MACHINE LEARNING

Assignment 1

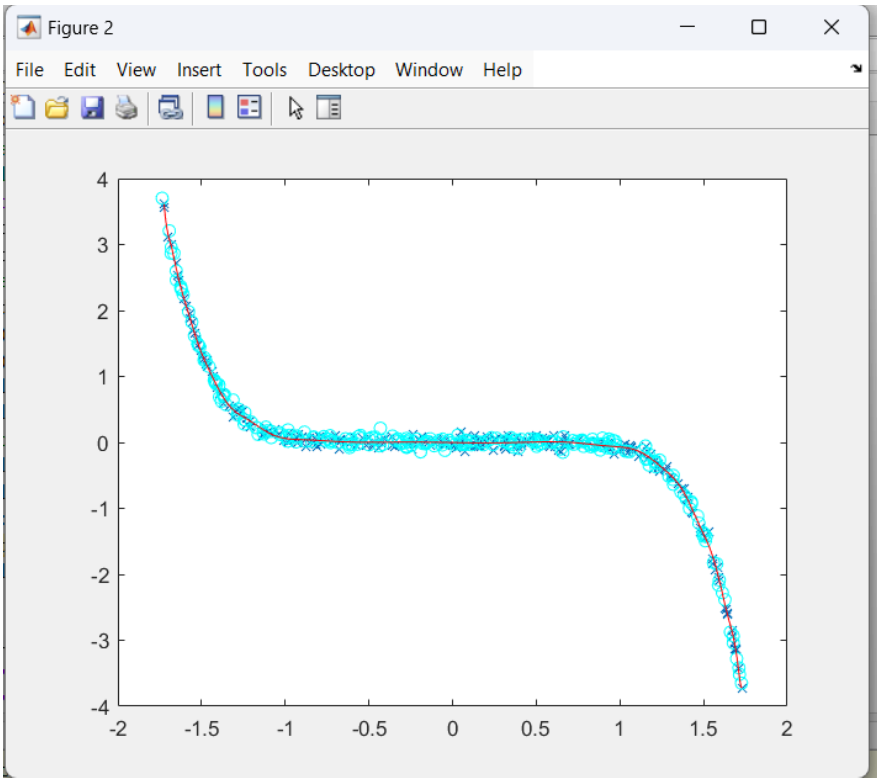
Harshini Vijaya Kumar (N11212080)

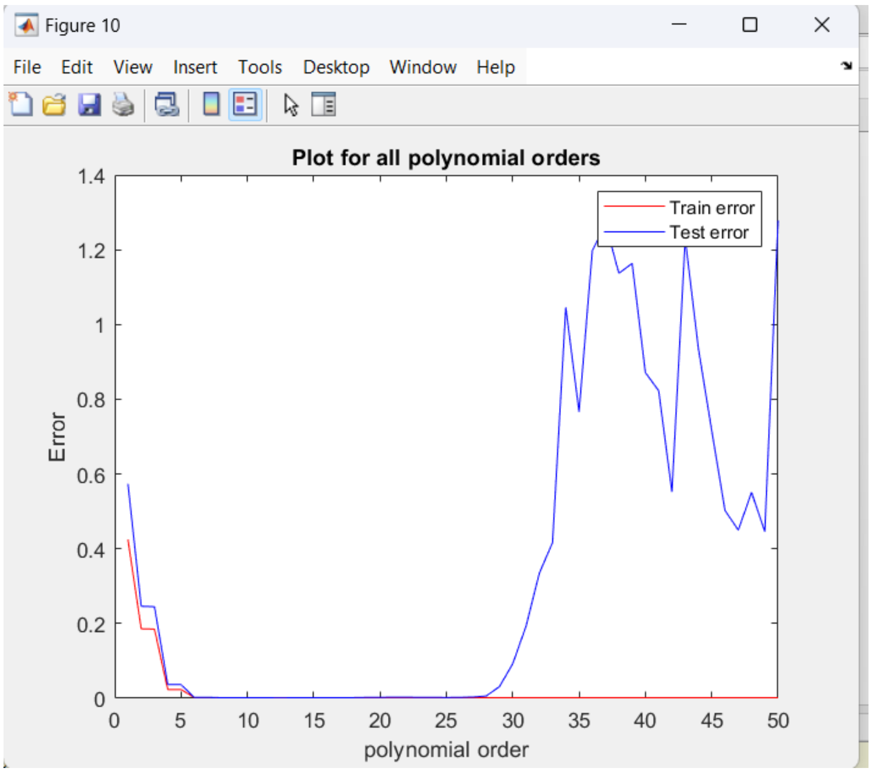
Problem 1:

```
load problem1.mat;
Lx=length(x);
Ly=length(y);
%Shuffle the data
ran=randperm(Lx);
x=x(ran);
y=y(ran);
x=normalize(x);
y=normalize(y);

%The problem asks to fit the data first without train and test split.
num_full=100;
err_full=zeros(num_full);
for m=1:num_full
[err,model] = polyreg(x,y,m);
err_full(m)=err;
end
clf
plot(err_full(1:num_full),'r');
xlabel('polynomial order');
ylabel('Error-fulldata');
title("Plot for polynomial order vs mean squared error ");

%Get train and test split
xtr=x(1:fix(Lx/2));
xts=x(fix(Lx/2)+1:end);
ytr=y(1:fix(Lx/2));
yts=y(fix(Lx/2)+1:end);

%len variable contains the maximum order of polynomial used to fit.
len= 50;
xu=1:len;
testerr=zeros(len);
trainerr=zeros(len);

%fit the model for various polynomial orders
```

```
for m=1:len
[err,model,errT] = polyreg(xtr,ytr,m,xts,yts);
testerr(m)=errT;
trainerr(m)=err;
end

%plot test and train error against the polynomial order
figure()
clf
plot(trainerr(1:len),'r');
hold on
plot(testerr(1:len),'b');
title("Plot for all polynomial orders");
legend('Train error','Test error');
xlabel('polynomial order');
ylabel('Error');

%The test error looks small in the 5 to 15 range
figure()
clf
plot(trainerr(5:15),'r');
xticklabels({5:15})
hold on
plot(testerr(5:15),'b');
xticklabels({5:15})
title("Plot to find best polynomial order");
legend('Train error','Test error');
xlabel('polynomial order');
ylabel('Error');

%The order with lowest error and small complexity is 6
%Plot the predicted data against input data
answer=6;
[err,model,errT] = polyreg(xtr,ytr,answer,xts,yts);
qq = zeros(length(x),answer);
for i=1:answer
qq(:,i) = x.^(answer-i);
end
q = 1:500 ;
figure()
clf
scatter(x,qq*model)
hold on
scatter(x,y)
```

```
legend("predicted data","true data")
title("Plot between predicted and original data");
```





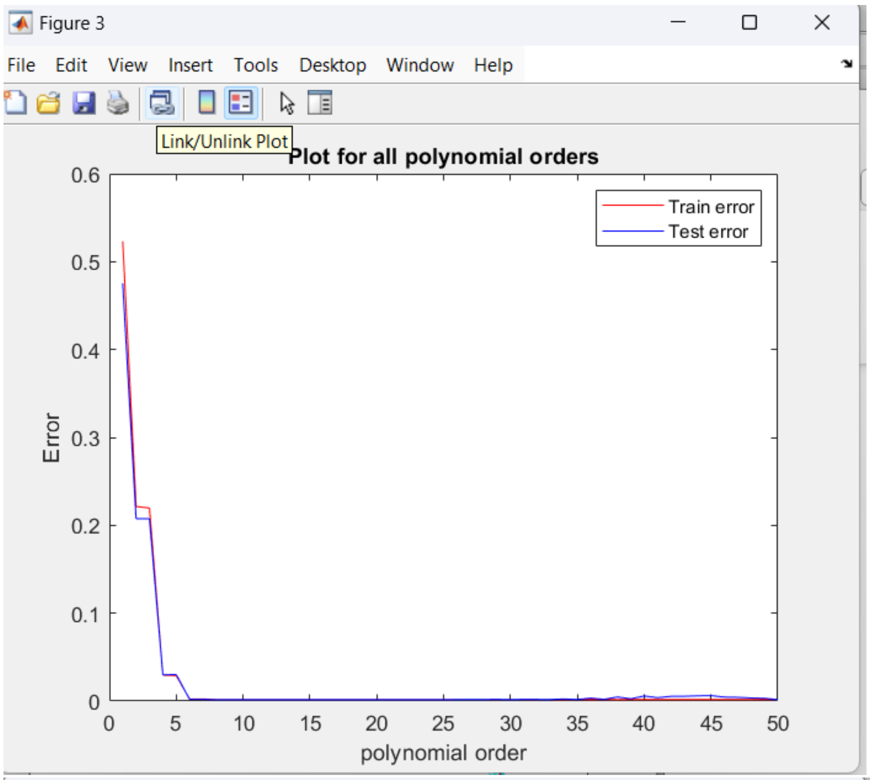


Figure 3
File Edit View Insert Tools Desktop Window Help
Link/Unlink Plot
Plot for all polynomial orders
Train error
Test error
Error
polynomial order
0.6
0.5
0.4
0.3
0.2
0.1
0
0 5 10 15 20 25 30 35 40 45 50

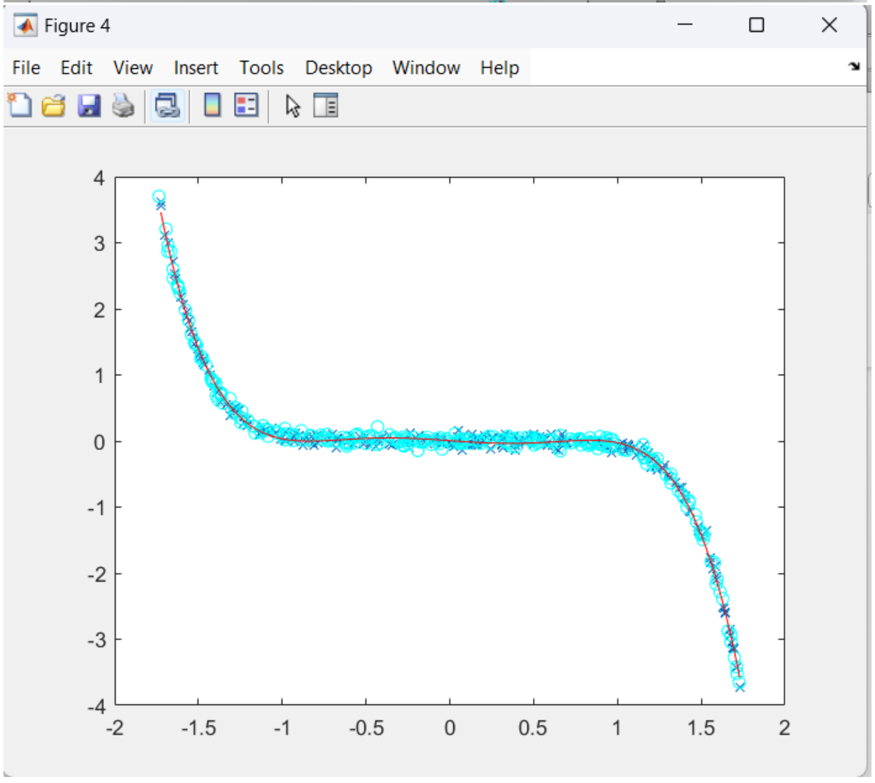
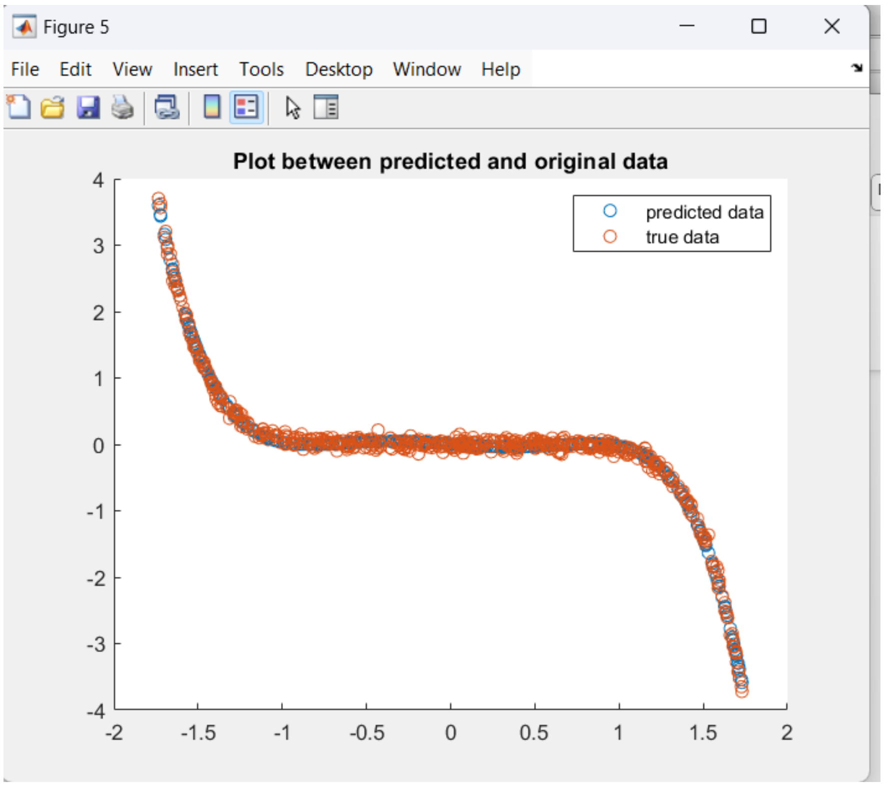


Figure 4
File Edit View Insert Tools Desktop Window Help
4
3
2
1
0
-1
-2
-3
-4
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2



Observation: The code successfully fits the data using polynomial regression without a train-test split, followed by fitting with a split. The errors decrease as the polynomial degree increases, but after a certain degree, the test error begins to rise, indicating overfitting.

Conclusion: The optimal polynomial degree (**order 6**) minimizes the test error while maintaining a balance between complexity and model fit. Increasing the polynomial order beyond this leads to overfitting, where the model fits the training data too well, but generalizes poorly to unseen data (test set).

Problem 2:

```
% Load dataset from problem2.mat
load problem2.mat;

% Get the number of samples
Lx = length(x);
Ly = length(y);

% Shuffle the data
ran = randperm(Lx);
x = x(ran, :);
y = y(ran, :);

% Normalize the data for better performance
x = normalize(x);
y = normalize(y);
```

```
% Split the data into training and testing sets
xtr = x(1:fix(Lx/2), :);
xts = x(fix(Lx/2)+1:end, :);
ytr = y(1:fix(Lx/2), :);
yts = y(fix(Lx/2)+1:end, :);

% Initialize arrays to store errors for different Lambda values
testerr = zeros(1000, 1);
trainerr = zeros(1000, 1);
testerr_unreg = zeros(1000, 1);
trainerr_unreg = zeros(1000, 1);

% Loop through different Lambda values and fit the model
index = 1;
for m = 0:1000
   [err, err_unreg, model, errT, errT_unreg] = regmulti(xtr, ytr, m, xts, yts);
   testerr(index) = errT;
   trainerr(index) = err;
   testerr_unreg(index) = errT_unreg;
   trainerr_unreg(index) = err_unreg;
   index = index + 1;
end

% Plot the regularized training and testing error vs Lambda values
clf;
plot(testerr(1:1000), 'b');
hold on;
plot(trainerr(1:1000), 'r');
title("Plot for all Lambda values with regularization");
legend('Test error', 'Train error');
xlabel('Lambda');
ylabel('Error');

% Plot unregularized training and testing error vs Lambda values
figure();
clf;
plot(testerr_unreg(1:1000), 'b');
hold on;
plot(trainerr_unreg(1:1000), 'r');
title("Plot for all Lambda values without regularization");
legend('Test error', 'Train error');
xlabel('Lambda');
ylabel('Error');
```

Output:

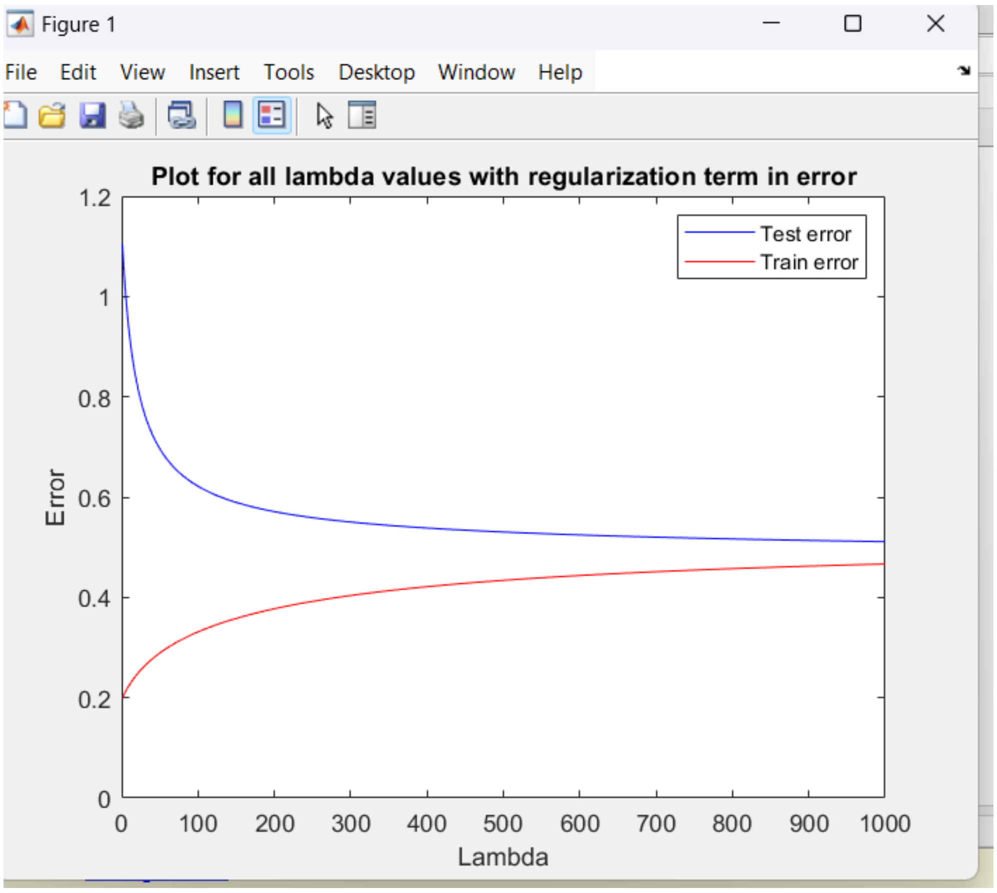


Figure 1
File Edit View Insert Tools Desktop Window Help
Plot for all lambda values with regularization term in error
Test error
Train error
Error
Lambda
0
0.2
0.4
0.6
0.8
1
1.2
0
100
200
300
400
500
600
700
800
900
1000

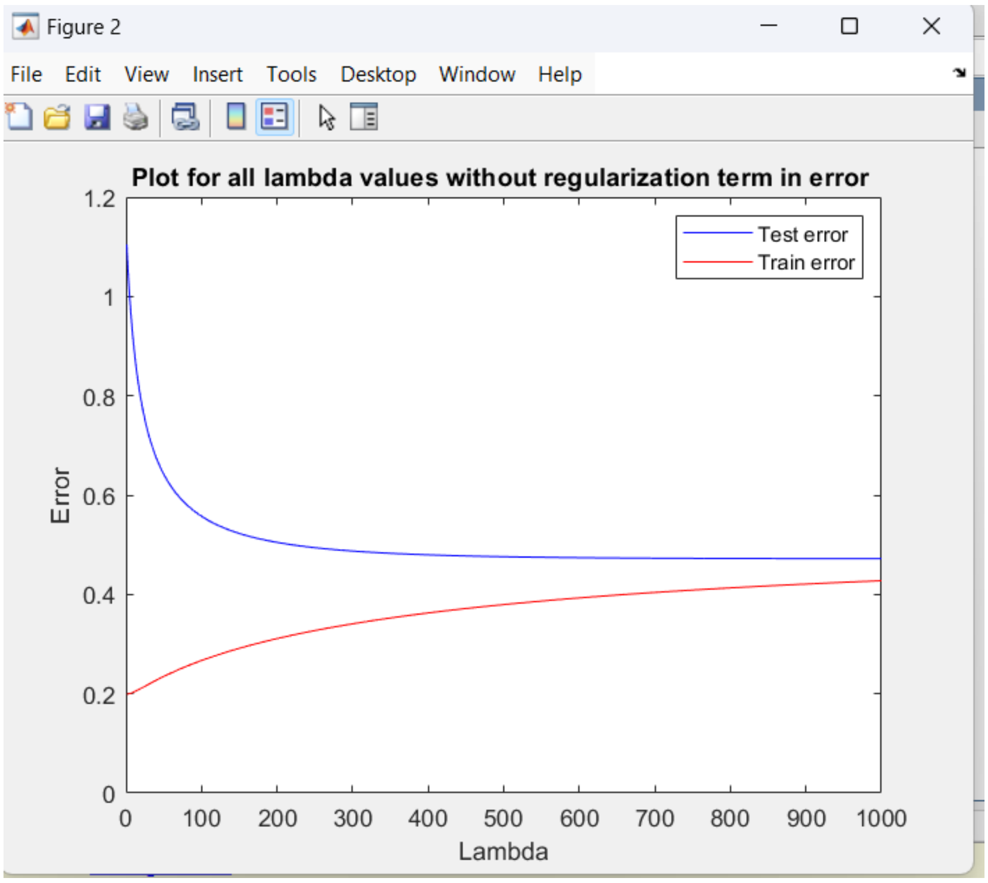


Figure 2
File Edit View Insert Tools Desktop Window Help
Plot for all lambda values without regularization term in error
Test error
Train error
Error
Lambda
0
0.2
0.4
0.6
0.8
1
1.2
0
100
200
300
400
500
600
700
800
900
1000

Observation: In Problem 2, I applied regularized multivariate regression using various values for the regularization parameter (λ), ranging from 0 to 1000. For lambda value of approximately 700 makes the train and test error stable with approximate error nearing 0.5. We see that as lambda, increases the test error decreases and then becomes stable after a while, whereas, simultaneously the train error increases and becomes stable after a certain limit.

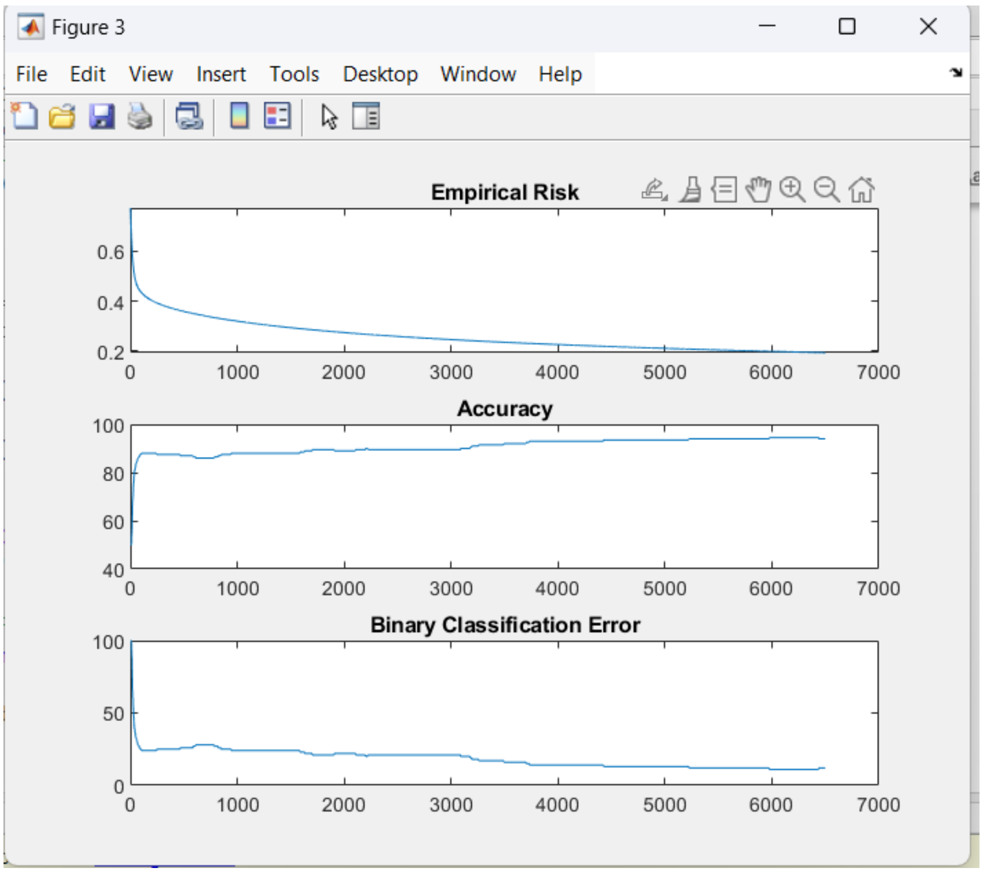
Problem 4:

```
load dataset4.mat;
shapeX = size(X);
theta = ones(shapeX(2),1); % Declaration of parameter vector
theta_prev = zeros(shapeX(2),1);
iter = 1; % Number of iterations
alpha = 0.1; % Learning rate
max_iter = 10000; % Correct syntax for setting max iterations
costs = zeros(max_iter,1);
accuracy = zeros(max_iter,1);
err = zeros(max_iter,1);
tolerance = 0.001;

% Gradient Descent Loop
while (norm(theta - theta_prev) > tolerance) && (iter < max_iter)
   [cost, grad, f] = Remp(X, Y, theta);
   theta_prev = theta;
   theta = theta - alpha * grad;
   costs(iter) = cost;
   [acc, err_temp] = Prediction(X, Y, theta);
   accuracy(iter) = acc;
   err(iter) = err_temp;
   iter = iter + 1;
end

disp("Number of iterations:");
disp(iter - 1);

% Plotting results
subplot(3,1,1)
plot(1:iter-1, costs(1:iter-1))
title("Empirical Risk")

subplot(3,1,2)
plot(1:iter-1, accuracy(1:iter-1))
```

```
title("Accuracy")
subplot(3,1,3)
plot(1:iter-1, err(1:iter-1))
title("Binary Classification Error")

% Decision Boundary Plot
figure()
mask1 = Y == 0;
mask2 = Y == 1;
X_out = X(mask1, :);
X_out1 = X(mask2, :);
XX = (-theta(3) - X(:,1) * theta(1)) / theta(2);

scatter(X_out(:,1), X_out(:,2), 'bs')
hold on
scatter(X_out1(:,1), X_out1(:,2), 'ro')
hold on
plot(X(:,1), XX, 'k')
title("Decision Boundary for the Dataset")
legend('0', '1', "Decision Boundary")

% Remp function for cost and gradient
function [cost, grad, f] = Remp(X, Y, theta)
   m = length(Y);  % Number of training examples
   grad = zeros(size(theta));
   f = sigmoid(X * theta);  % Corrected sigmoid calculation

   % Cost function with logistic regression
   cost = (-1/m) * sum(Y .* log(f) + (1 - Y) .* log(1 - f));
   % Gradient calculation
   for j = 1:length(grad)
      grad(j) = (1/m) * sum((f - Y) .* X(:, j));
   end
end
% Sigmoid function
function Y = sigmoid(z)
   Y = 1 ./ (1 + exp(-z));
end

% Prediction function for accuracy and error
function [accuracy, error] = Prediction(X, Y, theta)
   f = sigmoid(X * theta);  % Sigmoid applied to the hypothesis

   % Prediction thresholding
   f(f >= 0.5) = 1;
   f(f < 0.5) = 0;

   % Calculate errors and accuracy
   error = sum(abs(Y - f));  % Number of misclassified examples
   accuracy = 100 * (1 - error / length(Y));  % Correct accuracy calculation
end
```

Observation:

I used a learning rate/ step size (α) of 0.1 and a tolerance (ε) of 0.001 for Problem 4, and the logistic regression code converged after 6499 iterations.



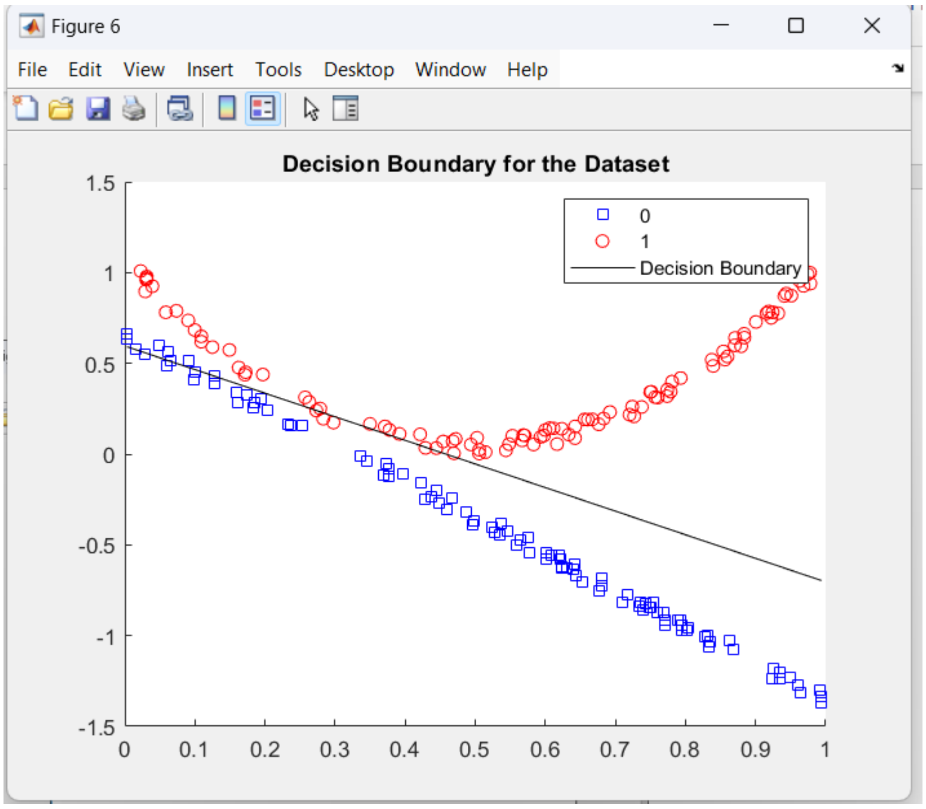


Figure 6
File Edit View Insert Tools Desktop Window Help
Decision Boundary for the Dataset
□ 0
○ 1
Decision Boundary
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1