1. *Solution:*

   Steps to execute the code:

   a. In the command prompt enter **clearvars; problem1_model('data2.mat',6);** Use different values of dimension so as to plot different graphs.

   b. In the command prompt enter **clearvars; problem1_crossvalid('data1.mat');**

Step 1:

Created a new function named it problem1_model with input parameters - filename, dimension

problem1_model.mat

```
function problem1_model(filename, dimension)
   data = load(filename); %this loads the file data
   X_data = data.x;
   Y_data = data.y;
   %divide the dataset test and train
   [length,~] = size(X_data);% get the size of the dataset to be
used for crossfold
   idenx = crossvalind('KFold', length, 2);
   trainMat = (idenx == 1);
   x_train = X_data(trainMat);
   y_train = Y_data(trainMat);

   testMat = (idenx == 2);
   x_test = X_data(testMat);
   y_test = Y_data(testMat);

   cv_err_test = [];
   cv_err_train = [];
   [errTrain, model, errTest] = polyreg(x_train, y_train,
dimension, x_test, y_test);
   cv_err_test(dimension) = errTest;
   cv_err_train(dimension) = errTrain;

   errorValue = ['errTrain= ',  num2str(errTrain) , ' errTest= ',
num2str(errTest)];
   disp(errorValue);

End
```

Step 2:.

Created a new function named it problem1_crossvalid with input parameters - filename.

problem1_crossvalid.mat

```
function problem1_crossvalid(filename)
  data = load(filename); %this loads the file data
  X_data = data.x;
  Y_data = data.y;
  %divide the dataset test and train
```

```
        [length,~] = size(X_data);% get the size of the dataset to be used
for crossfold
        idenx = crossvalind('KFold', length, 2);
        trainMat = (idenx == 1);
        x_train = X_data(trainMat);
        y_train = Y_data(trainMat);

        testMat = (idenx == 2);
        x_test = X_data(testMat);
        y_test = Y_data(testMat);

        cv_err_test = [];
        cv_err_train = [];
        for d=1:40
            [errTrain, model, errTest] = polyreg(x_train, y_train, d,
x_test, y_test);
            cv_err_test(d) = errTest;
            cv_err_train(d) = errTrain;
        end

        clf ;
        hold on ;
        plot(cv_err_train,'b');
        plot(cv_err_test,'r');
        % get the minimum of the error_test value and mark as cross on
graph
        [~, i ] = min(cv_err_test);
        plot(i , cv_err_test(i) , 'bx') ;
        %graph specifications
        title('Cross Validation');
        xlabel('Error');
        ylabel('Degree of Polynomial');
        legend('Train Error', 'Test Error');
    end
```

Step 2:

Reused the given polyreg.mat function and modified to change the aesthetics of the graph.

polyreg.mat

```
    function [errTrain,model,errTest] =
polyreg(xTrain,yTrain,dimension,xTest,yTest)
    %
    % Finds a D-1 order polynomial fit to the data
    %
    %    function [errTrain,model,errTest] = polyreg(x,y,D,xT,yT)
    %
    % x = vector of input scalars for training
    % y = vector of output scalars for training
```

```
% D = the order plus one of the polynomial being fit
% xT = vector of input scalars for testing
% yT = vector of output scalars for testing
% errTrain = average squared loss on training
% model = vector of polynomial parameter coefficients
% errTest = average squared loss on testing
%
% Example Usage:
%
% x = 3*(rand(50,1)-0.5);
% y = x.*x.*x-x+rand(size(x));
% [errTrain,model] = polyreg(x,y,4);
%
xx = zeros(length(xTrain),dimension);
for i=1:dimension
 xx(:,i) = xTrain.^(dimension-i);
end
model = pinv(xx)*yTrain;
errTrain   = (1/(2*length(xTrain)))*sum((yTrain-xx*model).^2);
if (nargin==5)
 xxT = zeros(length(xTest),dimension);
 for i=1:dimension
   xxT(:,i) = xTest.^(dimension-i);
 end
 errTest  = (1/(2*length(xTest)))*sum((yTest-xxT*model).^2);
end
q  = (min(xTrain):(max(xTrain)/300):max(xTrain))';
qq = zeros(length(q),dimension);
for i=1:dimension
 qq(:,i) = q.^(dimension-i);
end
clf
plot(xTrain,yTrain,'X');
hold on
if (nargin==5)
   plot(xTest,yTest,'cO');
end
plot(q,qq*model,'r')
title('Dimension :' , num2str(dimension), 'FontSize', 8)
legend('Training Data', 'Testing Data', 'Model');
```

Step 3:

Called the above created main1 function from the command line.

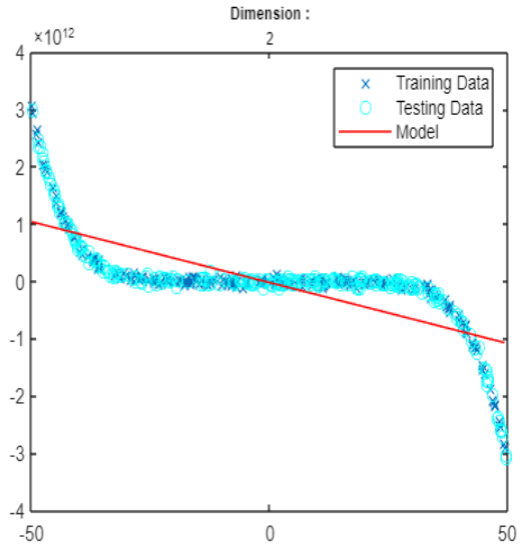
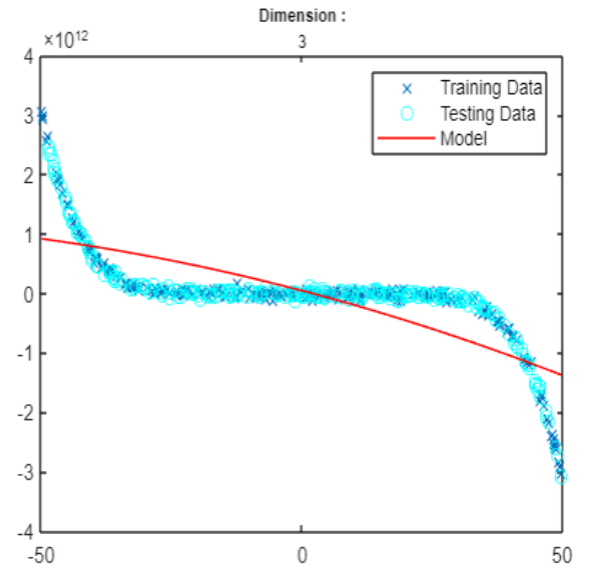
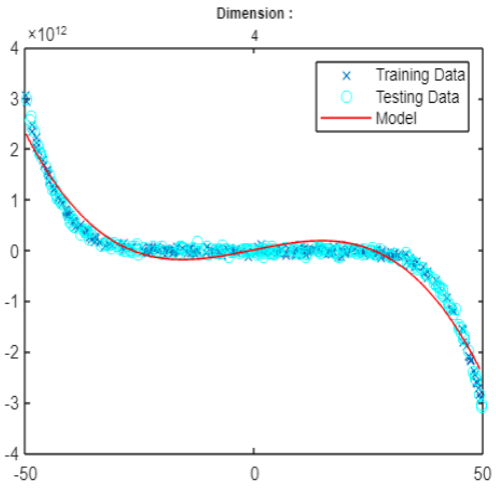
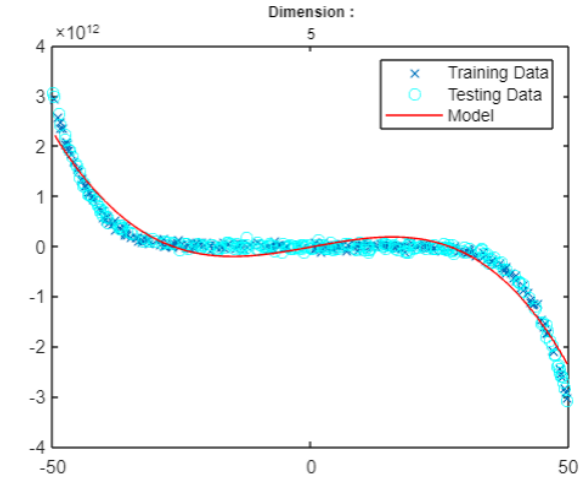
`problem1_model('data1.mat',1);` where data1.mat is the dataset and 1 is the dimension. Changed the second parameter values to get different graphs for dimension 2 to 8.

Note: since i am unable to install matlab on my local machine hence i could not use for loops for calculating the dimensions.

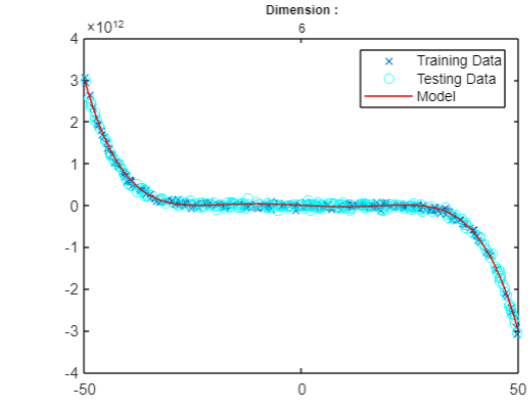
I had to pass dimensions one by one to plot the graph and then paste it in this document.

This function internally calls the polyreg function and passes the testing and training data to the polyreg function.

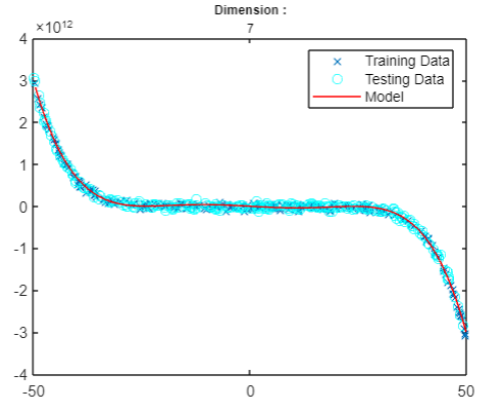
We tried the graph plotting for many dimensions and found the below graphs:

<table>
<tr>
<td>When dimension = 2
errTrain= 1.424401426218483e+23 errTest= 1.50959660582098e+23
 
</td>
<td>When dimension = 3
errTrain= 1.62849280818654e+23 errTest= 1.340692505304569e+23
 
</td>
</tr>
<tr>
<td>When dimension = 4
errTrain= 1.989487008376291e+22 errTest= 2.089966420334334e+22
 
</td>
<td>When dimension = 5
errTrain= 1.944145471380704e+22 errTest= 2.101959095650357e+22
 
</td>
</tr>
<tr>
<td>When dimension = 6</td>
<td>When dimension = 7</td>
</tr>
</table>

errTrain= 1.163397244354462e+21 errTest= 1.144847098536642e+21

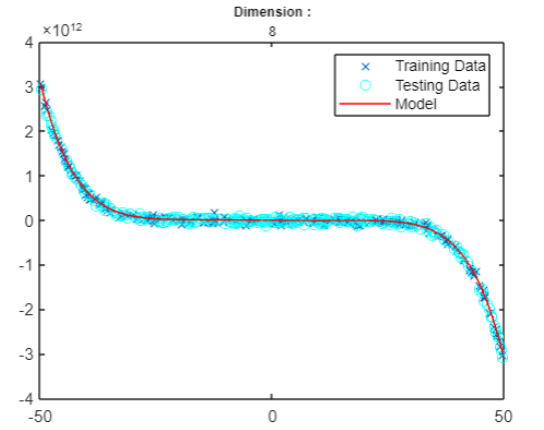


errTrain= 1.332588028583983e+21 errTest= 1.659525936232841e+21



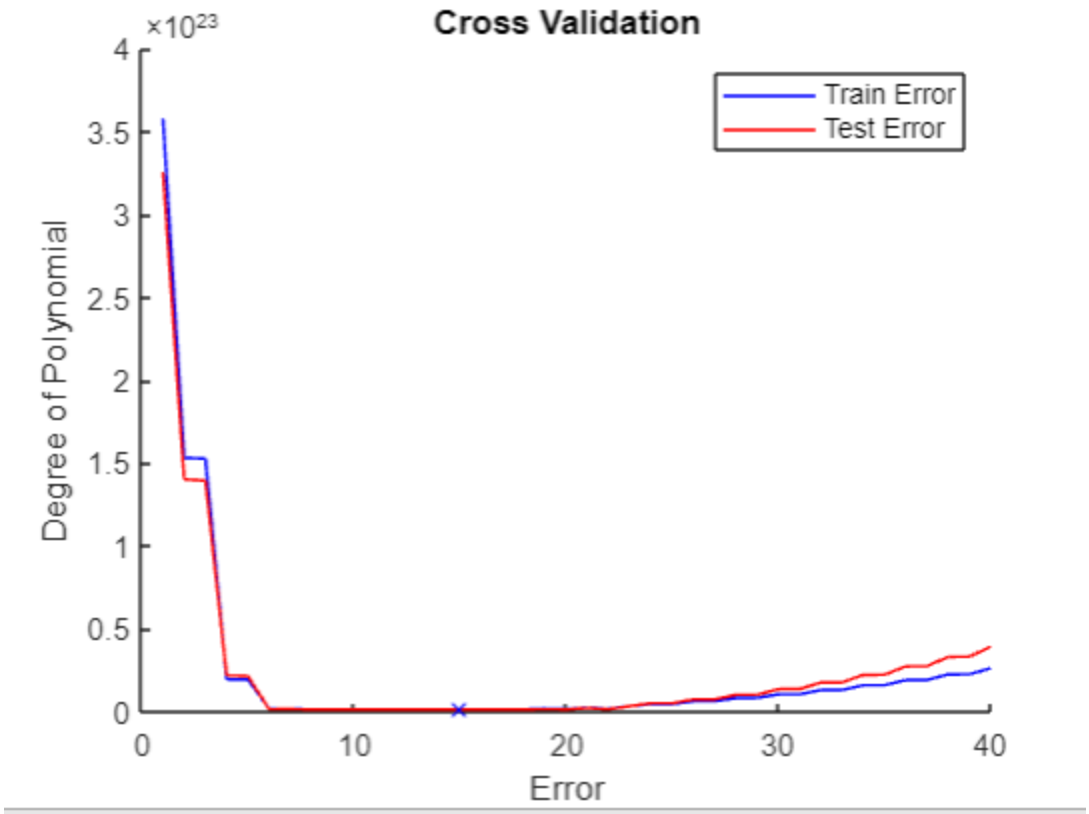
When dimension = 8

errTrain= 1.23802542677666e+21 errTest= 1.037595276859651e+21



We tried multiple plots for dimension =6 and higher value of d overfits the data. Hence dimension 6 is reasonable.

After Cross validation:



2. *Solution:*

We use the model equation:

$$I = X^T * X$$

$$model = \theta * = (X^T * X + \lambda I)^{-1} * X^T y$$

$$R(emp) = (0.5/N) * \sum N\,(y(i) - f(x;\,\theta))^2$$

These formulas are implemented in riskreg.m

riskreg.m

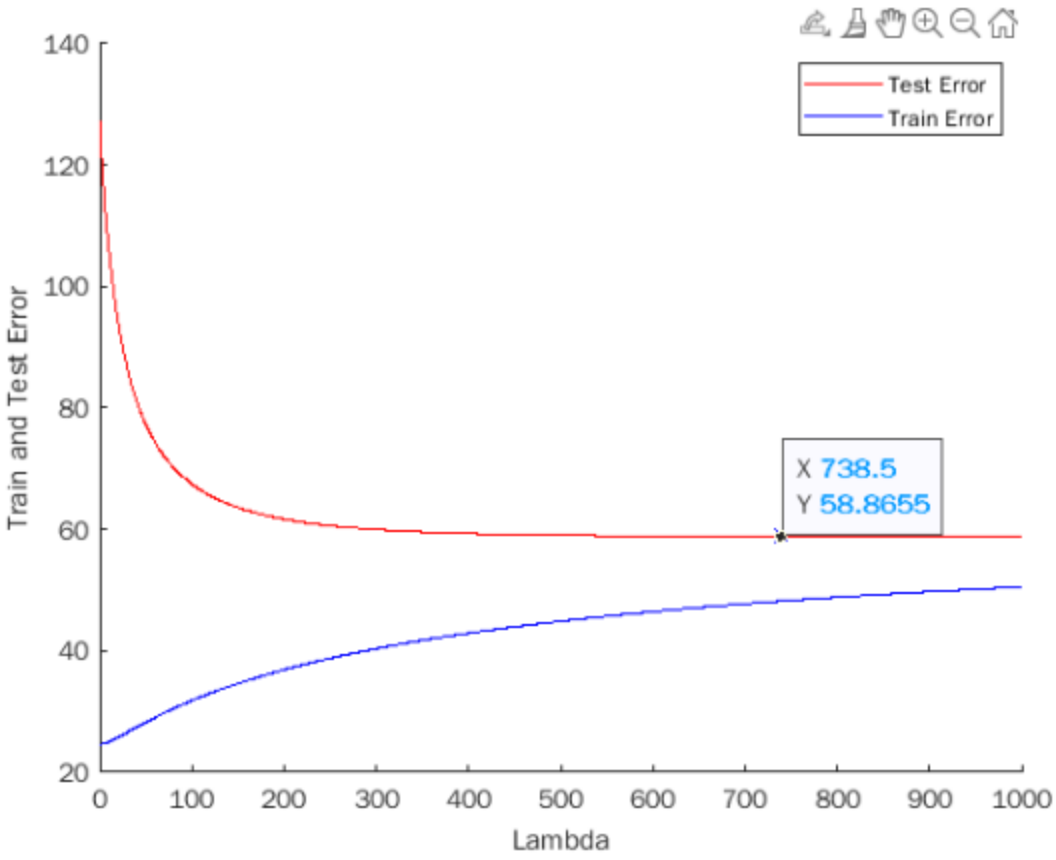
```
function [errTrain, model, errTest] = riskreg(xTrain, yTrain, lambda,
xTest, yTest)

   xtx = (xTrain' * xTrain);
   model = (xTrain' * xTrain + lambda * eye(size(xtx))) \ (xTrain' *
yTrain);
   errTrain = (1/(2*length(xTrain)))*(sum((yTrain - xTrain*model).^2));
   if(nargin == 5)
          errTest = (1/(2*length(xTest)))*(sum((yTest -
xTest*model).^2));
   end
End
```

From the below function we call the riskreg function to calculate the testerror and the trainerror which will then be used to calculate the graph of (testerror and trainerror) vs lambda.

main2.mat

```
function main2(filename)
   load(filename); % this loads the dataset from the file

   %divide the dataset test and train
   [length,~] = size(x); % get the size of the dataset to be used for
crossfold
   idenx = crossvalind('Kfold ', length, 2);
   x_train = x(idenx == 1, :);
   y_train = y(idenx == 1);
   x_test = x(idenx == 2, :);
   y_test = y(idenx == 2);

   cv_err_test = [];
   cv_err_train = [];
   d= 1;
   lambdas = 0:0.5:2000;
   % we use a for loop here so as to avoid passing individual values of
lambda.
   for lambda = lambdas
       [errTrain, model, errTest] = riskreg(x_train, y_train, lambda,
x_test, y_test);
       cv_err_test(d) = errTest;
       cv_err_train(d) = errTrain;
       d= d + 1;
   end
   close all;
   hold on;

   plot(lambdas, cv_err_test, 'r');
   plot(lambdas, cv_err_train, 'b');

   [~, i ] = min(cv_err_test);
   plot(i, cv_err_test(i) , 'bx');

   %graph specifications
   xlabel('Lambda');
   ylabel('Train and Test Error');
   legend('Test Error', 'Train Error');
```

From the graph we can see that when the lambda value is around **738.5** the testerror and train error has the minimum distance.
As the value of lambda increases the value of test error decreases.



3. *Solution:*

We need to prove,

$when\ equation\ g(z) = 1/(1 + exp(-z))$ is true then it satisfies the property,

$g(-z) = 1 - g(z)$

$g(z) = 1/(1 + e^{-z}))$

$= e^{z}/(1 + e^{z})$

$g(-z) = e^{-z}/(1 + e^{-z})$

$= 1/(1 + e^{z})$ $---------- LHS$

$1 - g(z) = 1 - e^{z}/(1 + e^{z})$

$= 1/(1 + e^{z})$ ----------RHS

Hence LHS = RHS.

To prove inverse $g^{-1}(y) = ln(y/(1 - y))$.

Assume $g(z) = y$

$hence\, z = g^{-1}(y)$

Substituting this in the logistic squashing function

$y = 1/(1 + e^{-z})$

$y(1 + e^{z}) = e^{z}$

$e^{z} * (1 - y)/y = 1$

$e^{z} = y/(1 - y)$

Taking log on both sides

$z = log(y / (1 - y))$

Substitute $z = g^{-1}(y)$

Hence $g^{-1}(y) = log(y/(1 - y))$

4. $f(x; \theta) = (1 + e^{(-\theta * X)})^{-1}$

$Remp(\theta) = (1/N) * \sum (yi - 1)\, log(1 - f(xi; \theta)) - yi\, log(f(xi; \theta))$

Solving in parts:

$log(1 - f(xi; \theta)) = log(1 - 1/(1 + e^{-\theta * X})) = log((1 + e^{-\theta * X} - 1)/(1 + e^{-\theta * X}))$

$log(e^{-\theta * X}/(1 + e^{-\theta * X})) = log(e^{-\theta * X}) - log(1 + e^{-\theta * X})$ ---------- part 1

$log(1/(1 + e^{-\theta * X})) = log(1) - log(1 + e^{-\theta * X}) = - log(1 + e^{-\theta * X})$---------- part 2

$Remp(\theta) = (1/N) * \sum (yi - 1) * (log(e^{-\theta * X}) - log(1 + e^{-\theta * X})) - yi * (- log(1 + e^{-\theta * X}))$

$$\nabla_{\theta} R = \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)\frac{d}{d\theta} log(\frac{e^{-\theta_T x_i}}{1+e^{-\theta_T x_i}}) - y_i \frac{d}{d\theta}(- log(1 + e^{-\theta_T x_i}))$$

$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)[(- x_i) - \frac{1}{1+e^{-\theta_T x_i}}(e^{-\theta_T x_i})(- x_i)] + y_i \frac{d}{d\theta}(\frac{1}{1+e^{-\theta_T x_i}}(e^{-\theta_T x_i})(- x_i))$$

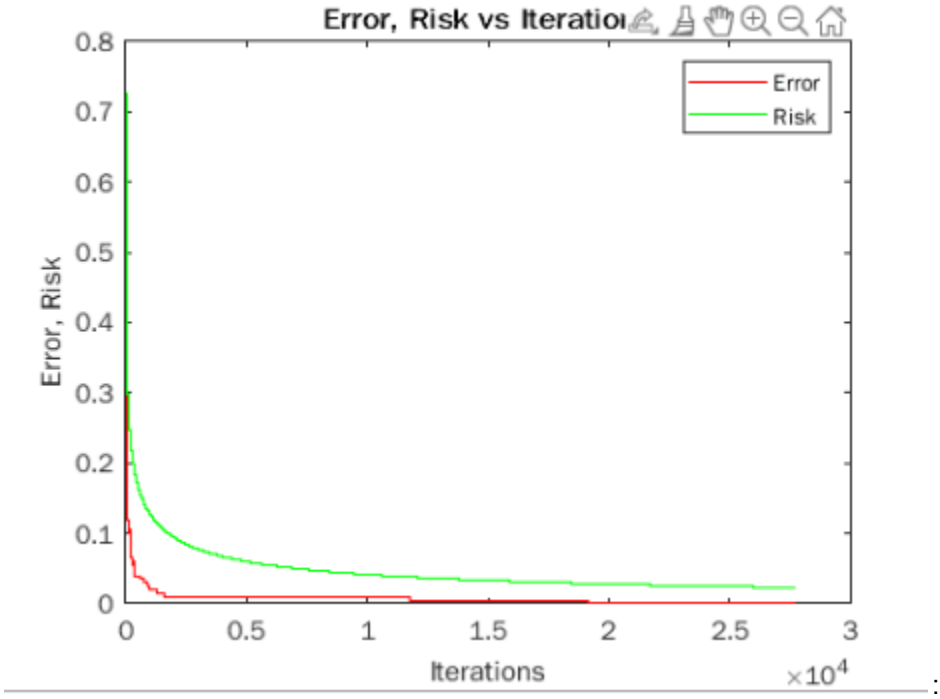
$$= \frac{1}{N}\sum_{i=1}^{N}(y_i - 1)[(- x_i) + \frac{e^{-\theta_T x_i} * x_i}{1+e^{-\theta_T x_i}}] - y_i \frac{d}{d\theta}(\frac{e^{-\theta_T x_i} * x_i}{1+e^{-\theta_T x_i}})$$

$$= \frac{1}{N}\sum_{i=1}^{N}(1 - y_i)[x_i - \frac{e^{-\theta_T x_i} * x_i}{1+e^{-\theta_T x_i}}] - y_i \frac{d}{d\theta}(\frac{e^{-\theta_T x_i} * x_i}{1+e^{-\theta_T x_i}})$$

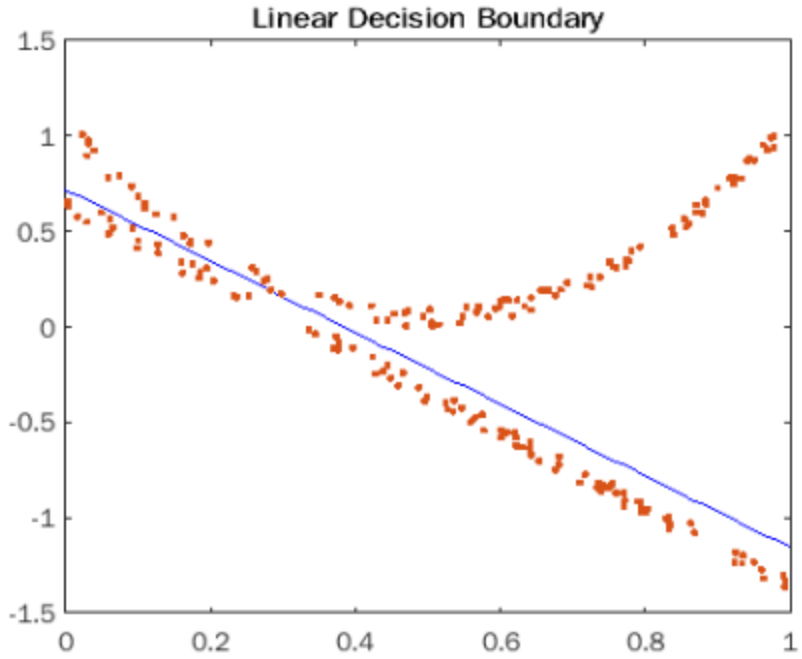
Matlab implementation.

1. calculateRisk.m
2. returnFunction.m - this returns the function that is described in the problem statement
3. calculateGradient.m - this is the equation that we derived
4. problem4.m - this is the function to be executed by passing the parameter - filename, epsilon, stepsize

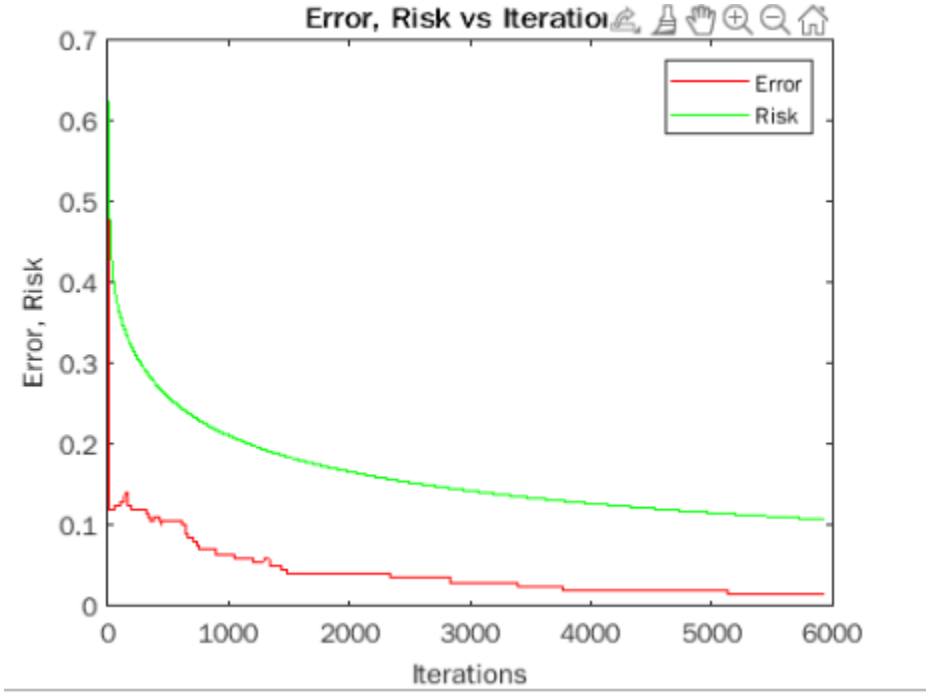
ϵ	η	θ
0.001	2.0	68.6234 36.5858 -26.4126

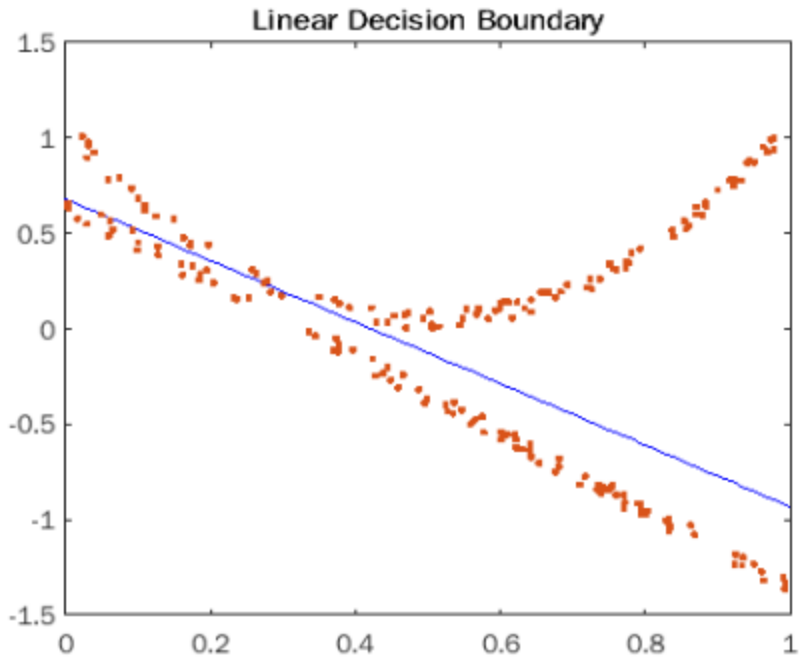


:

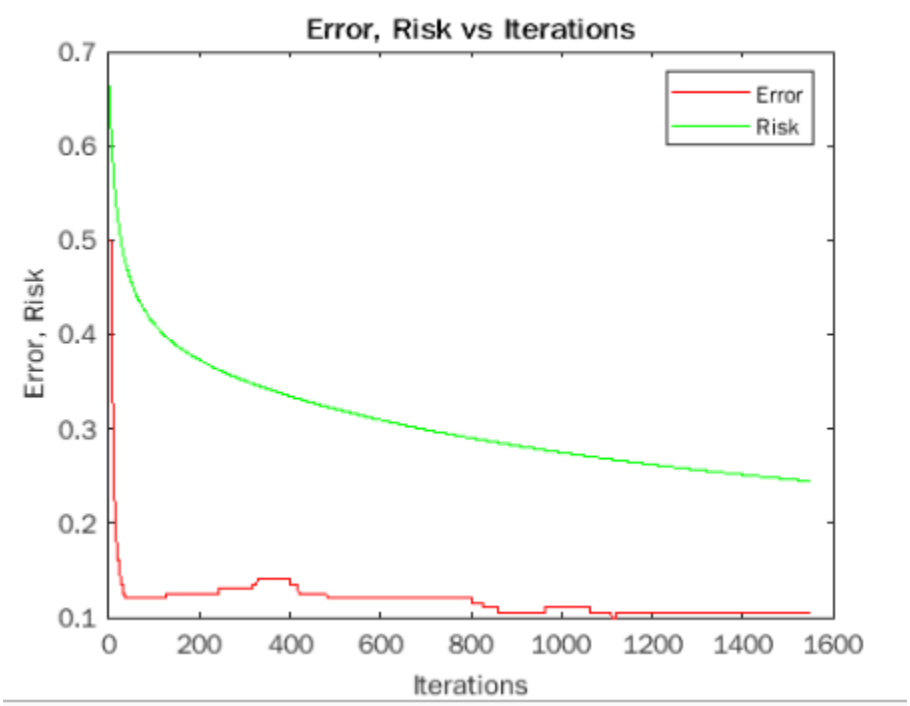


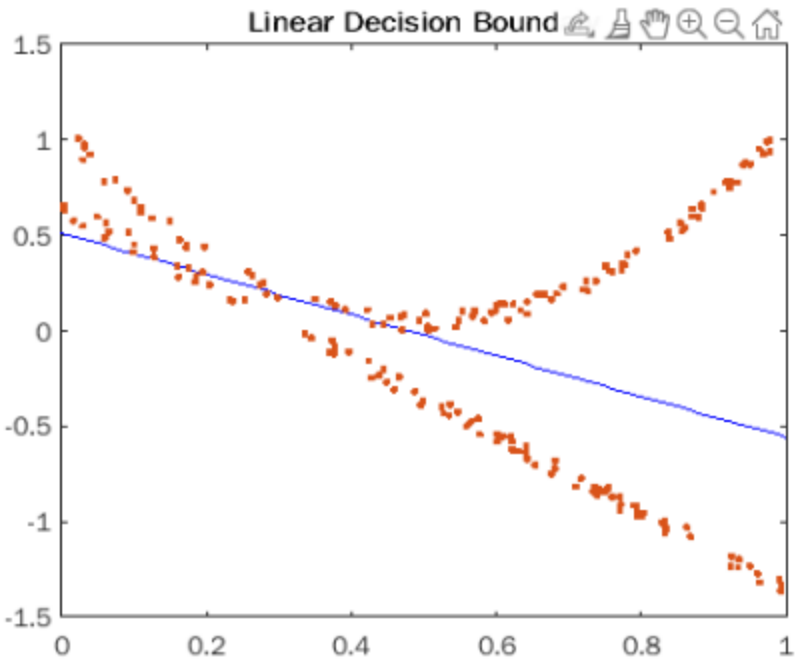
ϵ	η	θ
0.002	0.5	20.8239 12.9274 -8.8044





ϵ	η	θ
0.003	0.2	0.9989 0.6582 0.1805




Linear Decision Bound
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.2
0.4
0.6
0.8
1