Advanced Machine Learning
CPSC 8420

Problem 1: Ridge Regression

To prove:

$$(A^TA + \lambda I_p)^{-1} A^T = A^T(AA^T + \lambda I_n)^{-1}$$

Let, L.H.S. = K

R.H.S = P

$$(A^TA + \lambda I_p)^{-1} A^T = K$$

Multiplying by $(A^TA + \lambda I_p)$

$$(A^TA + \lambda I_p)(A^TA + \lambda I_p)^{-1} A^T = (A^TA + \lambda I_p) K$$

Multiplying both sides by $(A^TA^T + \lambda I_n)$

$$A^T(AA^T + \lambda I_n) = (A^TA + \lambda I_p)(K)(AA^T + \lambda I_n)$$

$$A^TAA^T + A^T\lambda I_n = (A^TA + \lambda I_p)(K)(AA^T + \lambda I_n) \quad -(1)$$

Now, for R.H.S:

$$A^T(AA^T + \lambda I_n)^{-1} = P$$

similarly, we can say that:

$$(A^TA + \lambda I_p)A^T(AA^T + \lambda I_n)^{-1}(AA^T + \lambda I_n) =$$

$$(A^TA + \lambda I_p)P(AA^T + \lambda I_n)$$

$$\Rightarrow (A^TAA^T + \lambda I_p)A^T = (A^TA + \lambda I_p)P(AA^T + \lambda I_n)$$

$$A^TAA^T + \lambda A^T I_n = (A^TA + \lambda I_p)(P)(AA^T + \lambda I_n) \quad -(2)$$

Subtracting (2) from (1), we get:

$$(A^TA + \lambda I_p)(K)(AA^T + \lambda I_n) = (A^TA + \lambda I_p)(P)(AA^T + \lambda I_n)$$

multiplying by $(A^TA + \lambda I_p)^{-1}$ on both sides, we get:

$$K(AA^T + \lambda I_n) = P(AA^T + \lambda I_n)$$

Similarly, multiplying by $(AA^T + \lambda I_n)^{-1}$ we get

$$K = P.$$

$$\therefore [LHS = RHS]$$

```
% Define the size of the random matrix (e.g., a 4x4 matrix)
n = 100;
p = [10, 100, 1000, 2000];
l = 0.1;
I_n = eye(n);
t_p = [];
t_n = [];

% Generate a random matrix of integers between 1 and 10
for i = 1:length(p)
    I_p = eye(p(i));
    A = randi([1, 100], n, p(i));
    tic;
    B = inv(A'*A + l .* I_p) * A';
    t_p(i) = toc;
    tic;
    C = A'*inv(A*A' + l * I_n);
    t_n(i) = toc;
end

t_p = 1000 * t_p;
t_n = 1000 * t_n;
disp("t_p: ")
```

```
t_p:
```

```
disp(t_p);
```

```
    0.8866    0.4024   34.2900  158.3368
```

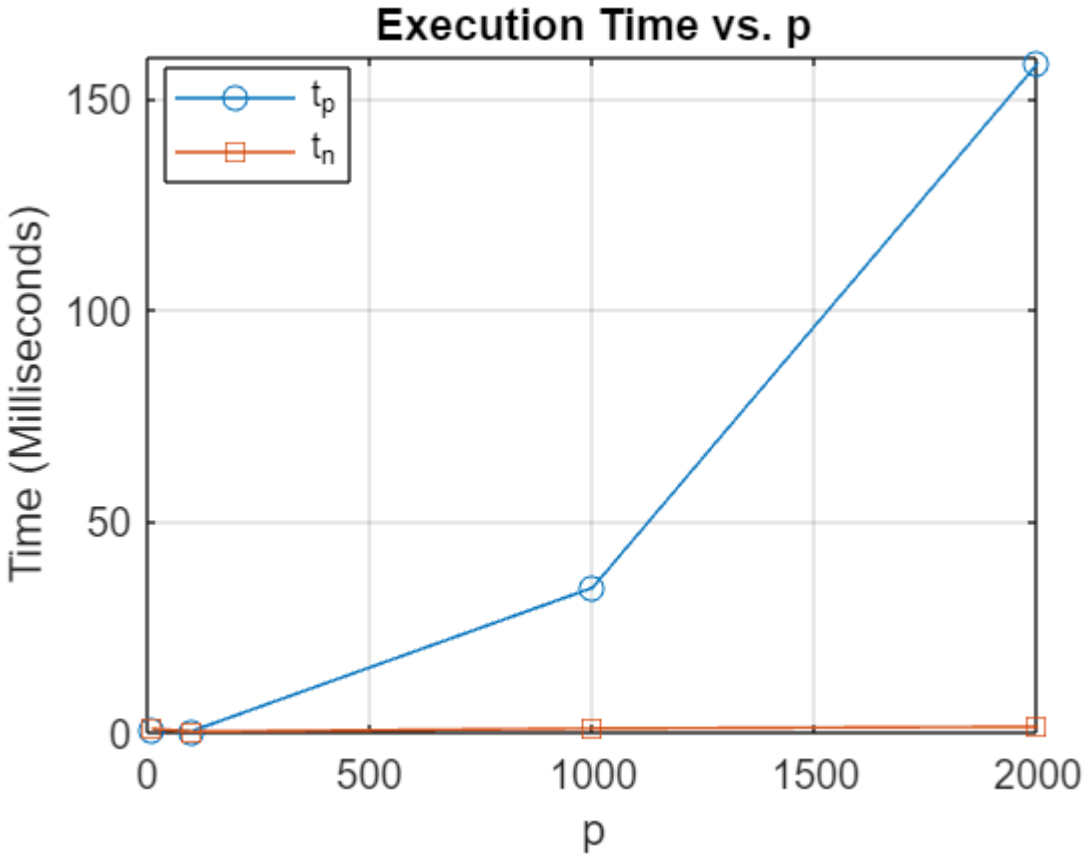
```
disp("t_n: ")
```

```
t_n:
```

```
disp(t_n);
```

```
    0.9951    0.3015    1.0300    1.4470
```

```
% Create a plot
figure;
plot(p, t_p, '-o', 'DisplayName', 't_p');
hold on;
plot(p, t_n, '-s', 'DisplayName', 't_n');
xlabel('p');
ylabel('Time (Milliseconds)');
title('Execution Time vs. p');
legend('Location', 'Best');
grid on;
hold off;
```



We can see that the execution time tn is co9nsiderably less than tp. So, this form should be preferred for solving a regressor with large number of samples.

Ans 2.) To solve:

$$\min_X \|AX + XC + Y\|_F^2 \quad \text{---} \quad (1)$$

given: $vec(AUC) = (C^T \otimes A)\, vec(U)$ — (2)

Using: $\min_X \|vec(AX) + vec(XC) + vec(Y)\|_F^2$

using eqn (2):

$$vec(AX) = vec(AXI) \quad \text{---} \quad (3)$$
$$= (I^T \otimes A)\, vec(X)$$

$$vec(XC) = vec(IXC)$$
$$= (C^T \otimes I)\, vec(X) \quad \text{---} \quad (4)$$

from 3, 4 we have:

$$\min_X \|(I^T \otimes A)\, vec(X) + (C^T \otimes I)\, vec(X) - Y\|_F$$

$$\min_X \|[(I^T \otimes A) + (C^T \otimes I)]\, vec(X) - Y\|_F^2$$

As, $(I^T \otimes A) + (C^T \otimes I) = M$

we have: $\min \|M.\, vec(X) - Y\|_F^2$

using least squares approach

$$\nabla f(w_{ec}(x)) = 2M^T(M.w_{ec}(x) - Y)$$

Putting $\nabla f(w_{ec}(x)) = 0$

$$2M^T(M.w_{ec}(x) - Y) = 0$$

$$M^T M w_{ec}(x) = M^T.Y$$

$$\boxed{w_{ec}(x) = (M^T M)^{-1} M^T Y.} \quad *$$

```
% Define the size of our example matrices
n = 2;

% Generate a random nxn matrix for A
A = randn(n, n);
disp('A: ')
```

A:

```
disp(A);
```

```
   -1.0891    0.5525
    0.0326    1.1006
```

```
% Generate a random nxn matrix for X_star (optimal X)
X_star = randn(n, n);
disp('X_Star: ')
```

X_Star:

```
disp(X_star);
```

```
    1.5442   -1.4916
    0.0859   -0.7423
```

```
% Generate a random nxn matrix for C
C = randn(n, n);
disp('C: ')
```

C:

```
disp(C);
```

```
   -1.0616   -0.6156
    2.3505    0.7481
```

```
% Compute the matrix Y based on the relation: Y = AX* + X*C
Y = A * X_star + X_star * C;

% Create an identity matrix of size nxn
I = eye(n);

% Construct matrix M using the Kronecker product
% This is used for reshaping the matrix multiplication into vector form
M = kron(I, A) + kron(C.', I);

% Use matrix division (or the inverse of M) to compute vecX from Y
vecX = M \ Y(:);

% Reshape vecX back into matrix form to get X
X = reshape(vecX, [n, n]);
```

```
% Display the computed X and M matrices
disp('X: ')
```

```
X:
```

```
disp(X);
```

```
    1.5442   -1.4916
    0.0859   -0.7423
```

```
disp('M: ')
```

```
M:
```

```
disp(M);
```

```
   -2.1506    0.5525    2.3505         0
    0.0326    0.0390         0    2.3505
   -0.6156         0   -0.3410    0.5525
         0   -0.6156    0.0326    1.8487
```

vec(X) and vec(X*) are equivalent.

Problem 3

Vanilla Linear Regression:

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

using least squares approach to minimize $f(\beta)$:

$$\nabla f(\beta) = 0$$

$$\nabla f(\beta) = -2A^T(y - A\hat{\beta}_{LS})$$

$$-2A^T(y - A\hat{\beta}_{LS}) = 0$$

$$A^T \cdot A\hat{\beta}_{LS} = A^T \cdot y$$

$$\hat{\beta}_{LS} = (A^T A)^{-1} A^T y$$

given $A^TA = I$, so $\boxed{\beta_{ls} = IA^Ty}$

Ridge Linear Regression:

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

using least squares approach to minimize $f(\beta)$

$$\nabla f(\beta) = 0$$

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

$$A^T(A\beta - y) + \lambda\beta = 0$$

$$A^TA\beta - A^Ty + \lambda\beta = 0$$

$$(A^TA + \lambda I)\beta = A^Ty$$

Given, $A^TA = I$

$$\hat{\beta}_{Ridge} = ((1+\lambda)I)^{-1} A^Ty$$

$$\hat{\beta}_{Ridge} = \frac{1}{1+\lambda} \cdot I A^Ty$$

6. Lasso Linear Regression

$$f(\beta) = \min_{\beta} \|y - A\beta\|_2^2 + \lambda \|\beta\|_1$$

Given $A^T A = I$ (i.e. $A \rightarrow$ orthogonal)

For lasso problem, $\hat{\beta}_{lasso}$ satisfies the subgradient optimality condition,

$$\nabla f(\beta) + \lambda \, sgn(\hat{\beta}_{lasso}) = 0 \quad \text{--(1)}$$

$$f_L(\beta) = \min_{\beta} \frac{1}{2} \|A\beta - y\|_2^2$$

Using optimality conditions for lasso cond^n

$$\nabla f_L(\beta) = A'(A\beta - y)$$
$$= A'A\beta - A'y$$

given $A'A \rightarrow I$

$$\nabla f_L(\beta) = \beta - A'y$$

from eq^n (1) $\rightarrow$

$$\beta - A'y + \lambda \, sgn(\beta) = 0$$

$$\hat{\beta}_{lasso} = A'y - \lambda \, sgn(\hat{\beta}_{lasso})$$

Now the soft thresholding operator $S_\lambda(z)$ is defined as:

$$S_\lambda(z) = sign(z) \max(|z| - \lambda, 0)$$

Given optimality condition each component $\hat{\beta}_{lasso}(i)$ of vector $\hat{\beta}_{lasso}$ can be found by applying the soft thresholding operator to vector $A^T y$ by.

$$\hat{\beta}_{lasso}(i) = S_\lambda((A^T y)_i)$$

Subset selection Method:

$$f(\beta) = \min_{\beta} \frac{1}{2} \|y - A\beta\|_2^2 + \lambda \|\beta\|_0.$$

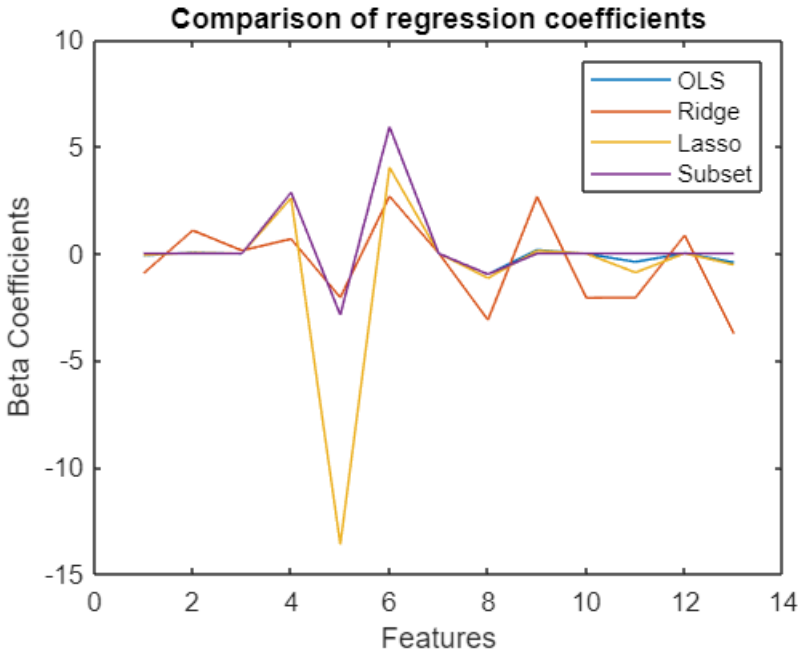
Given $A^T A = I$., ($A \rightarrow$ orthogonal)
This means that correlation b/w any two columns of A is zero.

∴ correlation b/w y & j^{th} column of A is just $A_j^T y$.

So, for each predictor j:
compute $c_j = A_j^T y$

$$\beta_j^{subset} = \begin{cases} \beta_j = 0 & \text{if } |c_j| < \lambda \\ \beta_j = c_j & \text{otherwise} \end{cases}$$

```
data = load(['C:\Users\param\OneDrive - Clemson University' ...
    '\Desktop\CU\Fall23\Advance Machine Learning\HW1/housing.data']);

x = data(:, 1:13);
y = data(:,14);

% Vanilla Linear Regression
beta_ols = pinv(x' * x) * x' * y;

lambda = 0.1;
% Ridge regression
beta_ridge = ridge(y, x, lambda);

% Lasso regression
[beta_lasso, fitinfo] = lasso(x, y, 'Lambda', lambda, 'intercept', true);

% Subset (Simple implementation using a thresholding on OLS coefficients)
lambda_subset = 0.5;
beta_subset = beta_ols;
beta_subset(abs(beta_ols) < lambda_subset) = 0;

% Plot
figure;
plot(1:13, beta_ols, 'DisplayName', 'OLS');
hold on;
plot(1:13, beta_ridge, 'DisplayName', 'Ridge');
plot(1:13, beta_lasso, 'DisplayName', 'Lasso');
plot(1:13, beta_subset, 'DisplayName', 'Subset');
title('Comparison of regression coefficients');
xlabel('Features');
ylabel('Beta Coefficients');
legend();
hold off;
```


Comparison of regression coefficients
Beta Coefficients
Features
OLS
Ridge
Lasso
Subset
10
5
0
-5
-10
-15
0
2
4
6
8
10
12
14

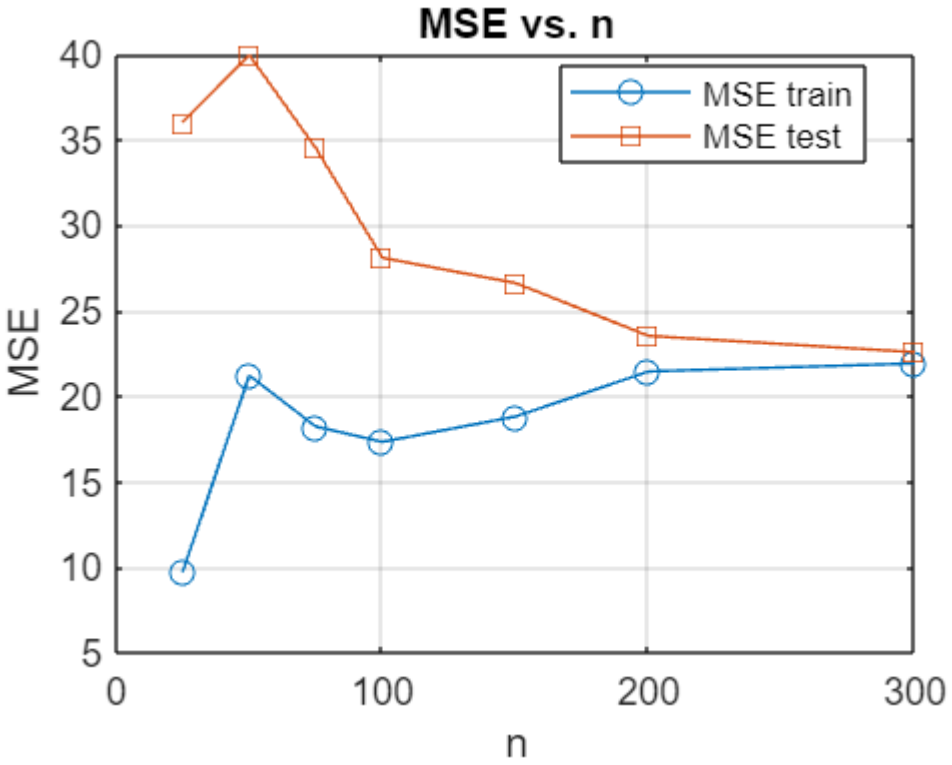
Problem 4: Linear Regression and it's regression

Preparing train and test sets for models

```
data = load('housing.data');
x = data(:, 1:13);
y = data(:, 14);
[n, d] = size(x);

seed = 2; rand('state', seed); randn('state', seed);
perm = randperm(n);  % remove any possible ordering fx
x = x(perm, :); y = y(perm);
Ntrain = 300;
Xtrain = x(1:Ntrain, :); ytrain = y(1:Ntrain);
Xtest = x(Ntrain + 1:end, :); ytest = y(Ntrain + 1:end);
```

Part 2

```
n = [25 50 75 100 150 200 300];

MSE_train = zeros(size(n));
MSE_test = zeros(size(n));

for i = 1:length(n)
    % Normalizing the train data
    [st_array1, m1 , s1] = zscore(Xtrain(1:n(i), :));
    X_train = [ones(size(Xtrain(1:n(i), :), 1), 1) st_array1];
    % Normalizing the test data
    s_test = (Xtest-m1)./s1;
    X_test = [ones(size(Xtest, 1), 1) s_test];
    y_train = ytrain(1:n(i));
    weights = pinv(X_train'*X_train)*(X_train' * y_train);
    MSE_train(i) = ((X_train * weights - y_train)' * (X_train * weights - y_train))/n(i);
    MSE_test(i) = ((X_test * weights - ytest)'*(X_test * weights - ytest))/size(X_test, 1);
end

% Create a plot
figure;
plot(n, MSE_train, '-o', 'DisplayName', 'MSE train');
hold on;
plot(n, MSE_test, '-s', 'DisplayName', 'MSE test');
xlabel('n');
ylabel('MSE');
title('MSE vs. n');
legend('Location', 'Best');
grid on;
hold off;
```



Expanding the size of the training dataset equips the model with a richer source of information, typically resulting in superior adaptability to unseen data, consequently reducing test error.

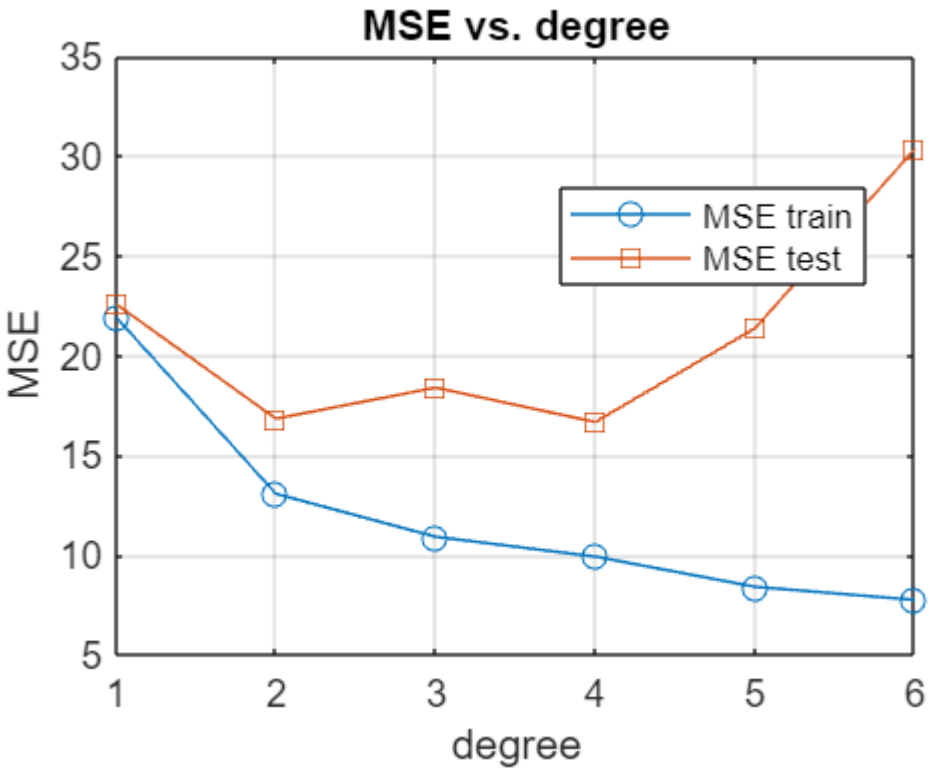
In scenarios where the training dataset is limited in size, the model can excessively tailor its predictions to the confined data, occasionally veering into overfitting. With the enlargement of the training dataset, the information to be absorbed becomes more multifaceted and intricate to accommodate flawlessly, leading to an upturn in training error.

Furthermore, as the training dataset's dimensions augment and edge nearer to the complete dataset, the model undergoes exposure to a more comprehensive portion of the available data during training. As a result, its performance on both the training set and the test set starts to draw closer, culminating in the convergence of the two error curves.

Part 3

```
deg = [1 2 3 4 5 6];

MSE_train = zeros(size(deg));
MSE_test = zeros(size(deg));

for i = 1:length(deg)
    % Train Feature expansion
    X_train = degexpand(Xtrain, deg(i), 1);
    [X_train(:, 2:end), mean, std] = zscore(X_train(:, 2:end));
    y_train = ytrain;
    y_test = ytest;
    % Test feature expansion
```

```
    X_test = degexpand(Xtest, deg(i), 1);
    X_test(:, 2:end) = (X_test(:, 2:end)-mean)./std;
    weights = pinv(X_train'*X_train) * X_train' * y_train;
    MSE_train(i) = ((X_train * weights - y_train)' * (X_train * weights - y_train))/size(X_trai
    MSE_test(i) = ((X_test * weights - y_test)'*(X_test * weights - y_test))/size(X_test, 1);
end

% Create a plot
figure;
plot(deg, MSE_train, '-o', 'DisplayName', 'MSE train');
hold on;
plot(deg, MSE_test, '-s', 'DisplayName', 'MSE test');
xlabel('degree');
ylabel('MSE');
title('MSE vs. degree');
legend('Location', 'Best');
grid on;
hold off;
```

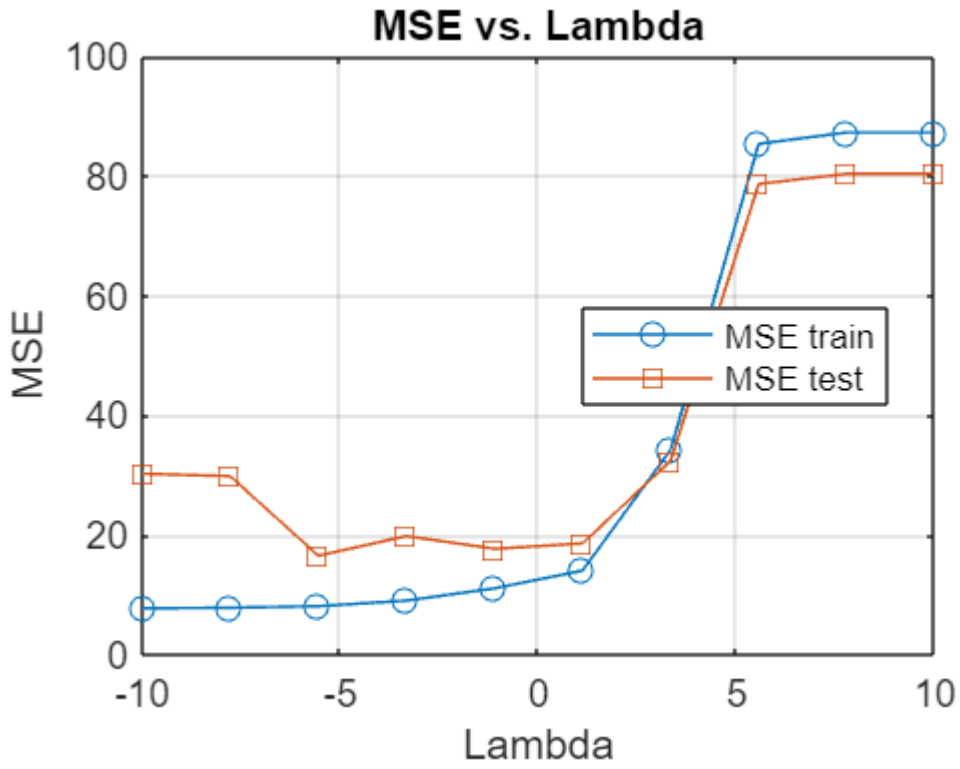


As the polynomial degree is raised, the model gains greater adaptability, allowing it to conform more closely to the training data, subsequently decreasing the training error.

In the early stages, the increased adaptability of the model aids in capturing the fundamental data patterns, which, in turn, results in a decrease in the test error. However, there comes a point where the model begins to overfit. In essence, it starts fitting the training data with excessive precision, encompassing not only the genuine patterns but also the noise and outliers. This diminishes its capacity to generalize effectively to new data, ultimately leading to an uptick in test error.

Part 4

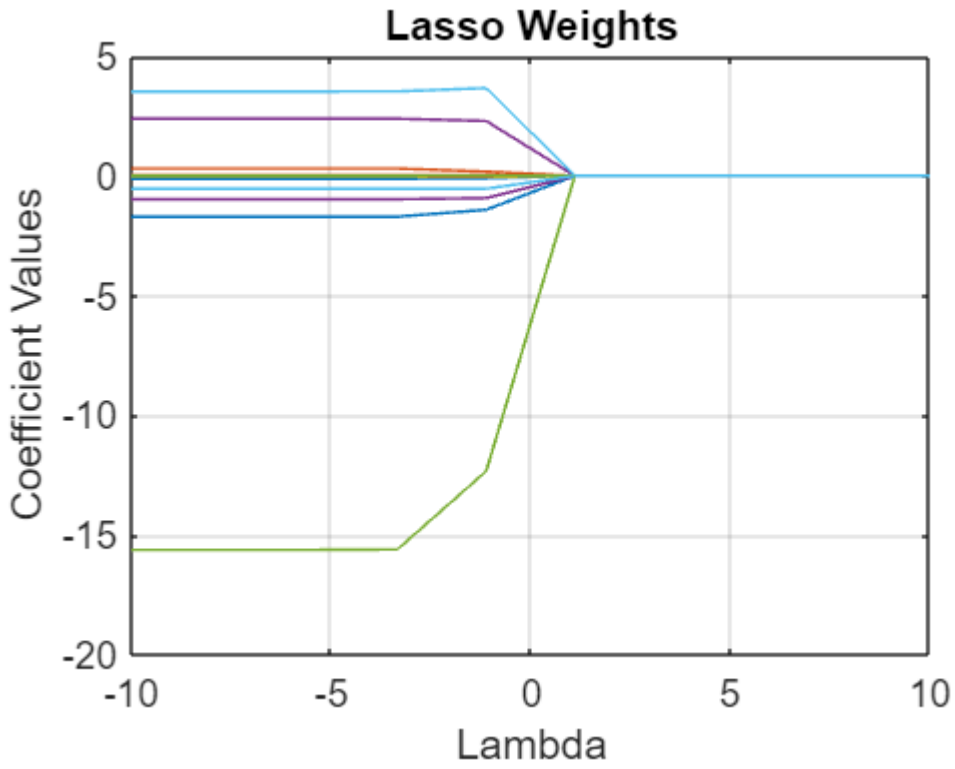
```
lambdas = [0 logspace(-10, 10, 10)];

X_train = degexpand(Xtrain, 6, 1);
[X_train(:, 2:end), mean, std] = zscore(X_train(:, 2:end));
y_train = ytrain;
y_test = ytest;
X_test = degexpand(Xtest, 6, 1);
X_test(:, 2:end) = (X_test(:, 2:end)-mean)./std;
I = eye(size(X_train, 2));
% To avoid regularizing the bias term, put I(0,0) = 0
I = [zeros(size(I, 1), 1), I(:,2:end)];

MSE_train = zeros(size(lambdas));
MSE_test = zeros(size(lambdas));

for i = 1:length(lambdas)
    weights = pinv(X_train'*X_train + lambdas(i)* I) * X_train' * y_train;
    MSE_train(i) = (((X_train * weights - y_train)' * (X_train * weights - y_train)))/size(X_tr
    MSE_test(i) = ((((X_test * weights - ytest)'*(X_test * weights - ytest))))/size(X_test, 1);
end

% Create a plot
figure;
plot(log10(lambdas), MSE_train, '-o', 'DisplayName', 'MSE train');
hold on;
plot(log10(lambdas), MSE_test, '-s', 'DisplayName', 'MSE test');
xlabel('Lambda');
ylabel('MSE');
title('MSE vs. Lambda');
legend('Location', 'Best');
grid on;
hold off;
```



Initially, regularization helps in avoiding overfitting. But as seen from the plot, as the regularization increases, the model starts to underfit which leads to an increased mean squared error.

Part 5

```
lambdas = logspace(-10, 10, 10);

[weights, FitInfo] = lasso(Xtrain, ytrain, 'Lambda', lambdas);

figure;
plot(log10(lambdas), weights);
xlabel('Lambda');
ylabel('Coefficient Values');
title('Lasso Weights');
grid on;
```



We observe that as the regularization increases, the weights are pushed towards zero.