRegCostFunction(Xval, yval, theta, 0);
end


% -------------------------------------------------------------

% =========================================================================

end

```

```
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.

mu = mean(X);
X_norm = bsxfun(@minus, X, mu);

sigma = std(X_norm);
X_norm = bsxfun(@rdivide, X_norm, sigma);


% ============================================================

end

```

```
function [X_poly] = polyFeatures(X, p)
%POLYFEATURES Maps X (1D vector) into the p-th power
%   [X_poly] = POLYFEATURES(X, p) takes a data matrix X (size m x 1) and
%   maps each example into its polynomial features where
%   X_poly(i, :) = [X(i) X(i).^2 X(i).^3 ...  X(i).^p];
%


% You need to return the following variables correctly.
X_poly = zeros(numel(X), p);

% ====================== YOUR CODE HERE ======================
% Instructions: Given a vector X, return a matrix X_poly where the p-th
%               column of X contains the values of X to the p-th power.
%
%

for i = 1: p
  X_poly(:, i) = X' .^i;
end

% =========================================================================

end

```

```matlab
function [theta] = trainLinearReg(X, y, lambda)
%TRAINLINEARREG Trains linear regression given a dataset (X, y) and a
%regularization parameter lambda
%   [theta] = TRAINLINEARREG (X, y, lambda) trains linear regression using
%   the dataset (X, y) and regularization parameter lambda. Returns the
%   trained parameters theta.
%

% Initialize Theta
initial_theta = zeros(size(X, 2), 1);

% Create "short hand" for the cost function to be minimized
costFunction = @(t) linearRegCostFunction(X, y, t, lambda);

% Now, costFunction is a function that takes in only one argument
options = optimset('MaxIter', 200, 'GradObj', 'on');

% Minimize using fmincg
theta = fmincg(costFunction, initial_theta, options);

end
```

```matlab
function [lambda_vec, error_train, error_val] = ...
    validationCurve(X, y, Xval, yval)
%VALIDATIONCURVE Generate the train and validation errors needed to
%plot a validation curve that we can use to select lambda
%   [lambda_vec, error_train, error_val] = ...
%       VALIDATIONCURVE(X, y, Xval, yval) returns the train
%       and validation errors (in error_train, error_val)
%       for different values of lambda. You are given the training set (X,
%       y) and validation set (Xval, yval).
%

% Selected values of lambda (you should not change this)
lambda_vec = [0 0.001 0.003 0.01 0.03 0.1 0.3 1 3 10]';

% You need to return these variables correctly.
error_train = zeros(length(lambda_vec), 1);
error_val = zeros(length(lambda_vec), 1);

% ====================== YOUR CODE HERE ======================
% Instructions: Fill in this function to return training errors in
%               error_train and the validation errors in error_val. The
%               vector lambda_vec contains the different lambda parameters
%               to use for each calculation of the errors, i.e,
%               error_train(i), and error_val(i) should give
%               you the errors obtained after training with
%               lambda = lambda_vec(i)
%
% Note: You can loop over lambda_vec with the following:
%
%       for i = 1:length(lambda_vec)
%           lambda = lambda_vec(i);
%           % Compute train / val errors when training linear
%           % regression with regularization parameter lambda
%           % You should store the result in error_train(i)
%           % and error_val(i)
%           ....
%
%       end
%
%


for i = 1:length(lambda_vec)
  lambda = lambda_vec(i);
  theta = trainLinearReg(X, y, lambda);
  error_train(i) = linearRegCostFunction(X, y, theta, 0);
  error_val(i) = linearRegCostFunction(Xval, yval, theta, 0);
end

```