Khirod Sahoo HW3 DATA 558

1.

A.

```
f = function(x) {
  return(0)
}
set.seed(100)
e <- rnorm(100)
set.seed(101)
x <- matrix(rnorm(100*10000),ncol=10000)
y <- f(x) + e
df <- data.frame(x,y)
```

Y can be expressed as a function of X:

Y = f(X) + ε, where ε ~ N(0,1)

Since, Y is generated randomly and doesn't depend on the values of X, its values only depend on the random error term and hence the above equation can be reduced to

Y= ε, where f(X) =0

B. The value of irreducible error is the variance of the error term which is 1(as given in the question)

C. The estimated function given here is $\hat{f}$(X). So, y can be expressed as

$\hat{Y}$ = $\hat{f}$(X) and $\hat{f}$(X)=0, for all the values of X

I. The bias of this procedure can be calculated from the formula E(E($\hat{f}$(X)) - f(X))

E($\hat{f}$(X)) =0 , since $\hat{f}$(X)=0, for all the values of X

f(X) =0 , due to the nature of the data generation

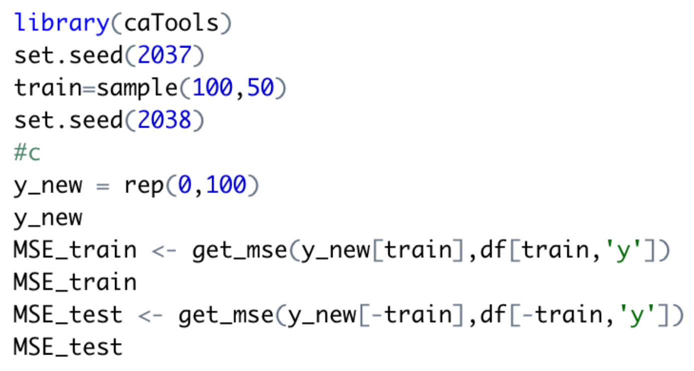
By fitting values, bias = 0

II. The variance of this procedure is also 0, since $\hat{f}$(X) is just a constant value.

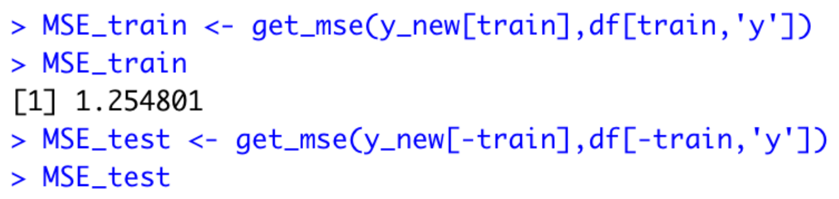
III. The expected prediction error of this procedure is simply the variance of the error since bias square and variance terms are zero. Hence, it is 1.

IV. Using the validation set approach, it was found that the estimated test error is lower than the training error and is equal to 0.808.

```
get_bias = function(estimate, truth) {
  mean((mean(estimate) - truth))
}

get_mse = function(estimate, truth) {
  mean(mean((estimate - truth) ^ 2))
}
```



```
library(caTools)
set.seed(2037)
train=sample(100,50)
set.seed(2038)
#c
y_new = rep(0,100)
y_new
MSE_train <- get_mse(y_new[train],df[train,'y'])
MSE_train
MSE_test <- get_mse(y_new[-train],df[-train,'y'])
MSE_test
```



```
> MSE_train <- get_mse(y_new[train],df[train,'y'])
> MSE_train
[1] 1.254801
> MSE_test <- get_mse(y_new[-train],df[-train,'y'])
> MSE_test
```

V. The expected prediction error was 1 whereas the estimated test error was 0.808. They are not exactly the same but are quite close. Since, esimated f(x) will always generate 0 for all value of x and true y is just a random distribution which is equal to the random error term. Hence , all the variance is induced by the error term which makes the expected prediction error as variance of error which is 1.
In the validation approach as well, the estimated f(x) will yield 0 for all values of x in both training and test set which implies it doesn't depend on the distribution of x in both training and test. And true y in both training and test is equal to the random error term. Hence, ideally the estimated test error should also be equal to variance of the error term i.e 1 but since there is some variation induced by the random split, it will be close to 1 but not exactly 1.

D. A least-square model was fit using all the 10000 predictors. The estimated test error using the validation test approach was 53.05.
Please find below the code for the same:

```
> model <- lm(y ~ .,data=df,subset=train)
> predict_train <- predict(model,df)[train]
Warning message:
In predict.lm(model, df) :
  prediction from a rank-deficient fit may be misleading
> predict_test <- predict(model,df)[-train]
Warning message:
In predict.lm(model, df) :
  prediction from a rank-deficient fit may be misleading
> MSE_train <- get_mse(predict_train,df[train,'y'])
>
> MSE_train
[1] 1.304363e-29
> MSE_test <- get_mse(predict_test,df[-train,'y'])
> MSE_test
[1] 53.05427
> bias_sq <- get_bias(predict_test,0)^2
> bias_sq
[1] 0.04057317
> variance <- var(predict_test)
> variance
[1] 51.76562
> err_var <- var(e)
> err_var
[1] 1.04185
```

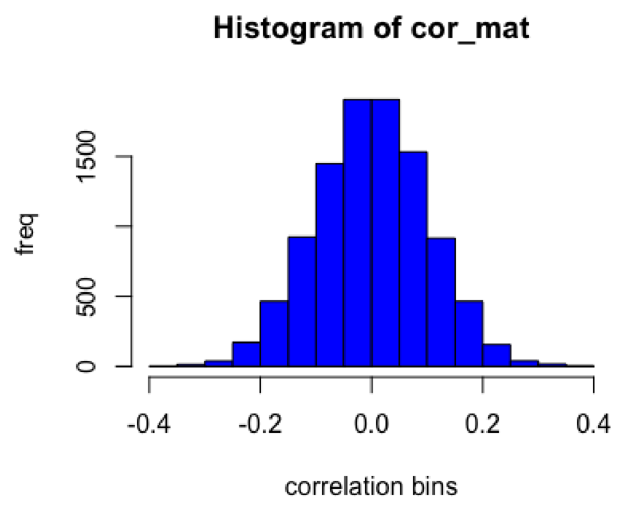
E. The procedure used in C has a smaller estimated test error, smaller bias(but the bias can be considered equal to D as bias in D is almost equal to 0), and smaller variance as well. This is because, in procedure C, the data is generated randomly and Y is independent of X. Given, the fact f(X) = 0, the estimated f(X) is also 0 as provided in the question leading to bias 0. Since, in approach C, estimated f(X) is constant, the variance is also 0 and hence the total estimated prediction error reduces to the variance of the error term which is the irreducible error. This leads to the MSE in C becoming the lowest.

2.

A.

```
set.seed(100)
y <- rnorm(100)

set.seed(101)
x <- matrix(rnorm(100*10000),ncol=10000)
df <- data.frame(x,y)

#a
cor_mat <- cor(df[,1:10000],df[,c('y')])
abs_corr <- abs(cor_mat)
cor_mat_sort <- sort(abs_corr,decreasing=TRUE,index.return=TRUE)
hist(cor_mat,breaks=20,col='blue',xlab='correlation bins',ylab='freq',title='
top_10_corr <- cor_mat_sort$x[1:10]
top_10_idx <- cor_mat_sort$ix[1:10]
```

The histogram of the correlations is shown below:



The top 10 absolute correlations are:

```
> top_10_corr
 [1] 0.3890718 0.3636382 0.3623648 0.3529197 0.3481053 0.3459373 0.3335661 0.3298934 0.3283445 0.3279711
```

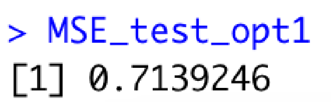
The variables with the top 10 absolute correlations are:

```
> names(df)[top_10_idx]
 [1] "X1590" "X5542" "X2146" "X2156" "X4241" "X6180" "X270"  "X7913" "X2430" "X6374"
```

B. Option 1 is carried out by first selecting the top 10 features from the entire dataset and splitting between train and test to use the validation approach.

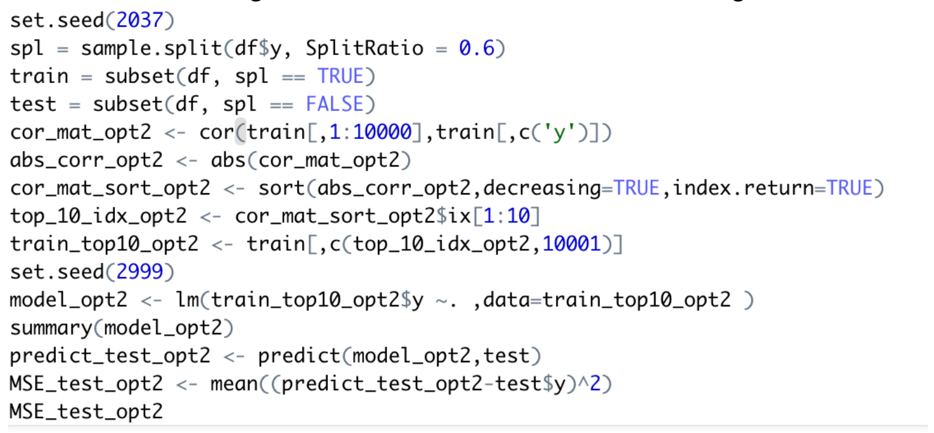
```
df_top10 <- df[,c(top_10_idx,10001)]
set.seed(2037)
spl = sample.split(df_top10$y, SplitRatio = 0.6)
train = subset(df_top10, spl == TRUE)
test = subset(df_top10, spl == FALSE)
set.seed(2999)
model_opt1 <- lm(train$y ~. ,data=train )
summary(model_opt1)
predict_test_opt1 <- predict(model_opt1,test)
MSE_test_opt1 <- mean((predict_test_opt1-test$y)^2)
MSE_test_opt1
```

The estimated test MSE is:



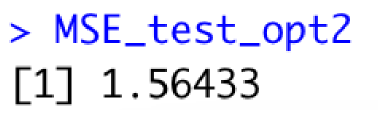
```
> MSE_test_opt1
[1] 0.7139246
```

C. Option 2 is carried out by first splitting between train and test, then finding the top 10 features from the train and calculating the estimated test MSE after training the model.



```
set.seed(2037)
spl = sample.split(df$y, SplitRatio = 0.6)
train = subset(df, spl == TRUE)
test = subset(df, spl == FALSE)
cor_mat_opt2 <- cor(train[,1:10000],train[,c('y')])
abs_corr_opt2 <- abs(cor_mat_opt2)
cor_mat_sort_opt2 <- sort(abs_corr_opt2,decreasing=TRUE,index.return=TRUE)
top_10_idx_opt2 <- cor_mat_sort_opt2$ix[1:10]
train_top10_opt2 <- train[,c(top_10_idx_opt2,10001)]
set.seed(2999)
model_opt2 <- lm(train_top10_opt2$y ~. ,data=train_top10_opt2 )
summary(model_opt2)
predict_test_opt2 <- predict(model_opt2,test)
MSE_test_opt2 <- mean((predict_test_opt2-test$y)^2)
MSE_test_opt2
```

The estimated test MSE is:

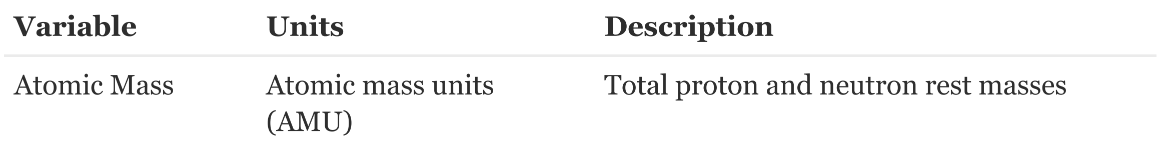


```
> MSE_test_opt2
[1] 1.56433
```

D. Option1 and Option2 give different results. The estimated test MSE in option 2 is higher than option1 because, in option1, we are using the entire dataset to check the highest correlated features. In this approach, we are using the information from the future test sample as well and hence the estimated test error will be lower in this approach.
The problem with the approach in option1 is that we are exposing our model to the test data which should not be touched while training. Due to this, we get misleading results. In the real world, when we actually test out a model on a new untouched data and we don't know anything about it, it will give worse results than what we saw in the option1 approach. Hence, option1 yields a misleading estimate of the error.

3.

A. It's a superconductivity dataset downloaded from the UCI machine learning website. This dataset contains information on around 21K superconductors. There is a total of 81 features with all being numeric. The response variable here is the critical temperature of the superconductor. There are no missing values in the data.
The primary features used in the dataset with their descriptions are



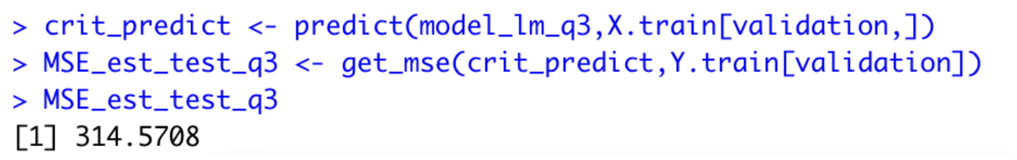
Variable	Units	Description
Atomic Mass	Atomic mass units (AMU)	Total proton and neutron rest masses

First Ionization Energy	Kilo-Joules per mole (kJ/mol)	Energy required to remove a valence electron
Atomic Radius	Picometer (pm)	Calculated atomic radius
Density	Kilograms per meters cubed (kg/m3)	Density at standard temperature and pressure
Electron Affinity	Kilo-Joules per mole (kJ/mol)	Energy required to add an electron to a neutral atom
Fusion Heat	Kilo-Joules per mole (kJ/mol)	Energy to change from solid to liquid without temperature change
Thermal Conductivity	Watts per meter-Kelvin (W/(m K))	Thermal conductivity coefficient κ
Valence	No units	Typical number of chemical bonds formed by the element

The other features in the dataset are based on these primary features.

B.

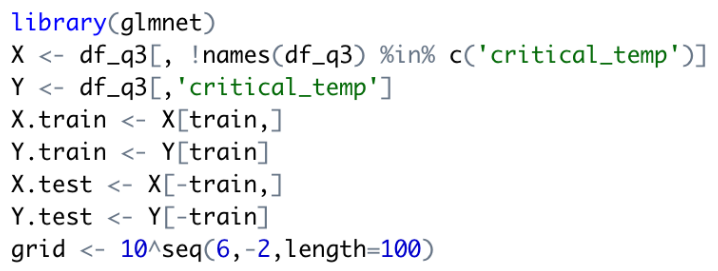
```r
set.seed(2037)
train=sample(1:nrow(df_q3),2*nrow(df_q3)/3)
X <- df_q3[, !names(df_q3) %in% c('critical_temp')]
Y <- df_q3[,'critical_temp']
X.train <- X[train,]
Y.train <- Y[train]
X.test <- X[-train,]
Y.test <- Y[-train]
set.seed(2039)
validation=sample(1:nrow(X.train),nrow(X.train)/2)
model_lm_q3 <- lm(critical_temp ~ .,data=df_q3[-validation,])
#calculate MSE on the validation set
crit_predict <- predict(model_lm_q3,X.train[validation,])
MSE_est_test_q3 <- get_mse(crit_predict,Y.train[validation])
MSE_est_test_q3

#calculate MSE on the test set
crit_predict_test <- predict(model_lm_q3,X.test)
MSE_test_q3 <- get_mse(crit_predict_test,Y.test)
MSE_test_q3
```



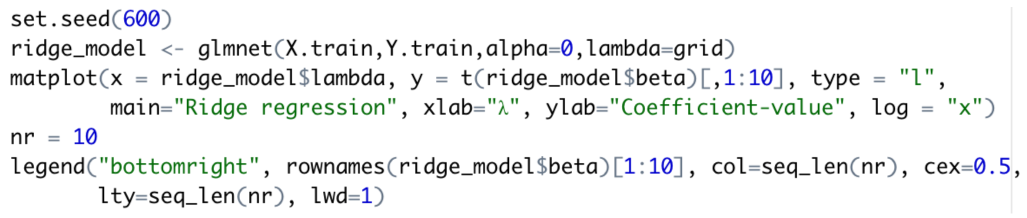
```
> crit_predict <- predict(model_lm_q3,X.train[validation,])
> MSE_est_test_q3 <- get_mse(crit_predict,Y.train[validation])
> MSE_est_test_q3
[1] 314.5708
```

The estimated test error was 314.57 and the actual test error is 302.21 and the test error was estimated using the validation set approach.
The data with around 21K rows was split between train and test. ‘Sample’ function was used to get random samples for train and test. 2/3rd of the samples were used for training and the remaining for the test. Further, the training set was split between training and validation. The model was trained using the training data and then the error was calculated on the predictions from the validation data. This resulting MSE is the estimated test error of the model.

C.

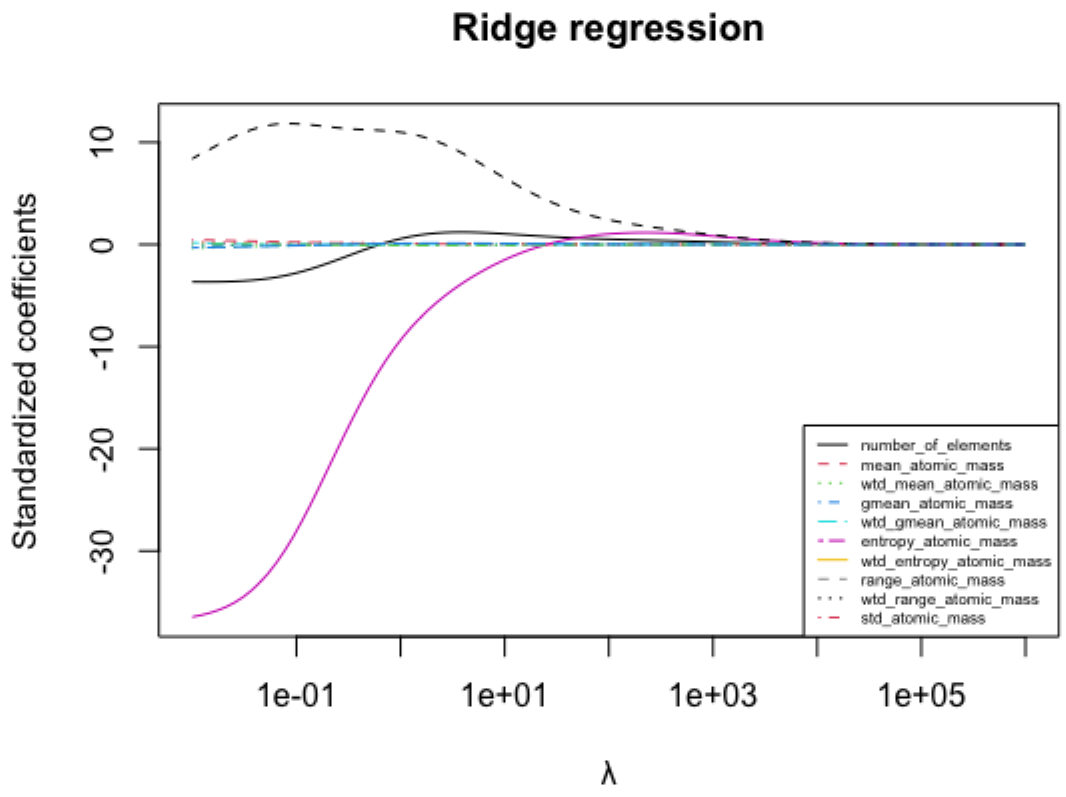


```
library(glmnet)
X <- df_q3[, !names(df_q3) %in% c('critical_temp')]
Y <- df_q3[,'critical_temp']
X.train <- X[train,]
Y.train <- Y[train]
X.test <- X[-train,]
Y.test <- Y[-train]
grid <- 10^seq(6,-2,length=100)
```



```
set.seed(600)
ridge_model <- glmnet(X.train,Y.train,alpha=0,lambda=grid)
matplot(x = ridge_model$lambda, y = t(ridge_model$beta)[,1:10], type = "l",
        main="Ridge regression", xlab="λ", ylab="Coefficient-value", log = "x")
nr = 10
legend("bottomright", rownames(ridge_model$beta)[1:10], col=seq_len(nr), cex=0.5,
       lty=seq_len(nr), lwd=1)
```

The ridge regression model was built on the train split from B for a range of values for lambda in the ‘grid’ variable.
The plot was built using the ‘matplot’ function. Please find the plot below:



There are a total of 81 features in the data but for the purpose of simplicity, only 10 features were displayed in the plot. For higher values of lambda, we can see that the coefficients approach zero as expected from a ridge regression model.

D.

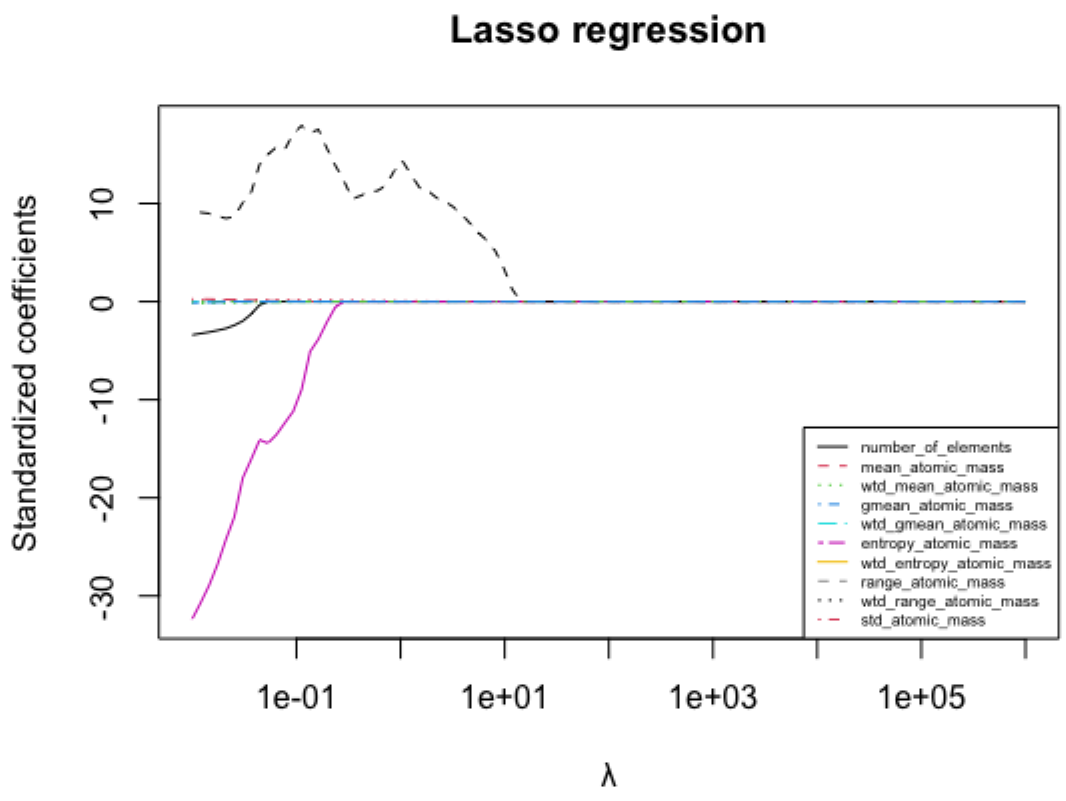
```
> grid <- 10^seq(6,-2,length=100)
> set.seed(5400)
> cv.out.ridge <- cv.glmnet(as.matrix(X.train),as.matrix(Y.train),alpha=0,lambda=grid,nfolds=5,
+                            type.measure='mse')
> best_lambda_ridge <- cv.out.ridge$lambda.min
> best_lambda_ridge
[1] 0.01
> cv.out.ridge

Call:  cv.glmnet(x = as.matrix(X.train), y = as.matrix(Y.train), lambda = grid,      type.measure
 "mse", nfolds = 5, alpha = 0)

Measure: Mean-Squared Error

     Lambda Index Measure    SE Nonzero
min 0.01000   100   318.3 3.640      81
1se 0.05337    91   321.3 3.871      81
>
> ridge.predict_bestlam <- predict(ridge_model,s=best_lambda_ridge,newx=as.matrix(X.test))
> MSE_ridge_bestlam <- get_mse(ridge.predict_bestlam,as.matrix(Y.test))
> MSE_ridge_bestlam
[1] 306.4509
```

The lambda = 0.01 provides the best result in this case i.e the minimum estimated test error.
The estimated test error with this lambda is 318.3 whereas the actual test error was 306.45.
The k-fold cross-validation approach was used to estimate the error. The train split from B is used to build models and perform k-fold cross-validation. For every value of lambda, cv error in each fold is calculated and averaged. This is achieved using the function cv.glmnet(). Finally the lambda for which the mean cv error is the minimum is observed using lambda.min and the corresponding cv error is the estimated test error for that lamda value which is 318.3 in this case for lambda=0.01.

E.

```
set.seed(600)
lasso_model <- glmnet(X.train,Y.train,alpha=1,lambda=grid)
matplot(x = lasso_model$lambda, y = t(lasso_model$beta)[,1:10], type = "l",
        main="Lasso regression", xlab="λ", ylab="Coefficient-value", log = "x")
nr = 10
legend("bottomright", rownames(ridge_model$beta)[1:10], col=seq_len(nr),
       cex=0.5, lty=seq_len(nr), lwd=1)
```

The lasso regression model was built on the train split from B for a range of values for lambda in the 'grid' variable.

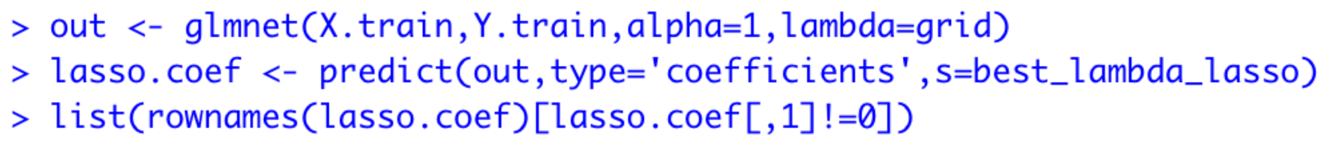


There are a total of 81 features in the data but for the purpose of simplicity, only 10 features were displayed in the plot. For higher values of lambda, we can see that the coefficients become zero as expected from a lasso regression model.

F. ad

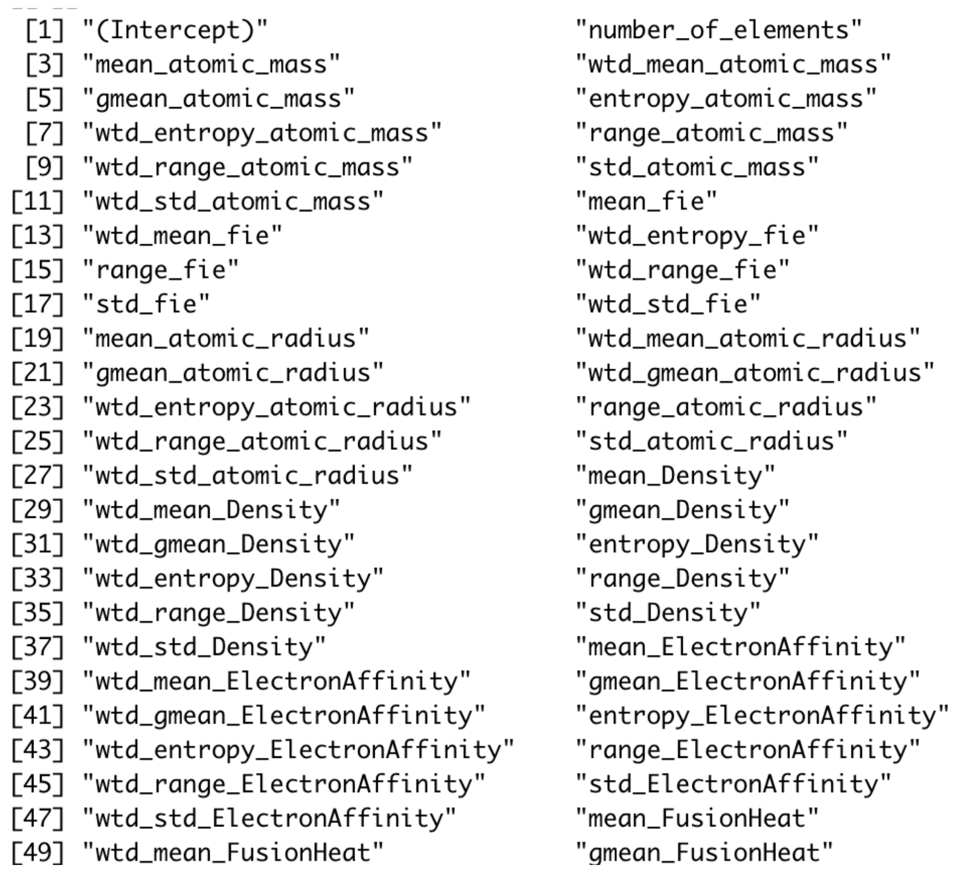
```
> set.seed(5400)
> cv.out.lasso <- cv.glmnet(as.matrix(X.train),as.matrix(Y.train),alpha=1,lambda=grid,
+                           type.measure='mse',nfolds=5)
> plot(cv.out.lasso)
> best_lambda_lasso <- cv.out.lasso$lambda.min
> best_lambda_lasso
[1] 0.01
> cv.out.lasso

Call:  cv.glmnet(x = as.matrix(X.train), y = as.matrix(Y.train), lambda = grid,      type.measure =
 "mse", nfolds = 5, alpha = 1)

Measure: Mean-Squared Error

     Lambda Index Measure    SE Nonzero
min 0.01000   100   319.9 3.706      74
1se 0.01748    97   322.2 3.751      66
>
> lasso.predict_bestlam <- predict(lasso_model,s=best_lambda,newx=as.matrix(X.test))
> MSE_lasso_bestlam <- get_mse(lasso.predict_bestlam,as.matrix(Y.test))
> MSE_lasso_bestlam
[1] 308.9543
```

The lambda = 0.01 provides the best result in this case i.e the minimum estimated test error.
The estimated test error with this lambda is 319.9 whereas the actual test error is 308.95.
The k-fold cross-validation approach is used to estimate the error. The train split from B is used to build models and perform k-fold cross-validation. For every value of lambda, cv error in each fold is calculated and averaged. This is achieved using the function cv.glmnet() with alpha=1 for lasso regression. Finally the lambda for which the mean cv error is the minimum is observed using lambda.min and the corresponding cv error is the estimated test error for that lamda value which is 319.9 in this case for lambda=0.01.



```
> out <- glmnet(X.train,Y.train,alpha=1,lambda=grid)
> lasso.coef <- predict(out,type='coefficients',s=best_lambda_lasso)
> list(rownames(lasso.coef)[lasso.coef[,1]!=0])
```

Out of 81 features, 75 features were included in the lasso model because for the remaining variables,coefficients became zero. Finally, the list of variables used in the model are shown below:



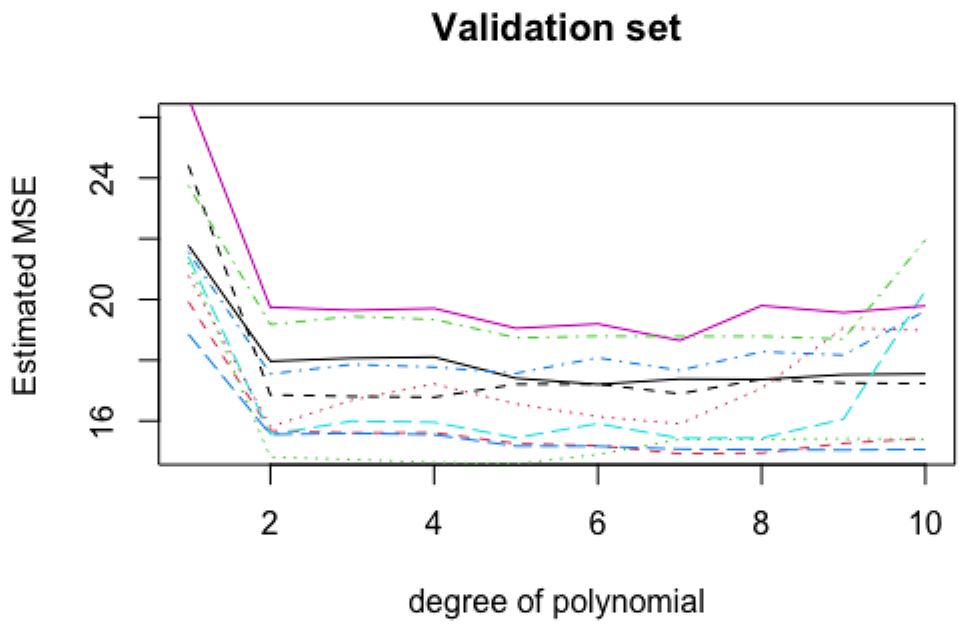
```
 [1] "(Intercept)"                     "number_of_elements"
 [3] "mean_atomic_mass"                "wtd_mean_atomic_mass"
 [5] "gmean_atomic_mass"               "entropy_atomic_mass"
 [7] "wtd_entropy_atomic_mass"         "range_atomic_mass"
 [9] "wtd_range_atomic_mass"           "std_atomic_mass"
[11] "wtd_std_atomic_mass"             "mean_fie"
[13] "wtd_mean_fie"                    "wtd_entropy_fie"
[15] "range_fie"                       "wtd_range_fie"
[17] "std_fie"                         "wtd_std_fie"
[19] "mean_atomic_radius"              "wtd_mean_atomic_radius"
[21] "gmean_atomic_radius"             "wtd_gmean_atomic_radius"
[23] "wtd_entropy_atomic_radius"       "range_atomic_radius"
[25] "wtd_range_atomic_radius"         "std_atomic_radius"
[27] "wtd_std_atomic_radius"           "mean_Density"
[29] "wtd_mean_Density"                "gmean_Density"
[31] "wtd_gmean_Density"               "entropy_Density"
[33] "wtd_entropy_Density"             "range_Density"
[35] "wtd_range_Density"               "std_Density"
[37] "wtd_std_Density"                 "mean_ElectronAffinity"
[39] "wtd_mean_ElectronAffinity"       "gmean_ElectronAffinity"
[41] "wtd_gmean_ElectronAffinity"      "entropy_ElectronAffinity"
[43] "wtd_entropy_ElectronAffinity"    "range_ElectronAffinity"
[45] "wtd_range_ElectronAffinity"      "std_ElectronAffinity"
[47] "wtd_std_ElectronAffinity"        "mean_FusionHeat"
[49] "wtd_mean_FusionHeat"             "gmean_FusionHeat"
```

```
[51] "wtd_gmean_FusionHeat"            "entropy_FusionHeat"
[53] "wtd_entropy_FusionHeat"          "range_FusionHeat"
[55] "wtd_range_FusionHeat"            "std_FusionHeat"
[57] "wtd_std_FusionHeat"              "mean_ThermalConductivity"
[59] "wtd_mean_ThermalConductivity"    "gmean_ThermalConductivity"
[61] "wtd_gmean_ThermalConductivity"   "entropy_ThermalConductivity"
[63] "wtd_entropy_ThermalConductivity" "range_ThermalConductivity"
[65] "wtd_range_ThermalConductivity"   "std_ThermalConductivity"
[67] "wtd_std_ThermalConductivity"     "gmean_Valence"
[69] "wtd_gmean_Valence"               "entropy_Valence"
[71] "wtd_entropy_Valence"             "range_Valence"
[73] "wtd_range_Valence"               "std_Valence"
[75] "wtd_std_Valence"
```

4.

A.

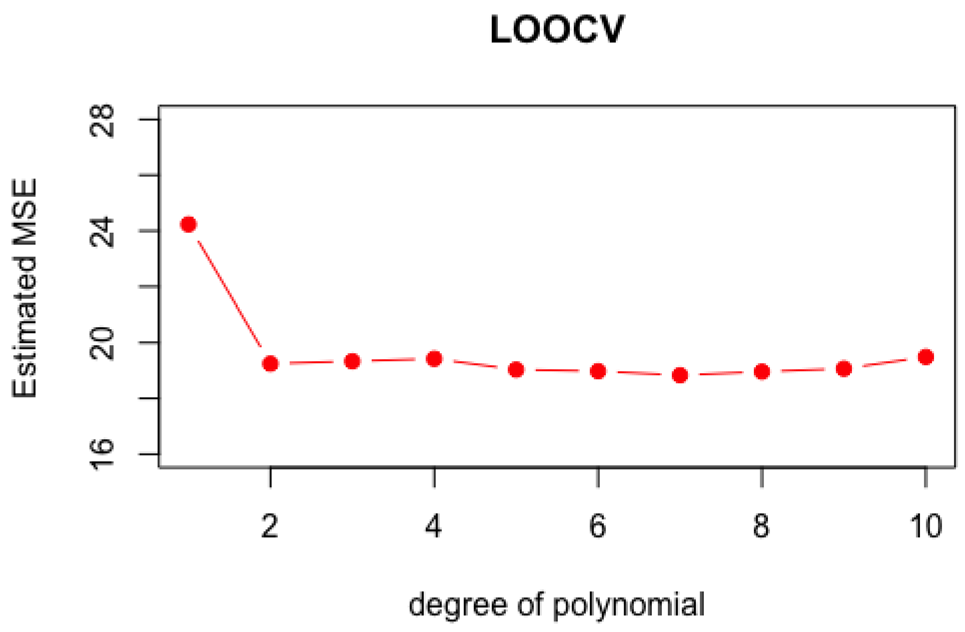
```r
library(ISLR2)

#function to calculate validation test error in training or test
Val_test <- function(data,model)
{
    predict <- predict(model,data)
    MSE <- get_mse(predict,data$mpg)
  return(MSE)
}

seed <- c(100,200,300,400,500,600,700,800,900,1000)
degree <- c(1,2,3,4,5,6,7,8,9,10)
plot_mat <- matrix(nrow=10,ncol=10)
for( j in 1:10){
  set.seed(seed[j])
  spl = sample.split(Auto$mpg, SplitRatio = 0.5)
  train = subset(Auto, spl == TRUE)
  test = subset(Auto, spl == FALSE)

  for(i in 1:10){
    set.seed(7000)
    model <- lm(mpg ~ poly(horsepower,degree[i]),data=train)
    MSE_scores <- append(MSE_scores,Val_test(test,model))
    plot_mat[j,i] <- Val_test(test,model)
  }
}

matplot(t(plot_mat),type='l',ylim=c(15,26),ylab='Estimated MSE',xlab='degree of polynomial')
```



From the several iterations, it was found that with an increase in the degree of the polynomial, the estimated test MSE decreases rapidly initially but then it starts to increase with further increase in degrees because with a very high degree of polynomial the model becomes more flexible and leads to increase in variance that may lead to increase in MSE as well.
It is evident from the plot above.
Also, a decrease in MSE with the increase in degree suggests that Y and X have a non-linear relationship which could not have been explained by just linear terms.
It can also be seen that every set of train and validation set yields a different curve and there is a lot of variation in this approach. It is hard to find which training and validation set gives the lowest MSE.
The degree=2 is the best in my opinion as it produces the lowest MSE with less complexity in all the iterations. In different iterations, there is variability in degrees of polynomial where MSE is the lowest but a higher degree makes the model more complex and will lead to an increase in variance at a cost of slightly lesser bias. It can be observed that a further increase in degree doesn’t add much improvement in terms of MSE.

B.

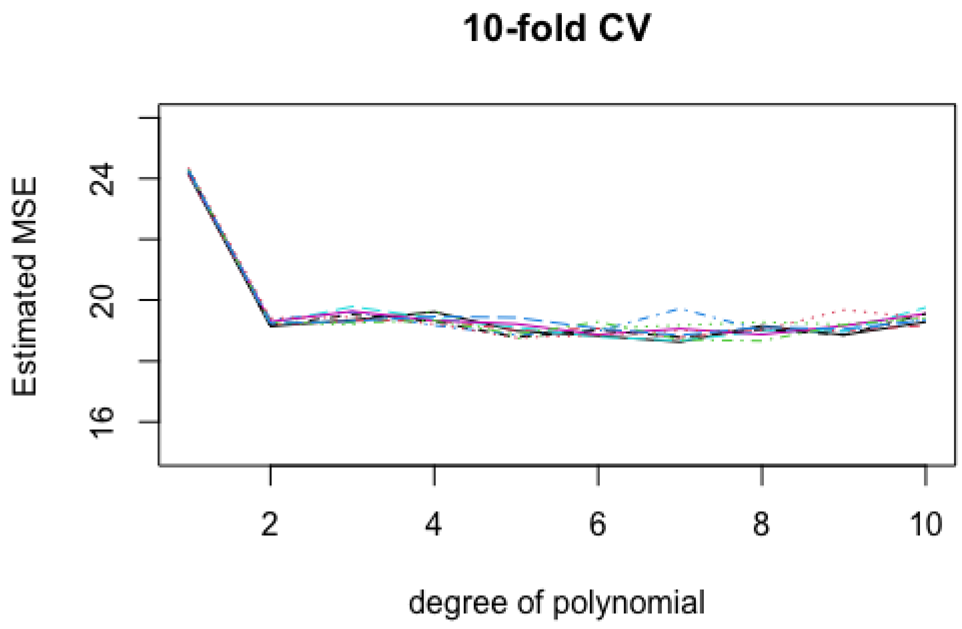
```
library(boot)
cv.error <- rep(0,10)
for(i in 1:10){
  glm.fit <- glm(mpg~poly(horsepower,i),data=Auto)
  cv.error[i] <- cv.glm(Auto,glm.fit)$delta[1]
}

cv.error
plot(c(1:10),cv.error,col='red',type = "b", pch = 19,
     ylim=c(16,28),xlab='degree of polynomial',ylab='Estimated MSE')
```



Using this approach, we will always get the same curve since there is no variation induced here like the validation approach where variation is induced by splitting data and hence leading to different curves. This approach uses all samples except one in each iteration leading to less bias and there is no randomness involved. Here, it can be seen that with an increase in the degree of the polynomial, the estimated test MSE decreases rapidly initially but then it starts to increase or remains the same with a further increase in degrees and adds no improvement.
The degree=2 is the best in my opinion as it produces the lowest MSE with less complexity. We can say degree=5 or degree=7 is the lowest with slightly lesser MSE than degree=2, but it makes the model more complex and will lead to an increase in variance at a cost of slightly lesser bias.

C.

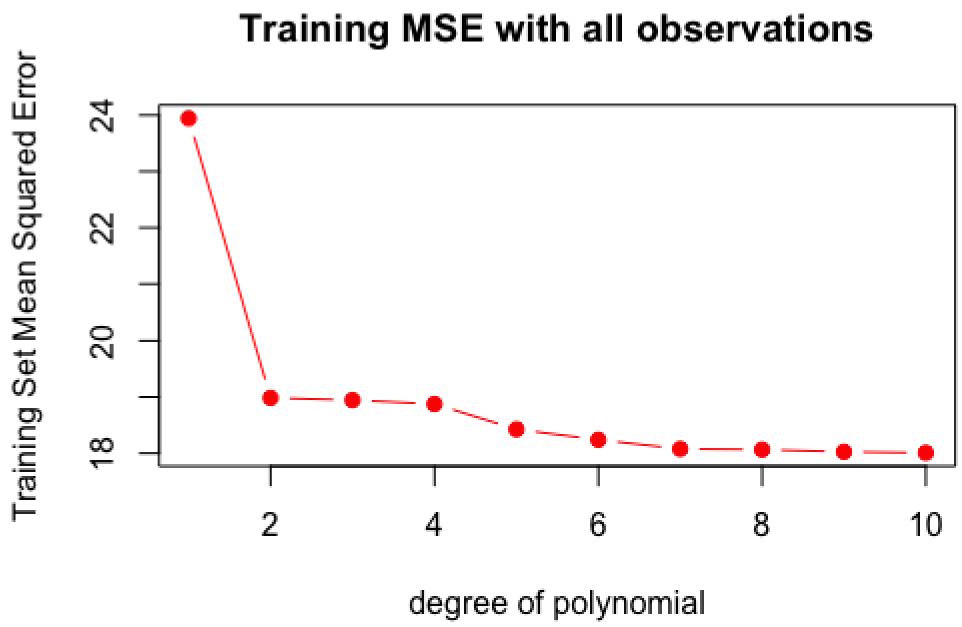
```
seed <- c(100,200,300,400,500,600,700,800,900,1000)
degree <- c(1,2,3,4,5,6,7,8,9,10)
cv.error.mat <- matrix(nrow=10,ncol=10)
for (j in 1:10){
  set.seed(seed[j])
  cv.error.10 <- rep(0,10)
  for (i in 1:10){
    glm.fit <- glm(mpg~poly(horsepower,i),data=Auto)
    cv.error.mat[j,i] <- cv.glm(Auto,glm.fit,K=10)$delta[1]
  }

}

matplot(t(cv.error.mat),type='l',ylim=c(15,26),ylab='Estimated MSE'
        ,xlab='degree of polynomial')
title('10-fold CV')
```



The 10-fold CV was run 10 different times and hence we have 10 different curves. We can see there is some variability in CV estimates due to a random split between folds but in this approach, there is larger overlap of training samples in each iteration and hence the variability is low as compared to the validation set approach.
The degree=2 is the best in my opinion as it produces the lowest MSE with less complexity.

D.

```
MSE_10 <- rep(0,10)
for (i in 1:10){

  lm.fit <- lm(mpg~poly(horsepower,i),data=Auto)
  MSE_10[i] <- Val_test(Auto,lm.fit)
}

plot(c(1:10),MSE_10,col='red',type = "b", pch = 19,xlab='degree of polynomial',
     ylab='Training Set Mean Squared Error')
title('Training MSE with all observations')
```



It can be observed that increasing the level of flexibility by increasing degrees of polynomial leads to better results. But the level of improvement decreases with an increase in the flexibility of the model. There is a sharp decline in MSE at degree=2 but the slope decrease as we increase the degree further. Since this is the training MSE, we can expect the MSE to decrease with an increase in degree but the variance will go up which can lead to worse test MSE.

E.

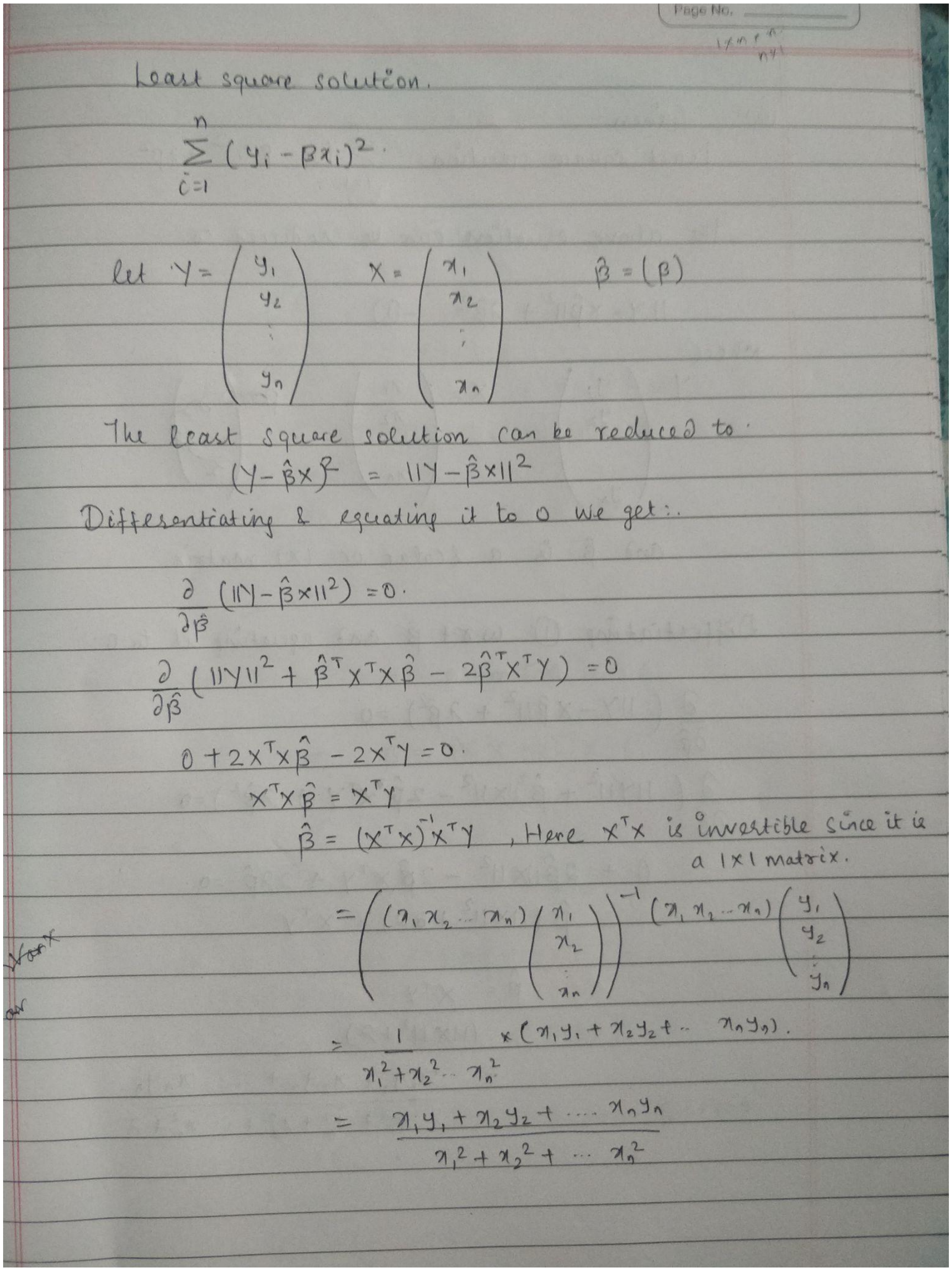
```
> lm.fit <- lm(mpg~poly(horsepower,10),data=Auto)
> MSE_10 <- Val_test(Auto,lm.fit)
> summary(lm.fit)

Call:
lm(formula = mpg ~ poly(horsepower, 10), data = Auto)

Residuals:
     Min       1Q   Median       3Q      Max 
-15.7081  -2.5904  -0.1922   2.2859  14.8338 

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)    
(Intercept)              23.4459     0.2174 107.840   <2e-16 ***
poly(horsepower, 10)1  -120.1377     4.3046 -27.909   <2e-16 ***
poly(horsepower, 10)2    44.0895     4.3046  10.242   <2e-16 ***
poly(horsepower, 10)3    -3.9488     4.3046  -0.917   0.3595    
poly(horsepower, 10)4    -5.1878     4.3046  -1.205   0.2289    
poly(horsepower, 10)5    13.2722     4.3046   3.083   0.0022 ** 
poly(horsepower, 10)6    -8.5462     4.3046  -1.985   0.0478 *  
poly(horsepower, 10)7     7.9806     4.3046   1.854   0.0645 .  
poly(horsepower, 10)8     2.1727     4.3046   0.505   0.6140    
poly(horsepower, 10)9    -3.9182     4.3046  -0.910   0.3633    
poly(horsepower, 10)10   -2.6146     4.3046  -0.607   0.5440    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.305 on 381 degrees of freedom
Multiple R-squared:  0.7036,	Adjusted R-squared:  0.6958 
F-statistic: 90.45 on 10 and 381 DF,  p-value: < 2.2e-16
```

From the results, we can see that the p-value of polynomial terms with power - (1,2,5,6) are lower than the alpha value of 0.05 which suggests that these terms are significant. Also, the absolute values of the coefficients of polynomial terms with power 1 and 2 are the highest among others. This implies that there is some non-linearity and the non-linearity is mostly explained by the first two polynomial terms.
This inference aligns with the results in the previous questions where we saw that there is a sharp decrease in MSE in the second degree of the polynomial and a further increase in degrees didn’t lead to much improvement in the MSE. There was a further slight decrease in MSE with degree=5 and it can be seen from the result of the above fit that the coefficient of the polynomial term with power=5 is also significant.
This means we may not need a higher degree polynomial than 2 to explain the non-linearity.

Khirod Sahoo HW3 DATA 558

5.

A.



Least square solution.

$$\sum_{i=1}^{n}(y_i - \beta x_i)^2$$

let $Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$ $X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ $\hat{\beta} = (\beta)$

The least square solution can be reduced to:

$$(Y - \hat{\beta}X)^2 = \|Y - \hat{\beta}X\|^2$$

Differentiating & equating it to 0 we get:

$$\frac{\partial}{\partial\hat{\beta}}(\|Y - \hat{\beta}X\|^2) = 0.$$

$$\frac{\partial}{\partial\hat{\beta}}(\|Y\|^2 + \hat{\beta}^T X^T X\hat{\beta} - 2\hat{\beta}^T X^T Y) = 0$$

$$0 + 2X^TX\hat{\beta} - 2X^TY = 0.$$

$$X^TX\hat{\beta} = X^TY$$

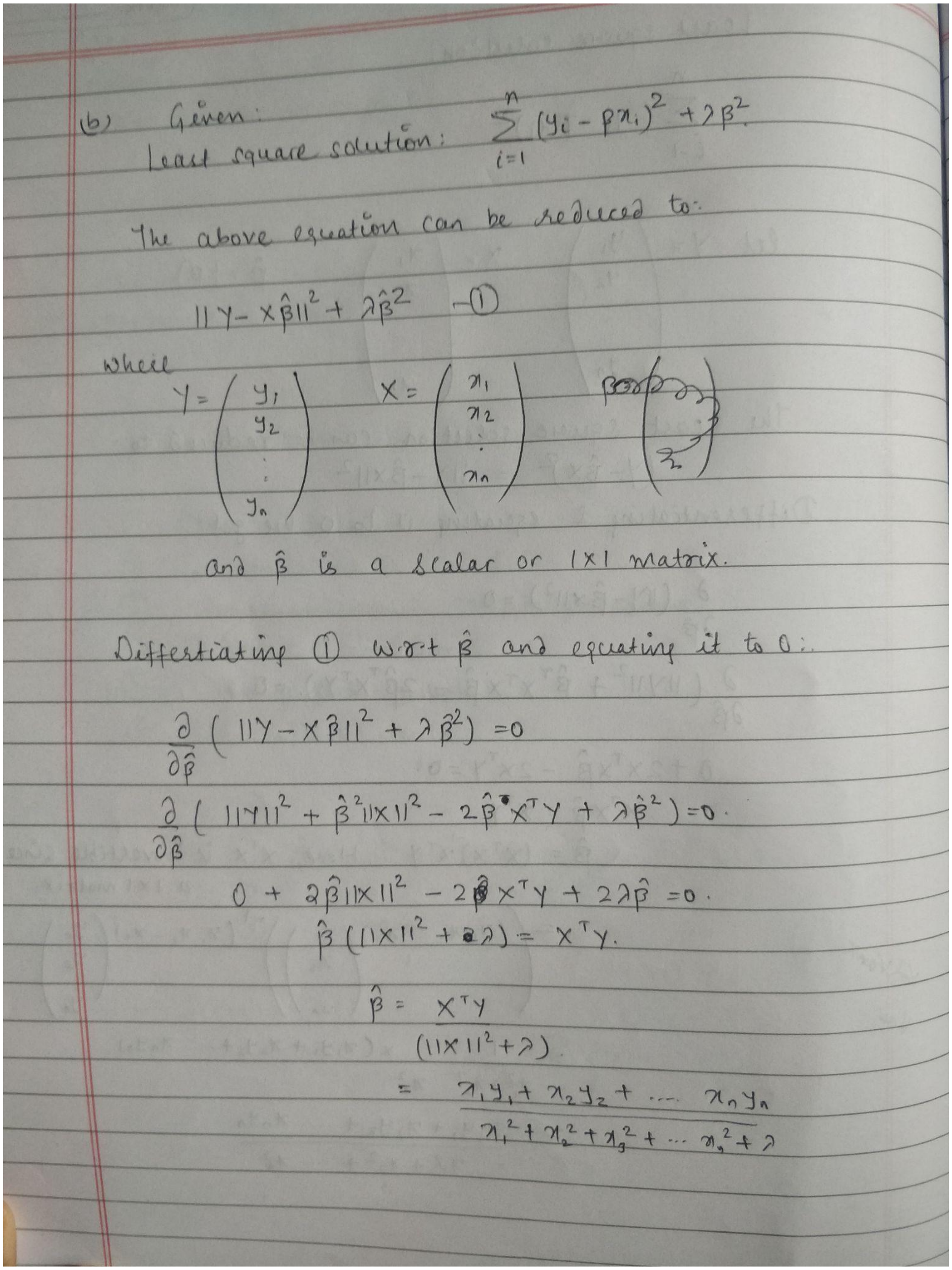
$\hat{\beta} = (X^TX)^{-1}X^TY$, Here X^TX is invertible since it is a 1×1 matrix.

$$= \left((x_1\, x_2 \dots x_n)\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}\right)^{-1}(x_1\, x_2 \dots x_n)\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \frac{1}{x_1^2 + x_2^2 \dots x_n^2} \times (x_1y_1 + x_2y_2 + \dots x_ny_n).$$

$$= \frac{x_1y_1 + x_2y_2 + \dots x_ny_n}{x_1^2 + x_2^2 + \dots x_n^2}$$

B.



(b) Given:
Least square solution: $\sum_{i=1}^{n} (y_i - \beta x_i)^2 + \lambda \beta^2$.

The above equation can be reduced to:

$$\|Y - X\hat{\beta}\|^2 + \lambda \hat{\beta}^2 \quad -①$$

where

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

[illegible]

and $\hat{\beta}$ is a scalar or 1×1 matrix.

Differentiating ① w.r.t $\hat{\beta}$ and equating it to 0:

$$\frac{\partial}{\partial \hat{\beta}} \left(\|Y - X\hat{\beta}\|^2 + \lambda \hat{\beta}^2 \right) = 0$$

$$\frac{\partial}{\partial \hat{\beta}} \left(\|Y\|^2 + \hat{\beta}^2 \|X\|^2 - 2\hat{\beta} X^T Y + \lambda \hat{\beta}^2 \right) = 0.$$

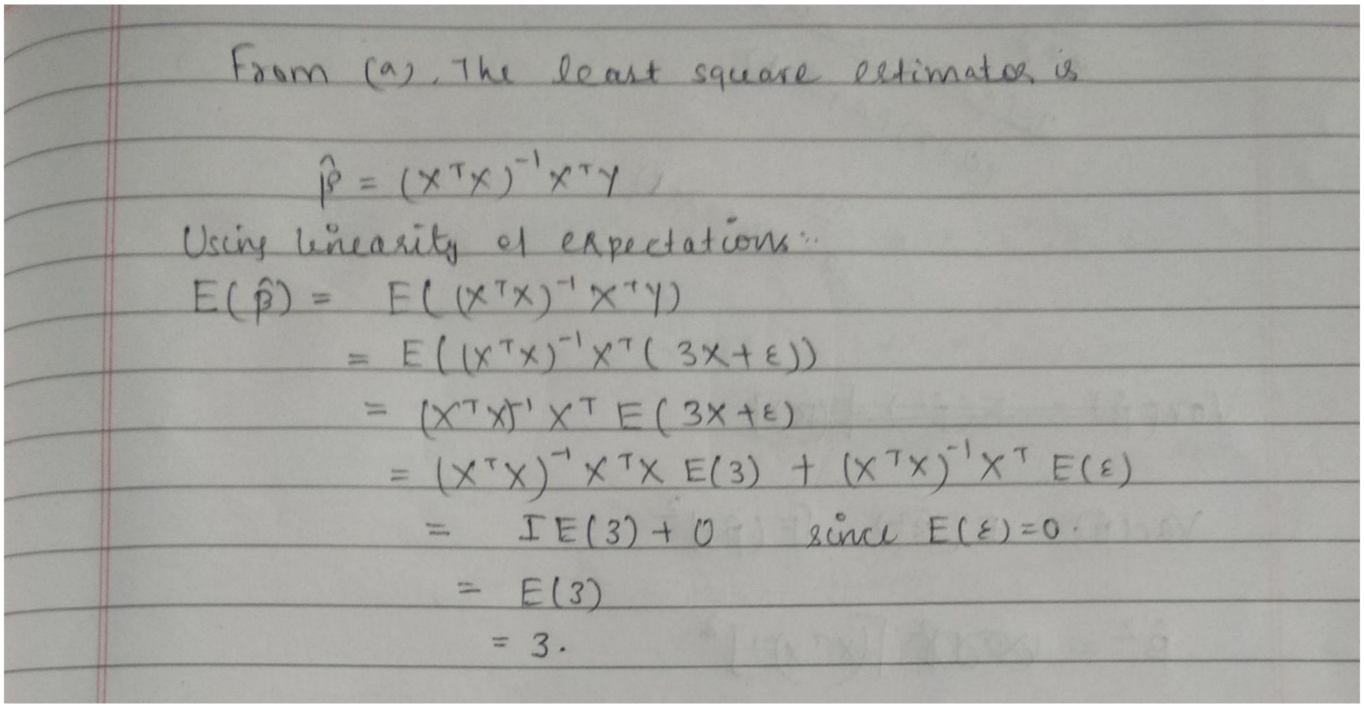
$$0 + 2\hat{\beta}\|X\|^2 - 2 X^T Y + 2\lambda\hat{\beta} = 0.$$

$$\hat{\beta}(\|X\|^2 + \lambda) = X^T Y.$$

$$\hat{\beta} = \frac{X^T Y}{(\|X\|^2 + \lambda)}$$

$$= \frac{x_1 y_1 + x_2 y_2 + \cdots x_n y_n}{x_1^2 + x_2^2 + x_3^2 + \cdots x_n^2 + \lambda}$$

C. Assuming X is fixed. The expectation here is unbiased because it is same as the beta of the true generating function.



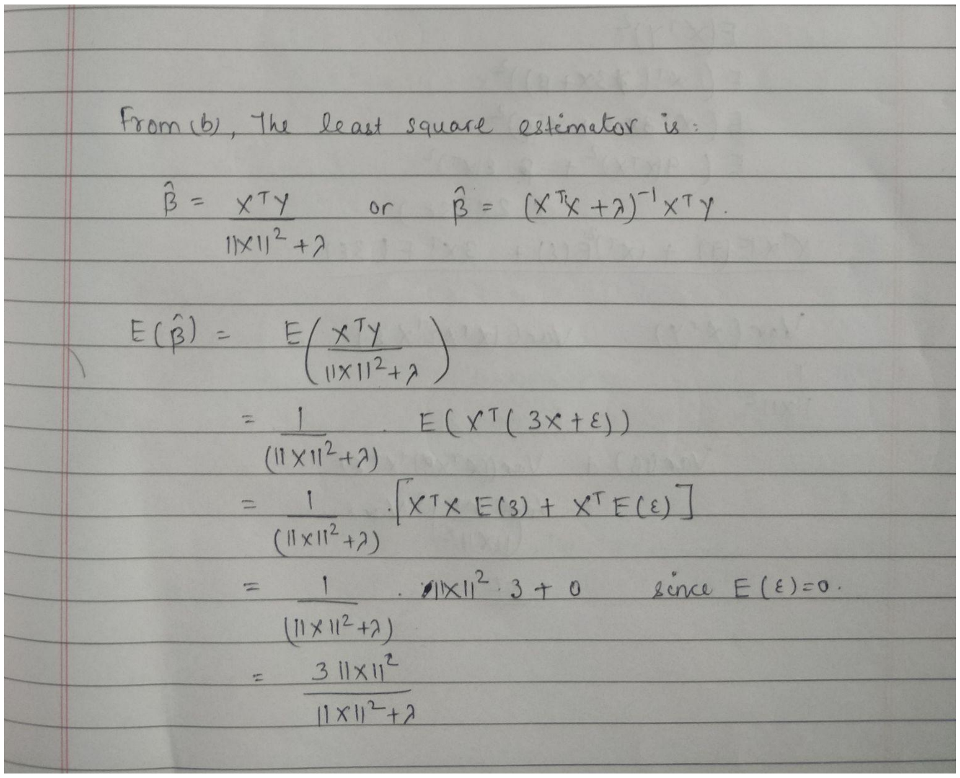
From (a), The least square estimator is

$$\hat{\beta} = (X^TX)^{-1}X^TY$$

Using linearity of expectations:

$$E(\hat{\beta}) = E((X^TX)^{-1}X^TY)$$
$$= E((X^TX)^{-1}X^T(3X+\varepsilon))$$
$$= (X^TX)^{-1}X^T E(3X+\varepsilon)$$
$$= (X^TX)^{-1}X^TX\,E(3) + (X^TX)^{-1}X^T E(\varepsilon)$$
$$= I\,E(3) + 0 \quad \text{since } E(\varepsilon)=0.$$
$$= E(3)$$
$$= 3.$$

D. Is is biased. As the lambda increases, it will also increase the bias. With an increase in lambda, the expected beta decreases, this will the change the estimated function of X and hence bias will change. Similarly, decreasing lambda will also lead to change in bias. In both the case bias will decrease.

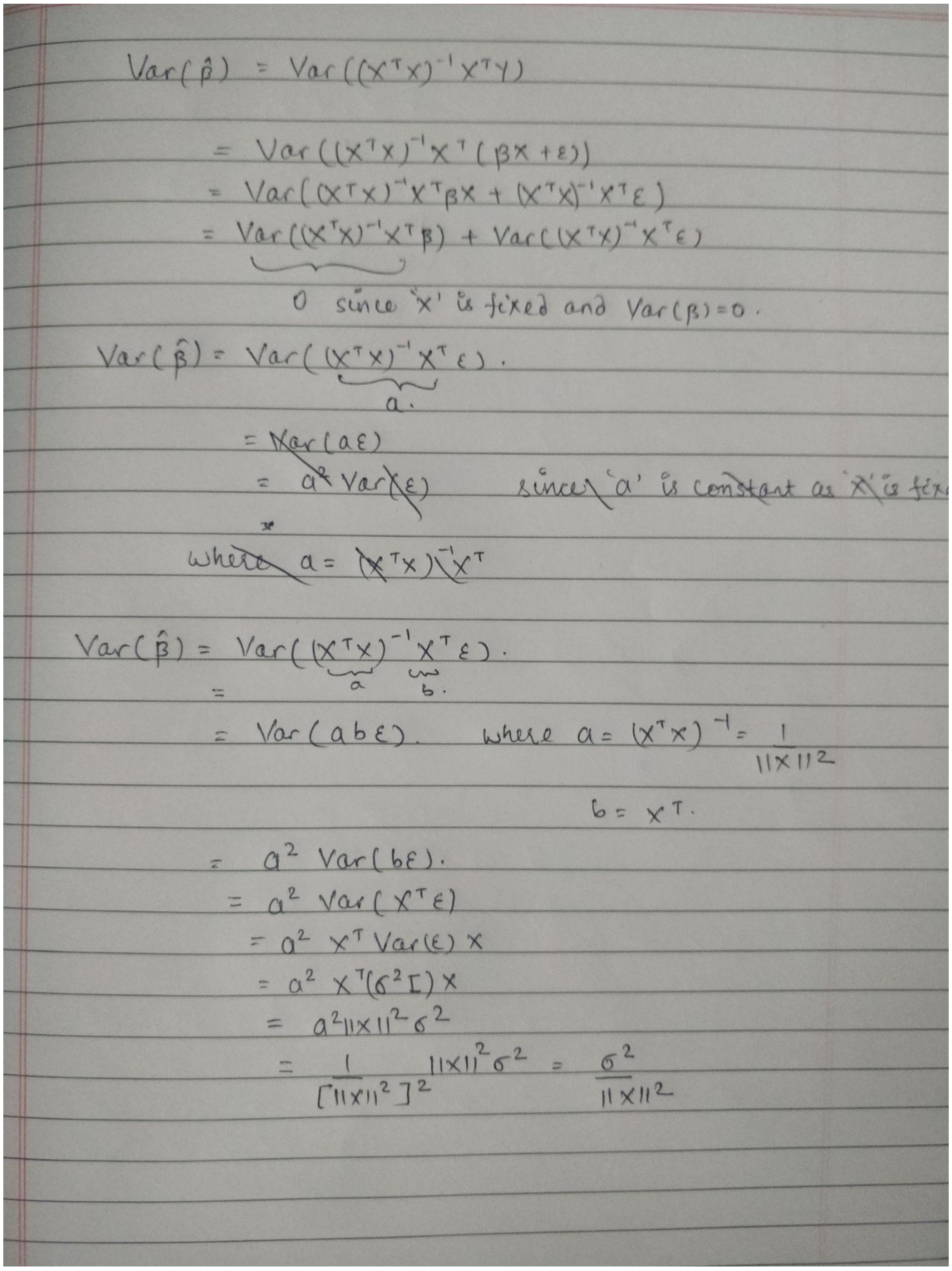


From (b), The least square estimator is:

$$\hat{\beta} = \frac{X^TY}{\|X\|^2+\lambda} \quad \text{or} \quad \hat{\beta} = (X^TX+\lambda)^{-1}X^TY.$$

$$E(\hat{\beta}) = E\left(\frac{X^TY}{\|X\|^2+\lambda}\right)$$
$$= \frac{1}{(\|X\|^2+\lambda)} \cdot E(X^T(3X+\varepsilon))$$
$$= \frac{1}{(\|X\|^2+\lambda)} \cdot [X^TX\,E(3) + X^TE(\varepsilon)]$$
$$= \frac{1}{(\|X\|^2+\lambda)} \cdot \|X\|^2 \cdot 3 + 0 \quad \text{since } E(\varepsilon)=0.$$
$$= \frac{3\|X\|^2}{\|X\|^2+\lambda}$$

E. Assuming X is fixed, Cov(X,e) =0, and all the error terms are independent since covariance is zero for them.



$$Var(\hat{\beta}) = Var((X^TX)^{-1}X^TY)$$

$$= Var((X^TX)^{-1}X^T(\beta X + \varepsilon))$$

$$= Var((X^TX)^{-1}X^T\beta X + (X^TX)^{-1}X^T\varepsilon)$$

$$= \underbrace{Var((X^TX)^{-1}X^T\beta)}_{0} + Var((X^TX)^{-1}X^T\varepsilon)$$

0 since 'x' is fixed and $Var(\beta)=0$.

$$Var(\hat{\beta}) = Var(\underbrace{(X^TX)^{-1}}_{a}X^T\varepsilon).$$

~~$= Var(a\varepsilon)$~~

~~$= a^2 Var(\varepsilon)$ since 'a' is constant as 'X' is fix~~

~~where $a = (X^TX)^{-1}X^T$~~

$$Var(\hat{\beta}) = Var(\underbrace{(X^TX)^{-1}}_{a}\underbrace{X^T}_{b}\varepsilon).$$

$$=$$

$$= Var(ab\varepsilon). \quad \text{where } a = (X^TX)^{-1} = \frac{1}{\|X\|^2}$$

$$b = X^T.$$

$$= a^2 Var(b\varepsilon).$$

$$= a^2 Var(X^T\varepsilon)$$

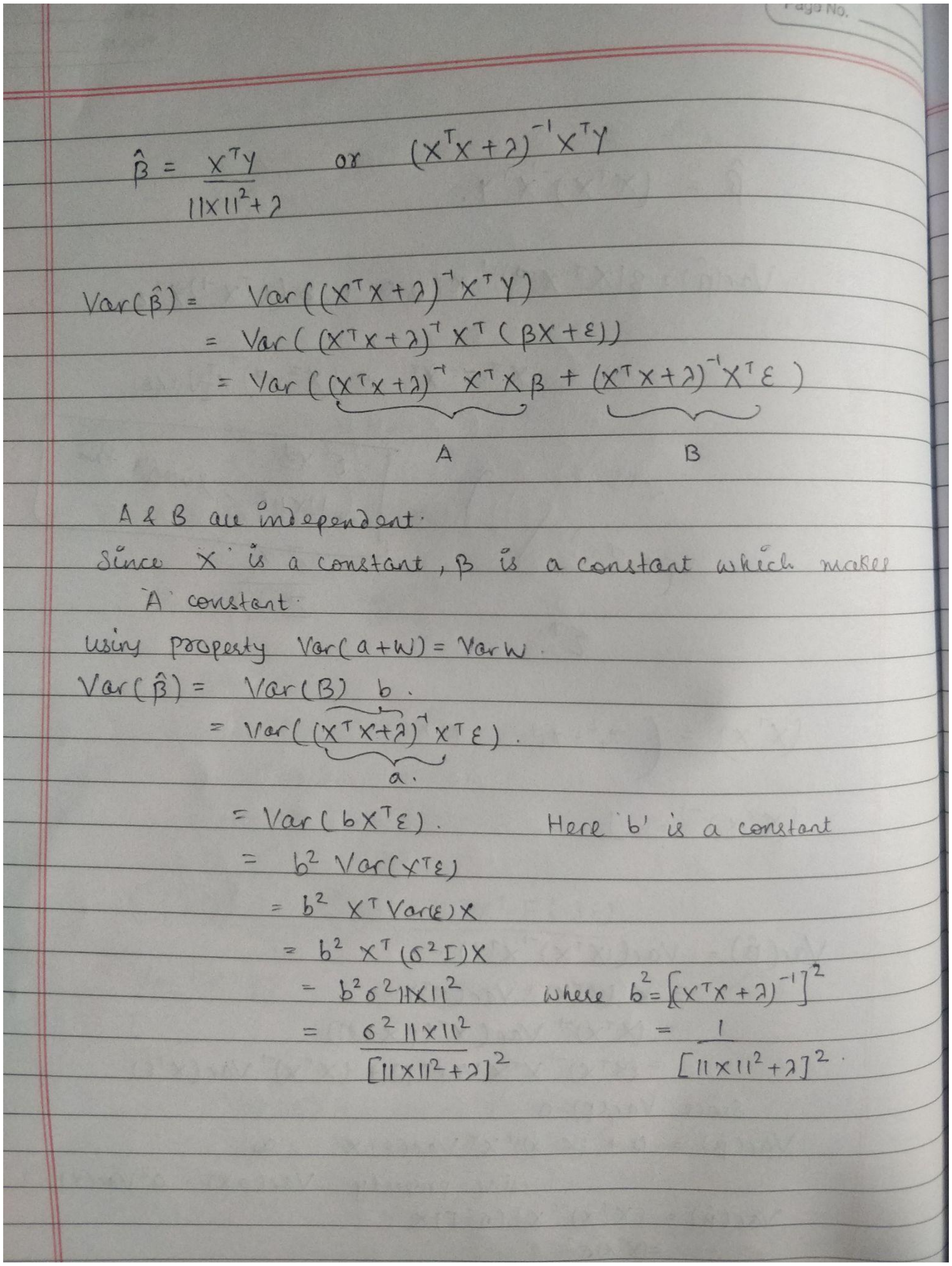
$$= a^2 X^T Var(\varepsilon) X$$

$$= a^2 X^T(\sigma^2 I) X$$

$$= a^2 \|X\|^2 \sigma^2$$

$$= \frac{1}{[\|X\|^2]^2} \|X\|^2 \sigma^2 = \frac{\sigma^2}{\|X\|^2}$$

F. As the lambda increases, it decreases the variance by a squared quantity which is a function of lambda. Hence, variance is directly affected a lot by change in lambda value.



$\hat{\beta} = \frac{X^TY}{||X||^2+\lambda}$ or $(X^TX+\lambda)^{-1}X^TY$

$$\begin{aligned} Var(\hat{\beta}) &= Var((X^TX+\lambda)^{-1}X^TY) \\ &= Var((X^TX+\lambda)^{-1}X^T(\beta X+\varepsilon)) \\ &= Var(\underbrace{(X^TX+\lambda)^{-1}X^TX\beta}_{A} + \underbrace{(X^TX+\lambda)^{-1}X^T\varepsilon}_{B}) \end{aligned}$$

A & B are independent.
Since X is a constant, β is a constant which makes 'A' constant.
using property $Var(a+W) = Var\, W$.

$$\begin{aligned} Var(\hat{\beta}) &= Var(B) \quad b. \\ &= Var(\underbrace{(X^TX+\lambda)^{-1}}_{a.}X^T\varepsilon). \\ &= Var(bX^T\varepsilon). \qquad \text{Here 'b' is a constant} \\ &= b^2\, Var(X^T\varepsilon) \\ &= b^2\, X^T Var(\varepsilon) X \\ &= b^2\, X^T(\sigma^2 I)X \\ &= b^2\sigma^2||X||^2 \qquad \text{where } b^2 = [(X^TX+\lambda)^{-1}]^2 \\ &= \frac{\sigma^2||X||^2}{[||X||^2+\lambda]^2} \qquad = \frac{1}{[||X||^2+\lambda]^2}. \end{aligned}$$

G. As we have seen above, in d and f, lambda can be used to control bias and variance. In d, we saw that the expected value of beta is inversely proportional to lambda but the changing the lambda won’t affect beta estimate much unless lambda is a large value. So the bias is changed slightly with increase in lambda value. On the other hand, in f, variance is inversely proportional to a squared quantity which is a function of lambda. Hence, we can control variance better with lambda i.e variance is affected more than bias due to change in lambda. This way, lambda can be used as tuning parameter for model complexity by trading off bias for variance.