Homework 2

Zachary DeStefano, 15247592

CS 273A: Winter 2015

Due: January 20, 2015

Problem 1

Part a

Here is the code to complete part a

```
iris=load('data/curve80.txt');
y=iris(:,2);
X=iris(:,1);

%Part A
[Xtr Xte Ytr Yte] = splitData(X,y, .75); % split data into 75/25 train/test
```

Part b

Here is the code to complete part b. It does rely on the code from part a:

```
%%
%Part B
lr = linearRegress( Xtr, Ytr ); % create and train model
xs = (0:.05:10)'; % densely sample possible x-values
ys = predict( lr, xs ); % make predictions at xs

plot(xs,ys)
hold on
plot(Xtr,Ytr,'rx')
plot(Xte,Yte,'g.')
legend('Prediction','Training Data','Test Data','Location','SouthEast');

%calculate MSE
YhatTr = predict(lr,Xtr); %gets predicted y for training data
YhatTe = predict(lr,Xte); %gets predicted y for test data
mseTr = sum(abs(YhatTr-Ytr).^2);
```

Here is the plot for part b:

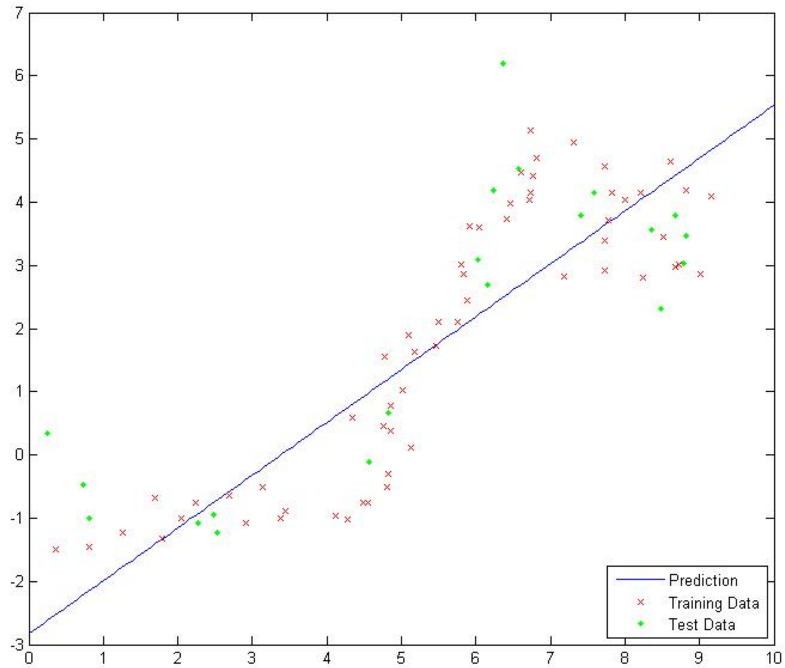

Figure 1: The training data, test data, and the predicted values

Part c

Here is the plot of the $f(x)$ functions and the training and test data

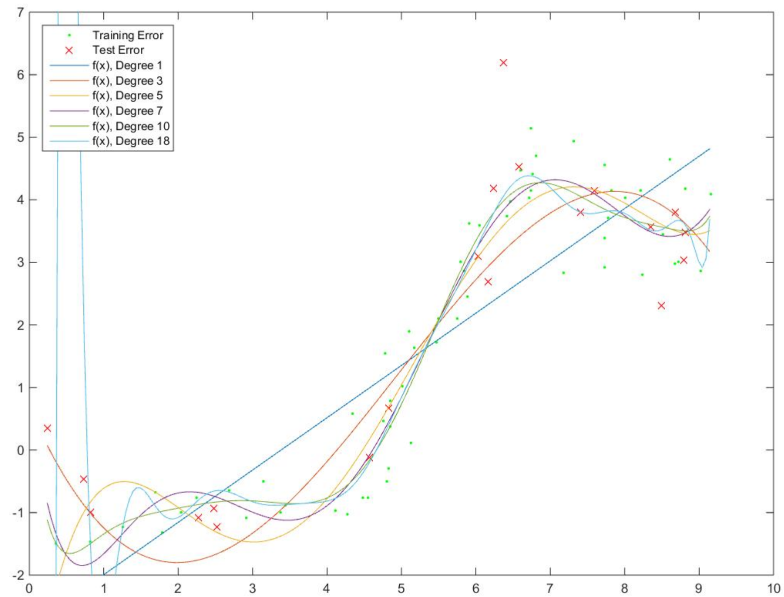

Figure 2: The training data, test data, and the best-fit polynomial

Here is the plot of the training and test error

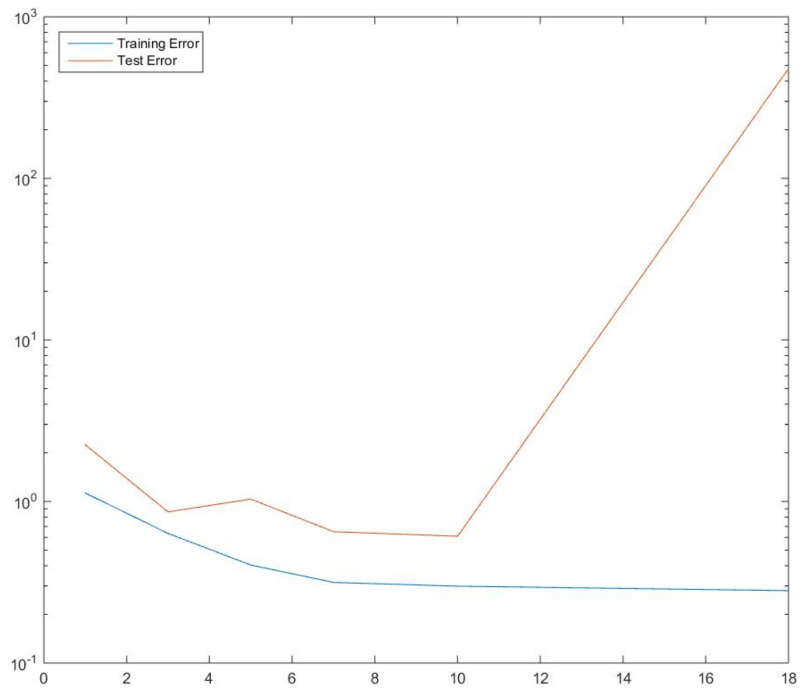

Figure 3: The training and test error

This is the code used to accomplish these plots. It is a continuation of the code from part a as it uses the arrays created there.

```
degs = [1 3 5 7 10 18];
%degs = [1 3 5];
%degs = [7 10 18];
YtrError = zeros(1,length(degs));
YteError = zeros(1,length(degs));
xs = (min(X):.05:max(X))'; % densely sample possible x-values

figure
plot(Xtr,Ytr,'g.');
hold on
plot(Xte,Yte,'rx');
ax = axis;

for i = 1:length(degs)

    degree = degs(i);

   % create poly features up to given degree; no "1" feature
    XtrP = fpoly(Xtr, degree, false);

    [XtrP, M,S] = rescale(XtrP); % it's often a good idea to scale the features
    lr = linearRegress( XtrP, Ytr ); % create and train model

    % defines an "implicit function" Phi(x)
    Phi = @(x) rescale( fpoly(x,degree,false), M,S);

    % parameters "degree", "M", and "S" are memorized at the function definition
    % Now, Phi will do the required feature expansion and rescaling:
    YhatTrain = predict( lr, Phi(Xtr) ); % predict on training data
    YhatTest = predict(lr, Phi(Xte) );

    ys = predict( lr, Phi(xs) ); % make predictions at xs
    plot(xs,ys)
    axis(ax)

    %now get the training and test error
    YtrError(i) = sum((YhatTrain-Ytr).^2)/length(Ytr);
    YteError(i) = sum((YhatTest-Yte).^2)/length(Yte);

end

%creates the legend for the f(x) plots
%   degs = [1 3 5 7 10 18];
legend('Training Error','Test Error','f(x), Degree 1','f(x), Degree 3',...
    'f(x), Degree 5','f(x), Degree 7','f(x), Degree 10','f(x), Degree 18',...
    'Location','Northwest');

%creates the training and test error plots
figure
semilogy(degs,YtrError);
hold on
```

Problem 3

This is the plot. As can be observed, the minimum average MSE occurs where $degree = 7$. The average cross-validation MSE for the degree 7 polynomial was 0.5138.

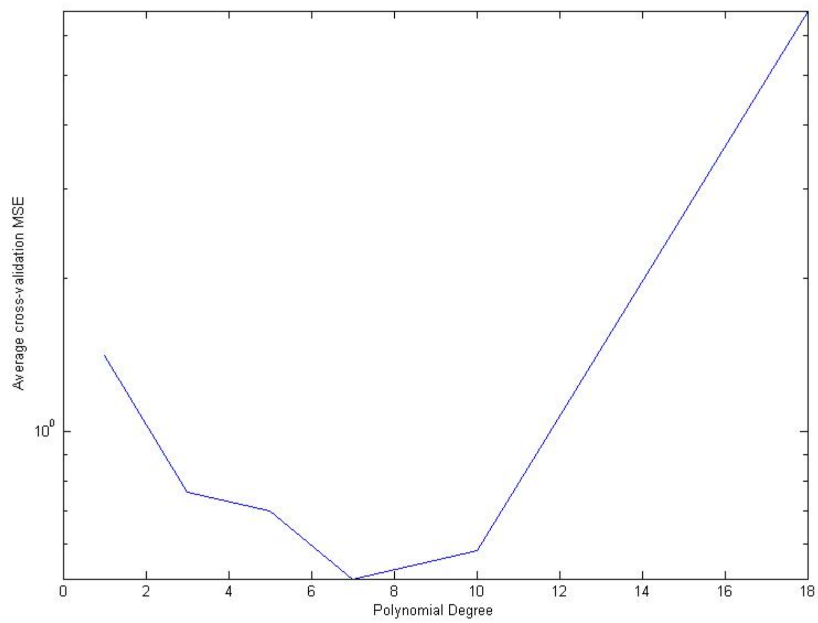

Figure 4: The average cross-validation MSE as function of polynomial degree

The test error for the degree 7 polynomial in problem 2 was 0.6502.
Therefore using cross-validation reduced the test error slightly.

Using cross-validation also changed which polynomial degree had the least test error. In problem 2, the degree 10 polynomial had the least test error whereas in problem 3, the degree 7 polynomial performed the best.

Here is the code that I used to get the numbers and the plot for problem 3

```
%%
iris=load('data/curve80.txt');
y=iris(:,2);
X=iris(:,1);
%%

degs = [1 3 5 7 10 18];
crossValidError = zeros(1,length(degs));

nFolds = 5;
J = zeros(1,nFolds);

for j=1:length(degs)

    degree = degs(j);

    for iFold = 1:nFolds,

        % take ith data block as validation
        [Xti,Xvi,Yti,Yvi] = crossValidate(X,y,nFolds,iFold);

        XtrP = fpoly(Xti, degree, false);

        [XtrP, M,S] = rescale(XtrP); % it's often a good idea to scale the features
        lr = linearRegress( XtrP, Yti ); % create and train model

        % defines an "implicit function" Phi(x)
        Phi = @(x) rescale( fpoly(x,degree,false), M,S);

        % parameters "degree", "M", and "S" are memorized at the function definition
        % Now, Phi will do the required feature expansion and rescaling:
        YhatTest = predict(lr, Phi(Xvi) );

        J(iFold) = sum((YhatTest-Yvi).^2)/length(Yvi);
    end;

    % the overall estimated validation performance is the average of the performance on each fold
    crossValidError(j) = mean(J);
end

semilogy(degs,crossValidError);
xlabel('Polynomial Degree');
ylabel('Average cross-validation MSE');
```