****All code is at the end****

PROBLEM 1: PART 1

Method: In forward stepwise selection, we start from the null model and iteratively add a variable to the model depending on which new added variable results in the minimum RSS. We continue until we have the full model i.e. all variables have been added. Then, we calculate the BIC for each model in order to find the best model.

Results:

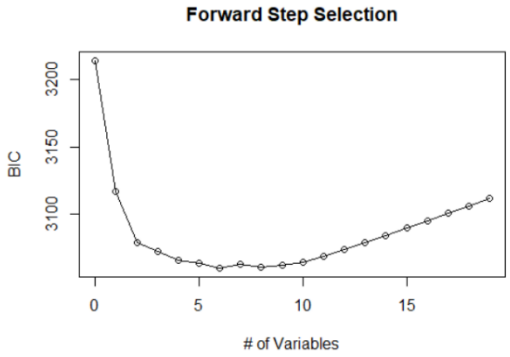
The **6 variable model** is the best model. It has a corresponding **BIC of 3060.279**. The selected model is:

$$y = 214.4633319 + 0.6430169*x_{CRBI} + 7.6043976*x_{Hits} + 0.2643076*x_{PutOuts} +$$

$$-122.9515338*x_{Division} + -1.8685892*x_{AtBat} + 3.6976468*x_{Walks}$$

The graph of BIC vs. # of Variables is below:

Figure 1.



PROBLEM 1: PART 2

Method: In backward stepwise selection, we start from the full model and iteratively remove a variable from the model depending on which remaining variables result in the minimum RSS. We continue until we have the null model i.e. all variables have been removed. Then, we calculate the BIC for each model in order to find the best model.

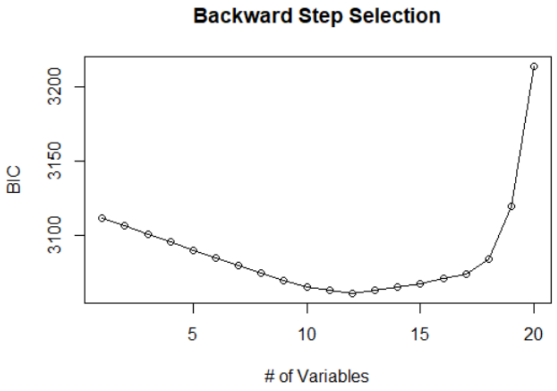
Results:

The **12 variable model** is the best model. It has a corresponding **BIC of 3060.814**. The selected model is:

$$y = -34.0919147 + 0.2783995*x_{Runs} + -0.4441261*x_{League} + 0.2725490*x_{Assists} + 103.7370075*x_{CAtBat} + 3.9193776*x_{CRBI} + -4.7445704*x_{CWalks} + -5.4192673*x_{Division} + 1.5583061*x_{Walks} + 2.5820753*x_{AtBat} + -34.3835149*x_{PutOuts} + 3.0732565*x_{Hits} + 1.7826529*x_{CRuns}$$

The graph of BIC vs. # of Variables is below:

Figure 2.



PROBLEM 1: PART 3

The selected models from Part 1 and 2 are **NOT** the same. The model chosen by forward stepwise selection has 6 variables, while the model chosen by backward stepwise selection has 12 variables. However, we can note that the BIC from both models are about equal. Furthermore, all 6 variables in the first model are also included in the second model.

PROBLEM 2: PART 1

Method: In order to find beta (20x1) vector for each lambda, I first used the standardized X and centered y data to calculate omega (19x1) vector through the closed form Ridge Regression formula. Then, I recovered the omegas back to betas.

Results:

Below is just a part of the full 20x100 matrix:

Figure 3.

1	5.359258e+02	5.359257e+02	5.359257e+02	5.359256e+02	5.359256e+02	5.359255e+02	5.359253e+02	5.359251e+02	5.359249e+02	5.359246e+02	5.359242e+02	5.359236e+02
2	3.179565e-08	4.203197e-08	5.556379e-08	7.345205e-08	9.709927e-08	1.283595e-07	1.696837e-07	2.243117e-07	2.965268e-07	3.919908e-07	5.181883e-07	6.850138e-07
3	1.153370e-07	1.524688e-07	2.015547e-07	2.664434e-07	3.522225e-07	4.656173e-07	6.155186e-07	8.136791e-07	1.075636e-06	1.421926e-06	1.879701e-06	2.484852e-06
4	4.647458e-07	6.143665e-07	8.121563e-07	1.073623e-06	1.419266e-06	1.876186e-06	2.480206e-06	3.278685e-06	4.334227e-06	5.729590e-06	7.574174e-06	1.001260e-05
5	1.950436e-07	2.578362e-07	3.408442e-07	4.505759e-07	5.956348e-07	7.873940e-07	1.040888e-06	1.375993e-06	1.818981e-06	2.404584e-06	3.178716e-06	4.202072e-06
6	2.060274e-07	2.723560e-07	3.600386e-07	4.759498e-07	6.291775e-07	8.317355e-07	1.099505e-06	1.453481e-06	1.921415e-06	2.539996e-06	3.357723e-06	4.438708e-06
7	2.424815e-07	3.205463e-07	4.237433e-07	5.601636e-07	7.405033e-07	9.789016e-07	1.294050e-06	1.710657e-06	2.261388e-06	2.989420e-06	3.951834e-06	5.224088e-06
8	9.916444e-07	1.310895e-06	1.732926e-06	2.290827e-06	3.028337e-06	4.003283e-06	5.292103e-06	6.995846e-06	9.248092e-06	1.222543e-05	1.616128e-05	2.136424e-05
9	2.729964e-09	3.608852e-09	4.770689e-09	6.306570e-09	8.336913e-09	1.102091e-08	1.456899e-08	1.925934e-08	2.545970e-08	3.365620e-08	4.449148e-08	5.881506e-08
10	1.004705e-08	1.328161e-08	1.755750e-08	2.320998e-08	3.068222e-08	4.056008e-08	5.361803e-08	7.087986e-08	9.369896e-08	1.238644e-07	1.637414e-07	2.164562e-07
11	7.576861e-08	1.001616e-07	1.324078e-07	1.750353e-07	2.313863e-07	3.058790e-07	4.043539e-07	5.345319e-07	7.066194e-07	9.341087e-07	1.234836e-06	1.632379e-06
12	2.015662e-08	2.664587e-08	3.522426e-08	4.656440e-08	6.155538e-08	8.137258e-08	1.075697e-07	1.422008e-07	1.879810e-07	2.484997e-07	3.285017e-07	4.342595e-07
13	2.080207e-08	2.749912e-08	3.635221e-08	4.805548e-08	6.352651e-08	8.397829e-08	1.110143e-07	1.467544e-07	1.940005e-07	2.564571e-07	3.390210e-07	4.481654e-07
14	2.200843e-08	2.909384e-08	3.846034e-08	5.084230e-08	6.721052e-08	8.884834e-08	1.174522e-07	1.552649e-07	2.052510e-07	2.713295e-07	3.586814e-07	4.741551e-07
15	-3.387975e-07	-4.478702e-07	-5.920577e-07	-7.826650e-07	-1.034636e-06	-1.367727e-06	-1.808052e-06	-2.390134e-06	-3.159609e-06	-4.176803e-06	-5.521465e-06	-7.299011e-06
16	-4.560275e-06	-6.028415e-06	-7.969209e-06	-1.053483e-05	-1.392642e-05	-1.840990e-05	-2.433681e-05	-3.217182e-05	-4.252925e-05	-5.622116e-05	-7.432104e-05	-9.824801e-05
17	1.273521e-08	1.683520e-08	2.225514e-08	2.941998e-08	3.889148e-08	5.141224e-08	6.796394e-08	8.984432e-08	1.187689e-07	1.570054e-07	2.075518e-07	2.743711e-07

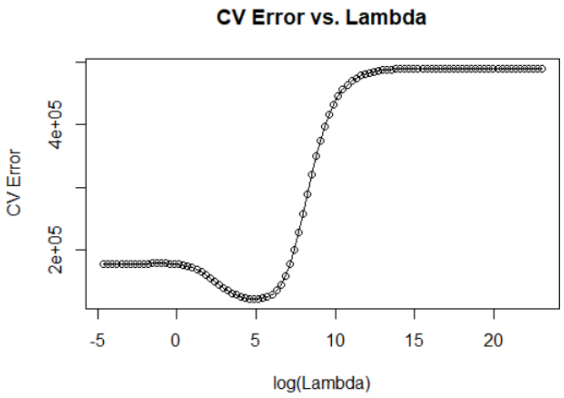
PROBLEM 2: PART 2

Method: First, I divided the data matrix X into 10 folds. My training set is defined as all of X excluding 1 one of these 10 folds, and my validation set is defined as only 1 of these 10 folds. Hence, I iterated through each of the 10 folds, assigned training and validation X, and used training X to calculate an omega, recovered it to a beta, and finally, calculated the prediction error using this beta on my X and y validation set. Then, I set the cross validation error as the mean of the prediction errors from each fold. I repeated this process for each of the 100 lambdas and selected the best lambda by finding the minimum cross validation error.

Results:

The best lambda is **132.1941**. The Cross-Validation Error vs. log(lambda) curve is below:

Figure 4.



PROBLEM 2: PART 3

Method: I used my computed matrix from Part 1 and selected the column that corresponds to the best lambda I found in Part 2.

Results:

Below are the coefficient estimates (beta: 20x1 vector):

Figure 5.

```
> betaBest
 [1]  65.58244292   0.04032386   0.98353983   0.22138773   1.10706890   0.87566359   1.77440079   0.38266721
 [9]   0.01123191   0.06347518   0.44241104   0.12602959   0.13460161   0.03396996  26.37565677 -89.78116635
[17]   0.18769756   0.04008479  -1.74091911   7.63891910
```

PROBLEM 3: PART 1

Method: Wrote the GraHTP algorithm as stated in assignment.

PROBLEM 3: PART 2

Method: Used the function from Part 1 on the baseball data set to find the best beta for each k-sparse model. Then used the BIC to determine the best of the k-sparse models.

Results:

The **1 variable model** is the best model. It has a corresponding **BIC of 3396.908**. The selected model is:

$$y = 797.2713779 + -0.7909536 * x_{CRBI}$$

PROBLEM 3: PART 3

This result is very different from those obtained in Problem 1. It has only 1 variable. However, the BIC of this model is still fairly close to the BIC of the models from Problem 1.

```
df <- read.csv(file="hitters.csv", header=TRUE, row.names=1)
dataMatrix <- data.matrix(df)
y <- dataMatrix[,19] #salary col
X <- dataMatrix[,-19] #263x19 matrix of predictors

#PROBLEM 1: Part 1--------------------------------------------------------------------------

#M = matrix of "best" variables in order M1 to M19
M <- matrix(0, nr=263, nc=19)

#store min RSS of each model M0 to M19
rssFinal <- numeric(20)

#M0 = sample mean of y
M0 <- mean(y)
#RSS for model M0
rssFinal[1] <- ( norm((y-M0),type="2") )^2

#col 1-19 = M1 to M19 = col's of X in order of best fit
for(k in 1:19)
{
  #create vector for RSS of each possible col to add to model
  rss <- numeric(20-k)
  for(j in 1:(20-k))
  {
    #calculate RSS for the possible models with k variables
    if(k == 1) {
      possibleModel <- as.matrix(X[,j])
    }
    else if((20-k) > 1) {
      possibleModel <- cbind(M[,1:k-1], as.matrix(X[,j]))
    }
    else {
      #adding last col left in X since k = 19
      X <- as.matrix(X)
      possibleModel <- cbind(M[,1:k-1], X)
    }

    rss[j] = (norm( as.numeric((lm(y~possibleModel))$residuals), type="2" ))^2
  }
  #store min rss for model with k variables
  rssFinal[k+1] <- min(rss)
  #add "best" col to M
  M[,k] = X[,which.min(rss)]
  #remove "best" col from matrix X
  if(ncol(X) > 1) {
    X <- X[,-which.min(rss)]
  }
}

#calculate BIC for all models
bic <- 263*log(rssFinal/263) + log(263)*c(0:19)
bestModelBIC <- min(bic)
#num of variables in the best model
bestModelNum <- which.min(bic) - 1
#beta of best model
beta <- (lm(y~M[,1:bestModelNum]))$coefficients

#plot BIC vs. # of variables
plot(c(0:19), bic, main="Forward Step Selection", xlab="# of Variables", ylab="BIC")
lines(c(0:19), bic, type="l")


#PROBLEM 1: Part 2--------------------------------------------------------------------------
#reset X
X <- dataMatrix[,-19] #263x19 matrix of predictors
y <- dataMatrix[,19] #salary col

#matrix of order of col removal from 19 variables in X
MBack <- matrix(0, nr=263, nc=18)

#store min RSS of each model Mk
rssFinalBack <- numeric(20)

#M0 = sample mean of y
M0 <- mean(y)
#RSS for model M0
rssFinalBack[1] <- ( norm((y-M0), type="2") )^2
#RSS for model M19
rssFinalBack[20] <- ( norm(as.numeric((lm(y~X))$residuals), type="2") )^2

for(k in 0:17)
{
  #create vector for RSS of each possible col to remove from model
  rssBack <- numeric(19-k)
  for(j in 1:(19-k))
  {
    #calculate RSS for each model with 18-k variables
    possibleModel <- as.matrix(X[,-j])
    rssBack[j] = (norm( as.numeric((lm(y~possibleModel))$residuals), type="2" ))^2
  }
  #store min rss for model with 18-k variables
  rssFinalBack[19-k] <- min(rssBack)
  #add "best" col to remove from X
  MBack[,k+1] = X[,which.min(rssBack)]
  #remove col from matrix X leaving cols with "best" RSS
  X <- X[,-which.min(rssBack)]
}
#add last col remaining from X
MBack <- cbind(MBack, matrix(X))

#calculate BIC for all models
bicBack <- 263*log(rssFinalBack/263) + log(263)*c(0:19)
bestModelBack <- min(bicBack)
#num of variables in the best model
bestModelNumBack <- 19 - which.min(bicBack) + 2
#beta of best model
beta <- (lm(y~M[,8:19]))$coefficients

#plot BIC vs. # of variables
plot(c(20:1), bicBack, main="Backward Step Selection", xlab="# of Variables", ylab="BIC")
lines(c(20:1), bicBack, type="l")


#PROBLEM 2: Part 1--------------------------------------------------------------------------
#reset X and y
df <- read.csv(file="hitters.csv", header=TRUE, row.names=1)
dataMatrix <- data.matrix(df)
X <- dataMatrix[,-19] #263x19 matrix of predictors
y <- matrix(dataMatrix[,19]) #salary col

#choose grid of lambda values:
grid <- 10^seq(10,-2,length=100)

#standardizing data X
standardizeX <- function(X)
{
```

```
  mX <- matrix(colMeans(X), nr=dim(X)[1], nc=dim(X)[2], byrow=T)
  X0 <- X - mX  #centered
  sd <- sqrt(colMeans(X0^2))
  X0 <- X0%*%diag(1/sd)  #standardized
  output <- list(X0=X0, diag=sd)
  return(output)
}

#create 20x100 matrix of ridge coef for each lambda
makeRidgeCoefMatrix <- function(X, y, grid) {
  #stand X and center y
  X0 <- standardizeX(X)$X0
  sd <- standardizeX(X)$diag
  ycent <- y - mean(y)

  #calculate omega with grid of lambdas
  omega <- matrix(0, nr=19, nc=100)
  for(i in 1:100) {
    omega[,i] <- solve(t(X0)%*%X0 + grid[i]*diag(19)) %*% t(X0)%*%ycent
  }

  #recover to beta with original X and y
  beta <- matrix(0, nr=20, nc=100)
  for(i in 1:19) {
    beta[i+1,] <- omega[i,] / sd[i]
  }
  beta[1,] <- mean(y) - t(colMeans(X)) %*% beta[2:20,]

  return(beta)
}

#create matrix
coefMatrix <- makeRidgeCoefMatrix(X,y,grid)


#PROBLEM 2: Part 2----------------------------------------------------------------------

#choose grid of lambda values:
grid <- 10^seq(10,-2,length=100)

#randomly choose 10-fold
set.seed(1)
perm <- sample(263, 263)
#matrix s.t. row i = list of indices of rows in X that are in fold i
fold <- matrix(0, nr=10, nc=27)
for(k in 1:10) {
  #first 7 folds have 26 obs, so last entry is 0
  if(k <= 7) {
    fold[k,] <- c( perm[ (1+26*(k-1)):(26*k) ], 0)
  }
  #last 3 folds have 27 obs
  else {
    fold[k,] <- perm[(183+27*(k-8)):(209+27*(k-8))]
  }
}

#cross validation for each lambda
crossv <- numeric(100)
#for lambda(1) to lambda(100)
for(i in 1:100) {
  #prediction error for each fold
  pe <- numeric(10)
  #from fold 1 to 10
  for(k in 1:10) {
    #create sd vector
    sd <- numeric(19)
    #assign Xt = training set: X without fold k
    #Xv = validation set
    if(k <= 7) {
      #get X0 and ycent for each fold
      X0 <- standardizeX(X[-fold[k,-27], ])$X0
      sd <- standardizeX(X[-fold[k,-27], ])$diag
      ycent <- y[-fold[k,-27]] - mean(y[-fold[k,-27]])
      #fold 1-7 have 26 obs, so remove last element in fold[i,]
      Xt <- X0
      yt <- ycent
      Xv <- X[fold[k,-27], ]
      yv <- y[fold[k,-27]]
    }
    else {
      #get X0 and ycent for each fold
      X0 <- standardizeX(X[-fold[k,], ])$X0
      sd <- standardizeX(X[-fold[k,], ])$diag
      ycent <- y[-fold[k,]] - mean(y[-fold[k,]])
      #fold 8-10 have 27 obs
      Xt <- X0
      yt <- ycent
      Xv <- X[fold[k,], ]
      yv <- y[fold[k,]]
    }

    #perform ridge reg on Xt with lambda(i)
    omegaRidge = solve(t(Xt)%*%Xt + grid[i]*diag(19)) %*% t(Xt)%*%yt

    #recover to beta with original X and y
    betaRidge <- numeric(20)
    for(j in 1:19) {
      betaRidge[j+1] <- omegaRidge[j] / sd[j]
    }
    betaRidge[1] <- mean(yt) - t(colMeans(Xt)) %*% betaRidge[2:20]

    #prediction error of betaRidge on Xv
    pe[k] = mean((yv - betaRidge[1] - Xv%*%betaRidge[2:20])^2)
  }
  #cross valid for lambda(i)
  crossv[i] <- mean(pe)
}

#plot CV versus lambda
plot(log(grid), crossv, main="CV Error vs. Lambda", xlab="log(Lambda)", ylab="CV Error")
lines(log(grid), crossv, type="l")
lambdaBest <- grid[which.min(crossv)]
lambdaBestInd <- which.min(crossv)


#PROBLEM 2: Part 3----------------------------------------------------------------------

#get ridge reg coef with lambdaBest from Part 2 on full data
betaMatrix <- makeRidgeCoefMatrix(X,y,grid)
betaBest <- betaMatrix[,lambdaBestInd]


#PROBLEM 3: Part 1----------------------------------------------------------------------

#package for which.maxn function
install.packages("doBy")
library(doBy)
```

```
#graHTP algorithm: ycent = centered y, X0 = stand X
grahtp <- function(ycent, X0, k, step) {

  #row and col of X0
  n = dim(X0)[1]
  p = dim(X0)[2]
  #initialize variable beta_0 = 0
  bt <- numeric(p)

  repeat {
    #initialize beta_t+1
    bt1 <- numeric(p)
    #calculate gradient of bt
    grad <- -(1/n)*t(X0)%*%(ycent-X0%*%bt)
    #get bt1
    bt1 <- bt - step*grad
    #get indices of largest k elements in bt1
    k_ind <- which.maxn(abs(bt1), k)
    #submatrix with k_ind cols
    Xk <- X0[,k_ind]
    #calculate bt1 with sparsity k
    bt1[k_ind] <- solve(t(Xk)%*%Xk) %*% t(Xk)%*%ycent
    bt1[-k_ind] <- 0

    #stopping condition
    if(norm(bt1-bt, type="2") < 10^-4) break

    #otherwise continue iteration
    bt <- bt1
  }

  #output
  return(bt1)
}


#PROBLEM 3: Part 2-------------------------------------------------------------------------

#reset X and y
df <- read.csv(file="hitters.csv", header=TRUE, row.names=1)
dataMatrix <- data.matrix(df)
X <- dataMatrix[,-19] #263x19 matrix of predictors
y <- matrix(dataMatrix[,19]) #salary col

#standardize X, center y
X0 <- standardizeX(X)$X0
sd <- standardizeX(X)$diag
ycent <- mean(y) - y

#matrix where col(k) = best beta for sparsity k
bestBetas <- matrix(0, nr=20, nc=19)
#store rss for each model Mk
rss <- numeric(19)

#get best omegas, recover to beta
for(k in 1:19) {
  bestOmega <- grahtp(ycent, X0, k, step=.1)

  for(j in 1:19) {
    bestBetas[j+1,k] <- bestOmega[j] / sd[j]
  }
  bestBetas[1,k] <- mean(y) - t(colMeans(X)) %*% bestBetas[2:20,k]

  #calculate rss for each model
  rss[k] <- sum((y - bestBetas[1,k] - X%*%bestBetas[2:20,k])^2)
}

#calculate BIC for all models
bic <- 263*log(rss/263) + log(263)*c(1:19)
bestModelBIC <- min(bic)
#num of variables in the best model
bestModelNum <- which.min(bic)

#best 1 var Model
beta_1var <- bestBetas[,1]
```