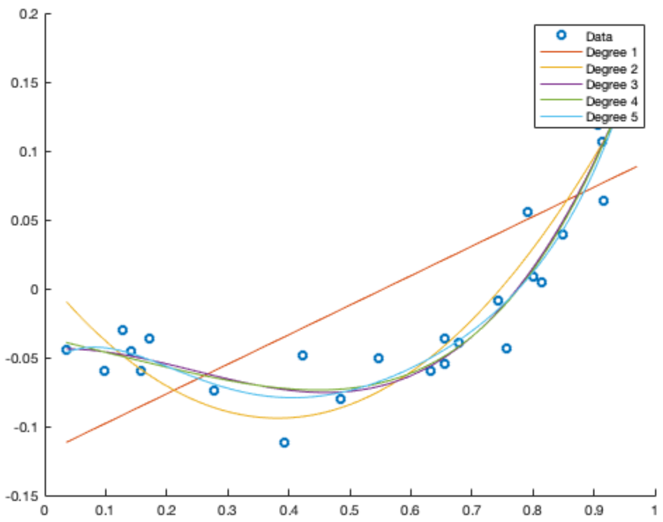
Homework 2 - Question 1

```
data = readtable('polydata.csv');
Y = data.Var2;
Y_size = size(Y);

disp("1a)");
X_d1 = create_Matrix_X(data, 1);
B_ols_d1 = fitlm(X_d1, Y);
fprintf("B_ols for polynomial degree 1 = \n");
disp(B_ols_d1.Coefficients.Estimate)

X_d2 = create_Matrix_X(data, 2);
B_ols_d2 = fitlm(X_d2, Y);
fprintf("B_ols for polynomial degree 2 = \n");
disp(B_ols_d2.Coefficients.Estimate)

X_d3 = create_Matrix_X(data, 3);
B_ols_d3 = fitlm(X_d3, Y);
fprintf("B_ols for polynomial degree 3 = \n");
disp(B_ols_d3.Coefficients.Estimate)

X_d4 = create_Matrix_X(data, 4);
B_ols_d4 = fitlm(X_d4, Y);
fprintf("B_ols for polynomial degree 4 = \n");
disp(B_ols_d4.Coefficients.Estimate)

X_d5 = create_Matrix_X(data, 5);
B_ols_d5 = fitlm(X_d5, Y);
fprintf("B_ols for polynomial degree 5 = \n");
disp(B_ols_d5.Coefficients.Estimate)

%b)Scatter plot
disp("1b)");
scatter(X_d1, Y)
hold on

myplot(X_d1, B_ols_d1.Coefficients.Estimate)
myplot(X_d1, B_ols_d2.Coefficients.Estimate)
myplot(X_d1, B_ols_d3.Coefficients.Estimate)
myplot(X_d1, B_ols_d4.Coefficients.Estimate)
myplot(X_d1, B_ols_d5.Coefficients.Estimate)
legend('Data', 'Degree 1','Degree 2', 'Degree 3', 'Degree 4', 'Degree 5')

snapnow

%c) Cross-validation
disp("1c)");
idxs = randperm(30);
fprintf("When using 5-fold cross validation, best degree polynomial = %d \n",perform_cross_validation(data, 5, 5, idxs));
disp("From the plots, it looks like the third order polynomial fits the data with some noise. However on repeating this");
disp("experiment multiple times, when the data is permuted, the best fit with varies d=3,4,5. Observed that d=3 is mostly the best fit.");
function min_idx = perform_cross_validation(data, k, d, idxs)
    average_MSE = zeros(d,1);
    Y = data.Var2;
    Y = Y(idxs);
    for j = 1:d
        X = create_Matrix_X(data, j);
        X = X(idxs, :);
        chunk_size = size(Y,1)/k;
        for i=1:k
            index = (i * chunk_size) - chunk_size;
            X_test = X(index+1:index+chunk_size, :);
            Y_test = Y(index+1:index+chunk_size, :);
            X_train = [X(1:index, :);X(index+chunk_size+1:end, :)];
            Y_train = [Y(1:index);Y(index+chunk_size+1:end)];
            B = fitlm(X_train,Y_train);
            yfit = predict(B, X_test);
            average_MSE(j) = average_MSE(j) + mean((Y_test-yfit).^2);
        end
    end
    average_MSE = average_MSE/k;
    fprintf("Average MSE for degree polynomial from d=1:%d is \n", d);
    fprintf("%f \n", average_MSE);
    [M, min_idx] = min(average_MSE);
end

function X = create_Matrix_X(data, d)
    Y = data.Var2;
    X = zeros(size(Y,1),d);
    for k = 1:d
        X(:, k) = data.Var1.^k;
    end
end

function myplot(X, B)
t = linspace(min(X),max(X));
plot(t, polyval(flipud(B), t))
end
```

```
1a)
B_ols for polynomial degree 1 =
   -0.1191
    0.2142

B_ols for polynomial degree 2 =
    0.0088
   -0.5382
    0.7046

B_ols for polynomial degree 3 =
   -0.0442
    0.0331
   -0.5857
    0.7963

B_ols for polynomial degree 4 =
   -0.0353
   -0.1075
   -0.0113
   -0.0447
    0.4024

B_ols for polynomial degree 5 =
   -0.0567
    0.3964
   -3.3845
    8.7537
   -9.3436
    3.8535

1b)
```



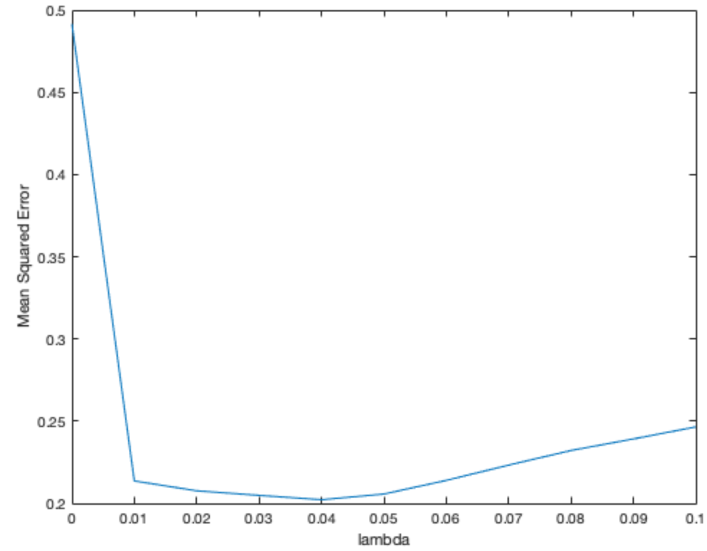

```
1c)
Average MSE for degree polynomial from d=1:5 is
0.003221
0.000625
0.000419
0.000474
0.000630
When using 5-fold cross validation, best degree polynomial = 3
From the plots, it looks like the third order polynomial fits the data with some noise. However on repeating this
experiment multiple times, when the data is permuted, the best fit with varies d=3,4,5. Observed that d=3 is mostly the best fit.
```

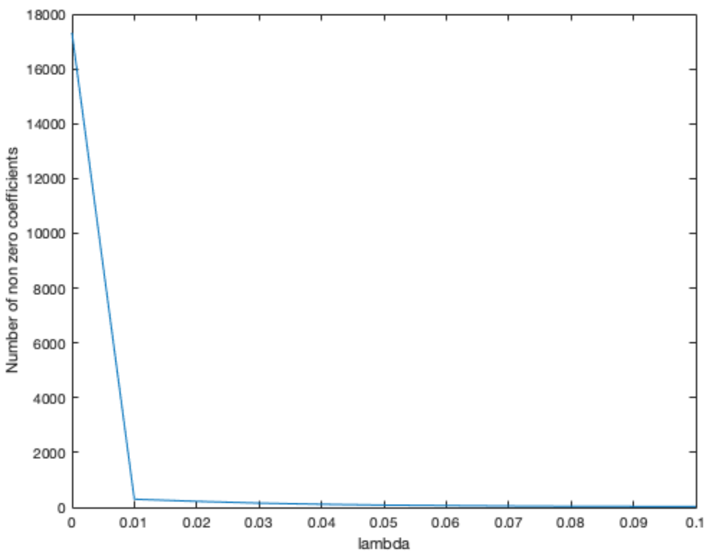
Published with MATLAB® R2018a

```
% Homework 2 - Question 2
data = csvread('brca.csv');

lambda = generate_lambda(0, 0.1, 0.01);
Y = data(:, end);
X = data(:, 1:end-1);

%a)
disp("2a) 5-fold CV")
perform_cross_validation(X,Y,5, lambda, 1);
disp("Average MSE and Sparsity as a function of lambda")
snapnow
%b)
disp("")
disp("2b) 10-fold CV")
perform_cross_validation(X,Y,10, lambda, 3);
disp("Average MSE and Sparsity as a function of lambda");
snapnow

disp(" The optimal lambda value is same in case of 5-fold CV and 10-fold CV. The number of non-zeros decreases as we increase lambda.")
%c)
disp("")
rng default
disp("2c) 5-fold CV using lassoplot")
use_built_in(X, Y, 5, lambda);

disp("10-fold CV using lassoplot")
use_built_in(X, Y, 10, lambda);

disp("The optimal lambda value might have differed if the input data wasnt randomly permuted.");
disp("Also, lassoplot ignores lamda=0, as it reduces to OLS in this case");

function perform_cross_validation(X,Y,k, lambda, fig_no)
    average_MSE = zeros(size(lambda));
    average_non_zeros = zeros(size(lambda));
    for j = 1:length(lambda)
        c = lambda(j);
        chunk_size = size(Y,1)/k;
        for i=1:k
            index = (i * chunk_size) - chunk_size;
            X_test = X(index+1:index+chunk_size, :);
            Y_test = Y(index+1:index+chunk_size, :);
            X_train = [X(1:index, :);X(index+chunk_size+1:end, :)];
            Y_train = [Y(1:index);Y(index+chunk_size+1:end)];
            [B,FitInfo]  = lasso(X_train, Y_train, "Lambda", c);
            coef0 = FitInfo.Intercept;
            average_MSE(j) = average_MSE(j) + mean((Y_test-(X_test * B + coef0)).^2);
            average_non_zeros(j) = average_non_zeros(j) + nnz(B);
        end
    end
    average_MSE = average_MSE/k;
    average_non_zeros = average_non_zeros/k;
    %subplot(2,1,1);
    %disp("Average MSE as a function of lambda = ")
    figure(fig_no);
    plot(lambda, average_MSE);
    xlabel('lambda')
    ylabel('Mean Squared Error')
    figure(fig_no+1);
    %subplot(2,1,2);
    %disp("Sparsity as a function of lambda = ")
    plot(lambda, average_non_zeros);
    xlabel('lambda')
    ylabel('Number of non zero coefficients')
    [M, min_idx] = min(average_MSE);
    lamda_optimal = lambda(min_idx)
end


function use_built_in(X,Y, k, lambda)
    [B, FitInfo] = lasso(X,Y,'Lambda', lambda, 'CV', k);
    lassoPlot(B,FitInfo,'PlotType','CV');
    snapnow
end


function lambda = generate_lambda(min, max, spacing)
    lambda = [];
    i = min;
    while i <= max
        lambda = [lambda; i];
        i = i + spacing;
    end
end
```

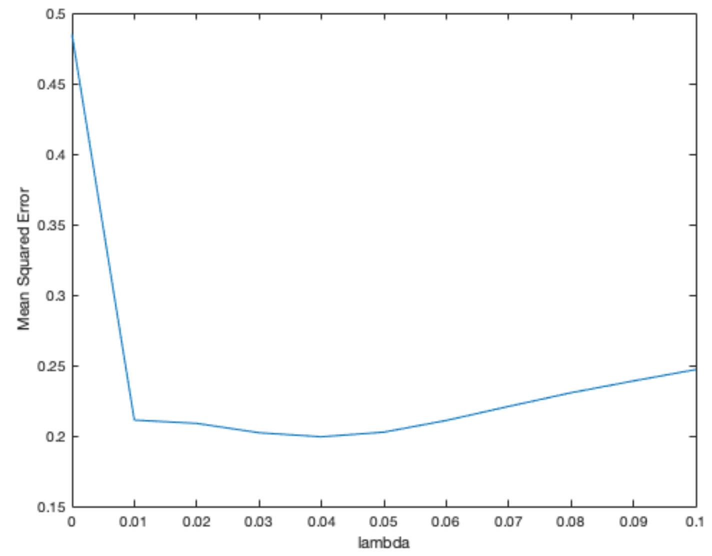
```
2a) 5-fold CV
```

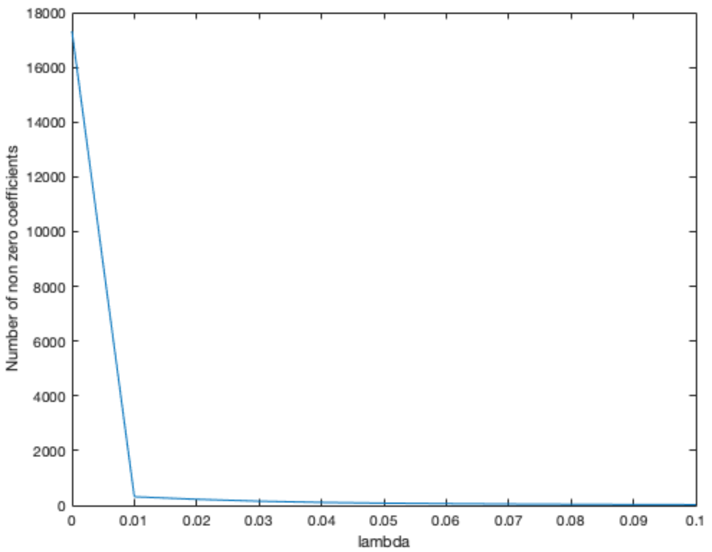
```
lamda_optimal =

    0.0400

Average MSE and Sparsity as a function of lambda
```



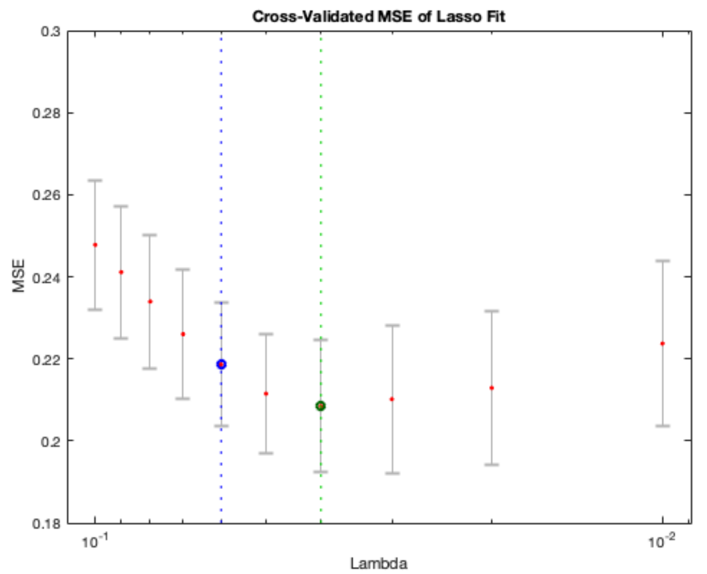



```
2b) 10-fold CV

lamda_optimal =

    0.0400

Average MSE and Sparsity as a function of lambda
```

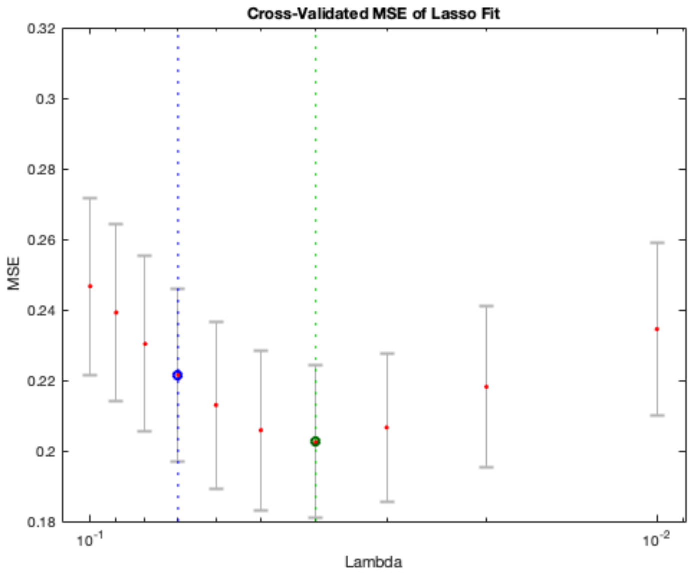





```
 The optimal lambda value is same in case of 5-fold CV and 10-fold CV. The number of non-zeros decreases as we increase lambda.
2c) 5-fold CV using lassoplot
```




```
10-fold CV using lassoplot
```




```
The optimal lambda value might have differed if the input data wasnt randomly permuted.
Also, lassoplot ignores lamda=0, as it reduces to OLS in this case
```

Published with MATLAB® R2018a

2c) Missed out on mentioning that lambda optimal obtained by lassoplot match a) and b)

Homework 2 - Question 3

```
data_1 = csvread('brca_reduced.csv');
Y_1 = data_1(:, end);
X_1 = data_1(:, 1:end-1);

%a)
disp("3a)")
fprintf("OLS fit \n")
X_1 = [ones(size(Y_1)),X_1];
B_ols_1 = regress(Y_1,X_1);
MSE_ols_1 = mean((Y_1 - X_1*B_ols_1).^2);
fprintf("MSE for clean data = %f \n",MSE_ols_1);


data_2 = csvread('brca_noisy.csv');
Y_2 = data_2(:, end);
X_2 = data_2(:, 1:end-1);
X_2 = [ones(size(Y_2)),X_2];
B_ols_2 = regress(Y_2,X_2);
MSE_ols_2 = mean((Y_2 - X_2*B_ols_2).^2);
fprintf("MSE for noisy data = %f \n", MSE_ols_2)
norm_ols = norm(B_ols_2-B_ols_1, inf);
fprintf("l-inf norm for noisy data = %f \n",norm_ols)

disp(" ")
%b)
disp("3b)")
fprintf("Robust fit using huber loss \n")
X_1 = data_1(:, 1:end-1);
B_robust_1 = robustfit(X_1,Y_1, 'huber');
MSE_robust_1 = mean((Y_1 - (X_1*B_robust_1(2:end) + B_robust_1(1))).^2);
fprintf("MSE for clean data= %f \n",MSE_robust_1)

X_2 = data_2(:, 1:end-1);
B_robust_2 = robustfit(X_2,Y_2, 'huber');
MSE_robust_2 = mean((Y_2 - (X_2*B_robust_2(2:end) + B_robust_2(1))).^2);
fprintf("MSE for noisy data = %f \n",MSE_robust_2)
norm_robost = norm(B_robust_2-B_robust_1, inf);
fprintf("l-inf norm = %f \n",norm_robost)
disp("On comparing a) and b) MSE of huber is more than OLS and huber results to more sparse coefficient matrix.");
disp(" ")
%c)
disp("3c)")
losses = ["cauchy", "talwar", "welsch"];
for i =1:length(losses)
    loss = losses(i);
    fprintf("Robust fit using %s loss \n", loss)
    B_robust_1 = robustfit(X_1,Y_1, loss);
    MSE_robust_1 = mean((Y_1 - (X_1*B_robust_1(2:end) + B_robust_1(1))).^2);
    fprintf("MSE for clean data = %f \n",MSE_robust_1);

    X_2 = data_2(:, 1:end-1);
    B_robust_2 = robustfit(X_2,Y_2, loss);
    MSE_robust_2 = mean((Y_2 - (X_2*B_robust_2(2:end) + B_robust_2(1))).^2);
    fprintf("MSE for noisy data = %f \n",MSE_robust_2)
    norm_robost = norm(B_robust_2-B_robust_1, inf);
    fprintf("l-inf norm = %f \n",norm_robost)
    disp(" ")
end


disp("We can see that in all the cases — OLS and Robust Regression, MSE calcuated for clean data is much lower than that calculated for noisy data
disp("MSE  for robust regression(in all 4 cases) is more than that of OLS for both noisy and clean data. The l-∞ norm of the difference between tl
disp("regression coefficients, is much more in case of OLS. This shows that OLS is highly effected to outliers when compared to robust regression.
disp("Specifically for Cauchy, Talwar and Welsch loss, MSE is higher than Huber and same goes with sparsity of coefficients. This could indicate "
disp("they are more robust when compared to Huber");
```

```
3a)
OLS fit
MSE for clean data = 0.159327
MSE for noisy data = 10.422337
l-inf norm for noisy data = 2.239375

3b)
Robust fit using huber loss
MSE for clean data= 0.167658
MSE for noisy data = 13.604143
l-inf norm = 0.221294
On comparing a) and b) MSE of huber is more than OLS and huber results to more sparse coefficient matrix.

3c)
Robust fit using cauchy loss
MSE for clean data = 0.170836
MSE for noisy data = 14.490056
l-inf norm = 0.176621

Robust fit using talwar loss
MSE for clean data = 0.173959
MSE for noisy data = 14.742861
l-inf norm = 0.187621

Robust fit using welsch loss
MSE for clean data = 0.182239
```

```
MSE for noisy data = 14.698762
l-inf norm = 0.192031
```

We can see that in all the cases – OLS and Robust Regression, MSE calcuated for clean data is much lower than that calculated for noisy data MSE for robust regression(in all 4 cases) is more than that of OLS for both noisy and clean data. The l-∞ norm of the difference between the regression coefficients, is much more in case of OLS. This shows that OLS is highly effected to outliers when compared to robust regression. Specifically for Cauchy, Talwar and Welsch loss, MSE is higher than Huber and same goes with sparsity of coefficients. This could indicate they are more robust when compared to Huber

Published with MATLAB® R2018a

(4)

a) In the matrix form, OLS objective is given by,

$$f(\beta) = \min_{\beta} \|y - X\beta\|_2^2$$

$$\nabla f(\beta) = -2X^T y + 2X^T X\beta$$
$$= 2X^T(X\beta - y)$$

Gradient descent formula is given by

$$\beta^t = \beta^{t-1} - \eta \nabla f(\beta^{t-1})$$
$$= \beta^{t-1} - \eta(2X^T(X\beta^{t-1} - y))$$
$$= \beta^{t-1} - 2\eta X^T(X\beta^{t-1} - y)$$

b) Incase of Newton Raphson algorithm, iterative steps, is given by

$$\beta^t = \beta^{t-1} - (\nabla^2 f(\beta^{t-1}))^{-1} \nabla f(\beta^{t-1})$$

$$\nabla^2 f(\beta) = \nabla(2X^T X\beta - 2X^T y)$$
$$= 2\nabla(X^T X\beta)$$
$$= 2X^T X$$

In general, $\nabla_\beta(A\beta) = A$

$$\beta^t = \beta^{t-1} - (2X^T X)^{-1}(2X^T X\beta^{t-1} - X^T y)$$
$$= \beta^{t-1} - \tfrac{1}{2}(X^T X)^{-1} * 2(X^T X\beta^{t-1} - X^T y)$$
$$= \beta^{t-1} - (X^T X^{-1})(X^T X)\beta^{t-1} + (X^T X)^{-1} X^T y$$

$$\beta^t = \beta^{t-1} - \beta^{t-1} + (X^TX)^{-1}X^Ty$$

we know that $A^{-1}A = 1$

$$\boxed{\beta^t = (X^TX)^{-1}X^Ty} = \beta_{OLS}$$

Observation → Optimal β is independent of initial β value. It seems like it converges to OLS.