MAST90083 Assignment 1

Haonan Zhong

Question 1.1

```
library("MASS")
library("ISLR")
suppressMessages(library("glmnet"))
data(Hitters)
# Remove rows with NA in the Salary column
Hitters <- Hitters[!is.na(Hitters$Salary),]
```

Question 1.2

```
# Construct design matrix and response variable
x <- model.matrix(~.-1, data = subset(Hitters, select = -(Salary)))
y <- Hitters$Salary
lambda <- 10^seq(10, -2, length = 100)

# Estimate ridge coefficients for 100 lambda values
coef_estimate <- glmnet(x, y, alpha = 0, lambda = lambda)
# Observe the coefficents for the largest lambda
coef(coef_estimate)[,1]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  5.359257e+02  5.443467e-08  1.974589e-07  7.956523e-07  3.339178e-07
##           RBI         Walks         Years        CAtBat         CHits
##  3.527222e-07  4.151323e-07  1.697711e-06  4.673743e-09  1.720071e-08
##        CHmRun         CRuns          CRBI        CWalks       LeagueA
##  1.297171e-07  3.450846e-08  3.561348e-08  3.767877e-08  5.800263e-07
##       LeagueN     DivisionW       PutOuts       Assists        Errors
## -5.800262e-07 -7.807263e-06  2.180288e-08  3.561198e-09 -1.660460e-08
##    NewLeagueN
## -1.152288e-07
```

```
#Observe the coefficients for the smallest lambda
coef(coef_estimate)[,100]
```

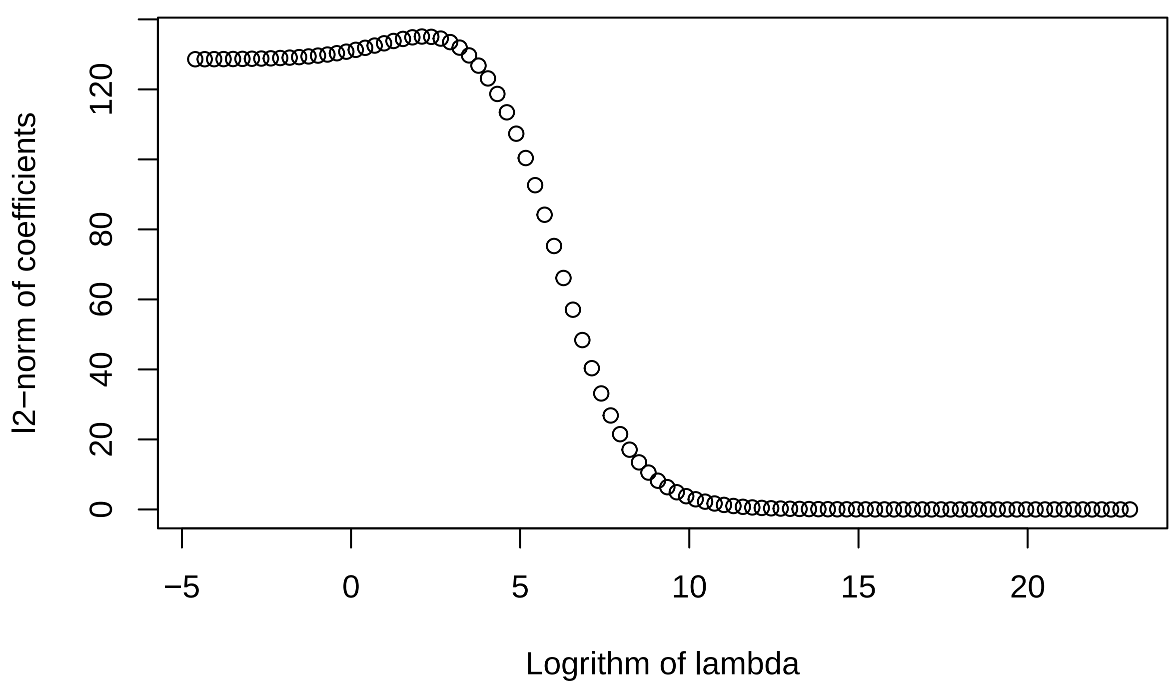
```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  195.91660826   -1.97384734    7.37766330    3.93654704   -2.19869563
##           RBI         Walks         Years        CAtBat         CHits
##   -0.91621489    6.20035808   -3.71378518   -0.17510678    0.21135663
##        CHmRun         CRuns          CRBI        CWalks       LeagueA
```

```
##     0.05633262     1.36604022     0.70963872    -0.79581596   -31.80455226
##        LeagueN      DivisionW        PutOuts        Assists         Errors
##    31.60419354 -117.08236876     0.28202517     0.37318700    -3.42404313
##     NewLeagueN
##   -25.99406318
```

As we can see from the output above, coefficients for the largest lambda is much more closer to 0. Which is quite reasonable, as the effect of shrinkage penalty grows as λ increases, and the ridge regression coefficients will get closer to 0.

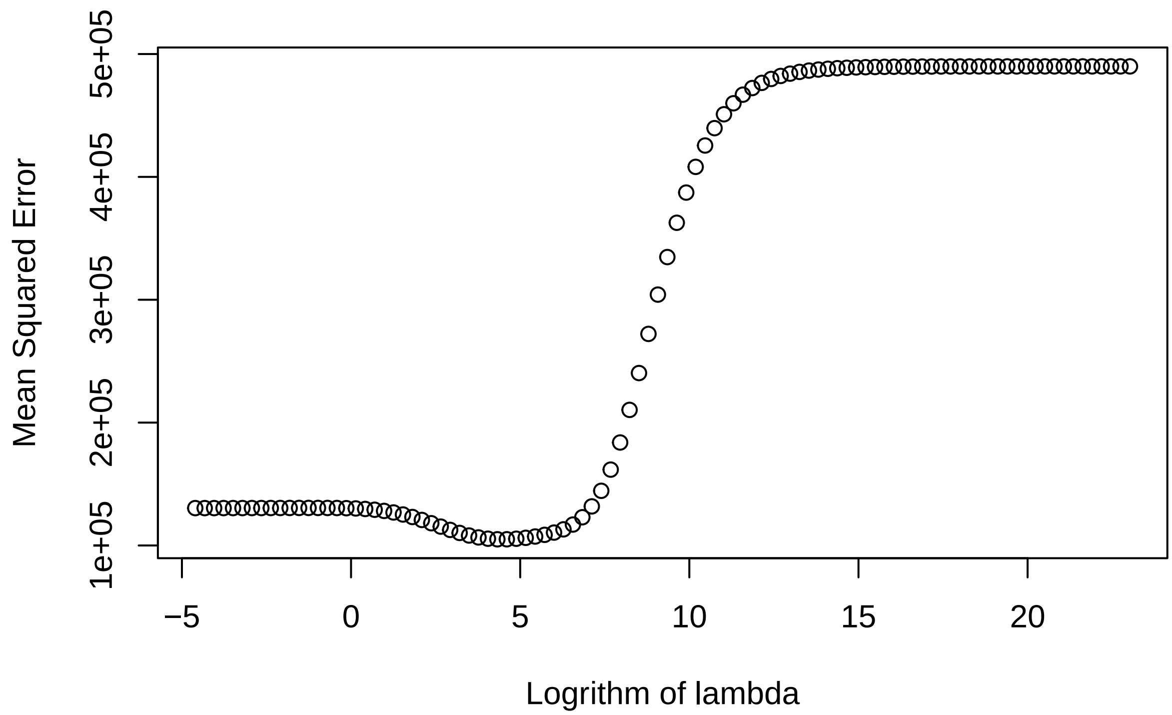
Question 1.3

```r
l2norm <- rep(0, length(lambda))
for (i in 1:100) {
  l2norm[i] <- norm(coef_estimate$beta[,i], type="2")
}
plot(log(lambda), l2norm, xlab = "Logrithm of lambda", ylab = "l2-norm of coefficients")
```




```r
# Computing the MSE for each lambda
mse <- rep(0, length(lambda))
for (i in 1:length(lambda)) {
  prediction <- rep(0, length(y))
  for (j in 1:length(y)) {
```

```
    prediction[j] <- t(x[j,]) %*% matrix(coef_estimate$beta[, i])
  }
  mse[i] <- mean((y - prediction)^2)
}

plot(log(lambda), mse, xlab = "Logrithm of lambda", ylab = "Mean Squared Error")
```



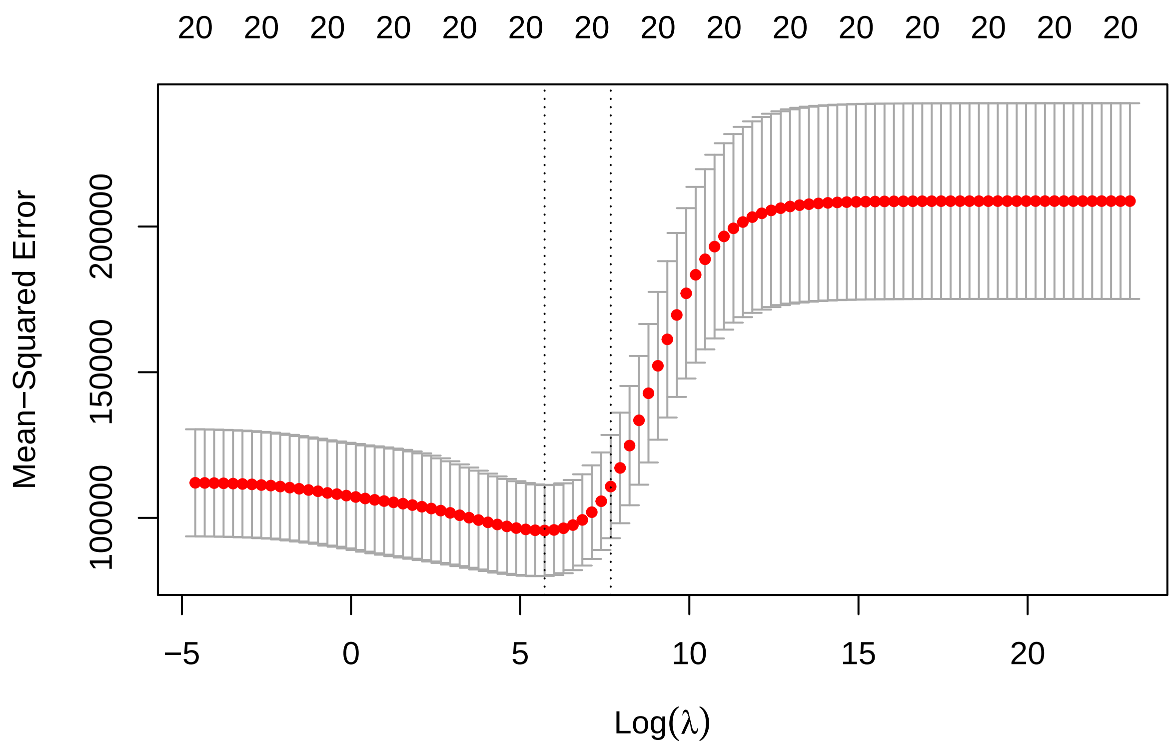
We cannot really say anything about the optimal value of λ with $l2$-norm, since it only tells us the size of the coefficients. On the other hand, mean squared error can tell us how accurate the coefficients are in terms of estimating the our response variable, *Salary*.

Question 1.4

```
# Set the seed equal to 10 for random number generator
set.seed(10)
# Sample the training set
n_train <- sample(seq_len(length(y)), size = 131)
x_train <- x[n_train, ]
y_train <- y[n_train]
# Sample the testing set
x_test <- x[-n_train, ]
y_test <- y[-n_train]

# Performing 10-fold cross validation
(train_cv <- cv.glmnet(x_train, y_train, lambda = lambda, type.measure = "mse", alpha = 0))
```

```
##
## Call:  cv.glmnet(x = x_train, y = y_train, lambda = lambda, type.measure = "mse",      alpha = 0)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min  305.4    63   95669 15586      20
## 1se 2154.4    56  110755 17714      20
```

```
plot(train_cv)
```



As the result above depicted, the model has the lowest mean squared error at 95,669 when $\lambda = 305.4$. Next, we will evaluate the test mean squared error.

```
test_pred <- predict(train_cv$glmnet.fit, train_cv$lambda.min, newx = x_test)
mean((y_test - test_pred)^2)
```

```
## [1] 143265.4
```

The corresponding MSE for $\lambda = 305.4$ on the testing set is 143265.4.

```
# Refit the ridge regression model on the full data set using the lambda chosen by CV
new_model <- glmnet(x, y, alpha = 0, lambda = train_cv$lambda.min)
coef(new_model)
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  26.64638332
## AtBat         0.07272471
## Hits          0.88144658
## HmRun         0.55633590
## Runs          1.07968045
## RBI           0.88278958
## Walks         1.64777011
## Years         1.21365821
## CAtBat        0.01134502
## CHits         0.05845175
## CHmRun        0.41373316
## CRuns         0.11649930
## CRBI          0.12319621
## CWalks        0.04999236
## LeagueA     -15.84232371
## LeagueN      15.84278645
## DivisionW   -80.96578812
## PutOuts       0.16967851
## Assists       0.03076577
## Errors       -1.45580137
## NewLeagueN    4.54951346
```

```
ols_model <- lm(Salary ~ ., data = Hitters)
ols_model$coefficients
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  163.1035878   -1.9798729    7.5007675    4.3308829   -2.3762100   -1.0449620
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    6.2312863   -3.4890543   -0.1713405    0.1339910   -0.1728611    1.4543049
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##    0.8077088   -0.8115709   62.5994230 -116.8492456    0.2818925    0.3710692
##       Errors   NewLeagueN
##   -3.3607605  -24.7623251
```

As we can see from the output above, the coefficients of the ridge regression model are much smaller compare to the one from ordinary least square model. In some cases, the coefficients in ridge regression are shrinked close to zero, but ridge regression still retains all the variables.

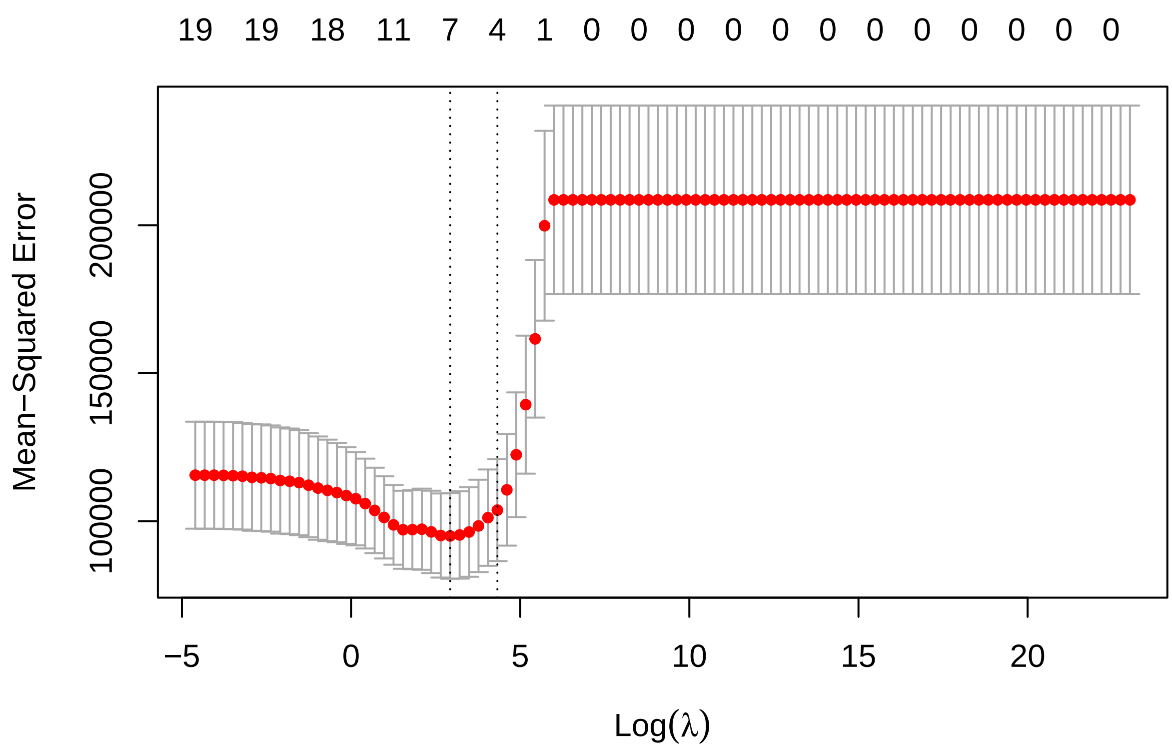
Question 1.5

```
set.seed(10)
(lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1, lambda = lambda, type.measure = "mse"))
```

```
##
## Call:  cv.glmnet(x = x_train, y = y_train, lambda = lambda, type.measure = "mse",      alpha = 1)
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
```

```
## min  18.74    73   95029 14464       7
## 1se  75.65    68  103753 17212       4
```

```r
plot(lasso_cv)
```




```r
lasso_test_pred <- predict(lasso_cv$glmnet.fit, lasso_cv$lambda.min, newx = x_test)
mean((y_test - lasso_test_pred)^2)
```

```
## [1] 142270.1
```

As the output above suggests, the optimal λ value for lasso regression is 18.74, and the corresponding mean squared error for the testing set is 142270.1.

```r
lasso_new_model <- glmnet(x, y, alpha = 1, lambda = lasso_cv$lambda.min)
coef(lasso_new_model)
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                     s0
## (Intercept)  24.6396990
## AtBat         .
## Hits          1.8499834
## HmRun         .
## Runs          .
## RBI           .
```

```
## Walks         2.1959762
## Years         .
## CAtBat        .
## CHits         .
## CHmRun        .
## CRuns         0.2058514
## CRBI          0.4095910
## CWalks        .
## LeagueA       .
## LeagueN       .
## DivisionW   -99.9733911
## PutOuts       0.2158910
## Assists       .
## Errors        .
## NewLeagueN    .
```

Finally, we refitted the lasso regression model using the $\lambda = 18.74$ selected from cross-validation. As we can see, most of the coefficients were shrinked to zero, thus, less important variables are eliminated when penalized, resulted in a sparse model compare to ordinary least square and ridge regression model.

Question 2.1

The number of effective sample size T is $n - p$.

Question 2.2

The model can be consider as $y = X\beta + \eta$, and it can be presented as,

$$\begin{bmatrix} y_{p+1} \\ y_{p+2} \\ . \\ . \\ . \\ y_n \end{bmatrix} = \begin{bmatrix} y_p & y_{p-1} & \cdots & y_1 \\ y_{p+1} & y_p & \cdots & y_2 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ y_{n-1} & y_{n-2} & \cdots & y_{n-p} \end{bmatrix} \times \begin{bmatrix} \phi_1 \\ \phi_2 \\ . \\ . \\ . \\ \phi_p \end{bmatrix}$$

.

Therefore, to obtain the least square estimator, we can apply the normal equation $\hat{\phi} = (X^T X)^{-1} X^T y$, where,

$$X = \begin{bmatrix} y_p & y_{p-1} & \cdots & y_1 \\ y_{p+1} & y_p & \cdots & y_2 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ y_{n-1} & y_{n-2} & \cdots & y_{n-p} \end{bmatrix} \; and \; y = \begin{bmatrix} y_{p+1} \\ y_{p+2} \\ . \\ . \\ . \\ y_n \end{bmatrix}$$

Question 2.3

Given that

$$\sigma_p^2 = \frac{RSS^2}{T} = \frac{||Y - \hat{Y}||^2}{T}$$

Therefore,

$$\sigma_p^2 = \frac{||Y - X\hat{\phi}||^2}{n-p} = \frac{(X\hat{\phi} - Y)^T(X\hat{\phi} - Y)}{n-p} = \frac{Y^TY - Y^TX\hat{\phi} - \hat{\phi}^TX^TY + \hat{\phi}^TX^TX\hat{\phi}}{n-p}$$

where,

$$X = \begin{bmatrix} y_p & y_{p-1} & \cdots & y_1 \\ y_{p+1} & y_p & \cdots & y_2 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ y_{n-1} & y_{n-2} & \cdots & y_{n-p} \end{bmatrix} \; and \; y = \begin{bmatrix} y_{p+1} \\ y_{p+2} \\ . \\ . \\ . \\ y_n \end{bmatrix}$$

Question 2.4

```
# Here we assume the first p samples to be zero
set.seed(10)
n_samples <- 100
M1 <- rep(0, n_samples)
M2 <- rep(0, n_samples)

# Generate samples for model M1
for (i in 6:n_samples){
  M1[i] <- 0.434 * M1[i-1] + 0.217 * M1[i-2] + 0.145 * M1[i-3] + 0.108 * M1[i-4] +
    0.087 * M1[i-5] + rnorm(1)
}

# Generate samples for model M2
for (i in 3:n_samples) {
  M2[i] <- 0.682 * M2[i-1] + 0.346 * M2[i-2] + rnorm(1)
}
```

Question 2.5

```
P <- c(1:10)
# Here we will construct the response variable and the matrix X using the samples
Create_Response <- function(n, p, model) {
  y <- rep(0, n-p)
  for (t in (p+1):n) {
    y[t-p] <- model[t]
  }
  return(y)
}

Create_X <- function(n, p, model) {
  X <- matrix(0, n-p, p)
  for (i in 0:(p-1)) {
    for (j in 1:(n-p)) {
      X[j, p-i] <- model[i+j]
    }
  }
  return(X)
}

Compute_IC1 <- function(sigma2, p, T) {
  return(log(sigma2) + (2 * (p+1)/T))
}

Compute_IC2 <- function(sigma2, p, T) {
  return(log(sigma2) + ((T + p)/(T - p - 2)))
}

Compute_IC3 <- function(sigma2, p, T) {
  return(log(sigma2) + (p * log(T)/T))
}

# Compute the ICs
Compute_IC <- function(model, n, IC1, IC2, IC3) {
  for (p in P) {
    T <- n-p
    tmp_y <- Create_Response(n_samples, p, model)
    tmp_X <- Create_X(n_samples, p, model)
    coefficient <- solve(t(tmp_X) %*% tmp_X) %*% t(tmp_X) %*% matrix(tmp_y)
    pred <- tmp_X %*% coefficient
    sigma2 <- sum((model[c((p+1):n)] - pred)^2)/T

    # Compute the criteria
    IC1[p] <- Compute_IC1(sigma2, p, T)
    IC2[p] <- Compute_IC2(sigma2, p, T)
    IC3[p] <- Compute_IC3(sigma2, p, T)
  }
  return(cbind(IC1, IC2, IC3))
}
```

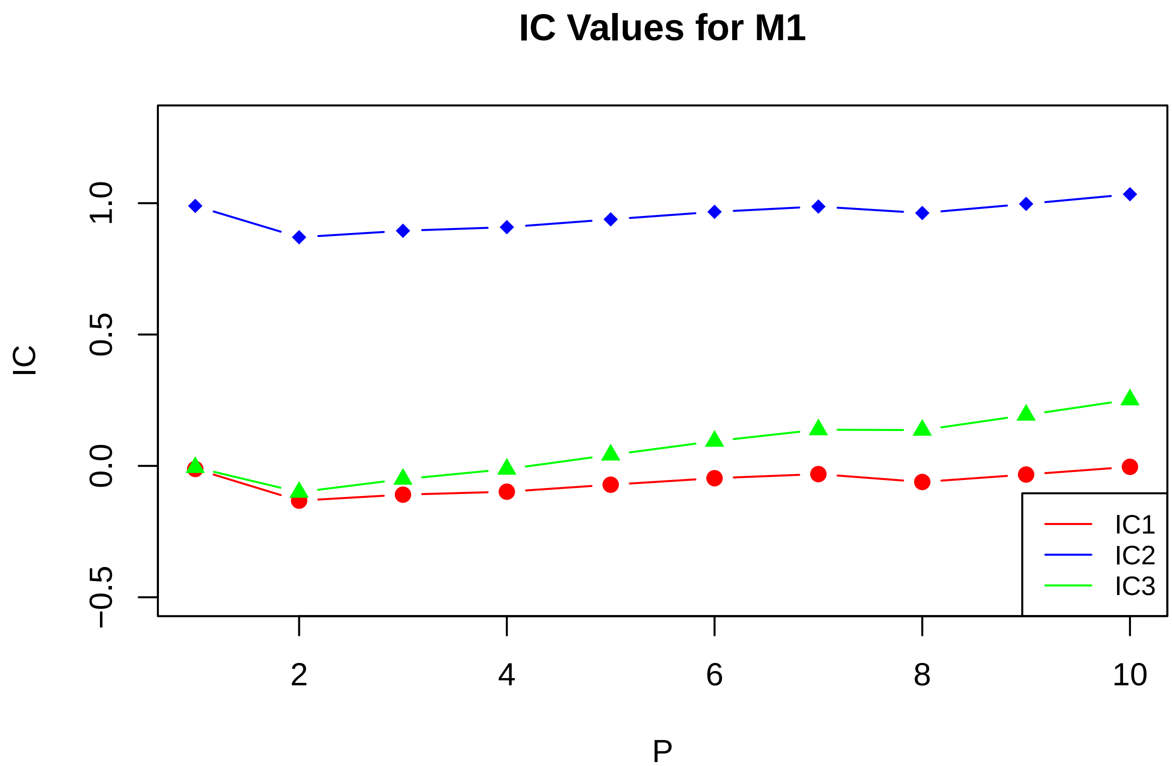
```r
n_samples <- 100
M1_IC1 <- rep(0, 10)
M1_IC2 <- rep(0, 10)
M1_IC3 <- rep(0, 10)
M1_IC <- Compute_IC(M1, n_samples, M1_IC1, M1_IC2, M1_IC3)

M2_IC1 <- rep(0, 10)
M2_IC2 <- rep(0, 10)
M2_IC3 <- rep(0, 10)
M2_IC <- Compute_IC(M2, n_samples, M2_IC1, M2_IC2, M2_IC3)

# Values of IC for model M1
M1_IC
```

```
##               IC1       IC2          IC3
##  [1,] -0.011763558 0.9894991 -0.005752247
##  [2,] -0.132347501 0.8702578 -0.100001226
##  [3,] -0.109525572 0.8949567 -0.050513892
##  [4,] -0.098316908 0.9086275 -0.012302400
##  [5,] -0.071641003 0.9384068  0.041720939
##  [6,] -0.046897938 0.9669566  0.094163431
##  [7,] -0.031286564 0.9871466  0.137833828
##  [8,] -0.061324068 0.9625360  0.136222765
##  [9,] -0.032933141 0.9972866  0.193415601
## [10,] -0.003615629 1.0339912  0.251918778
```

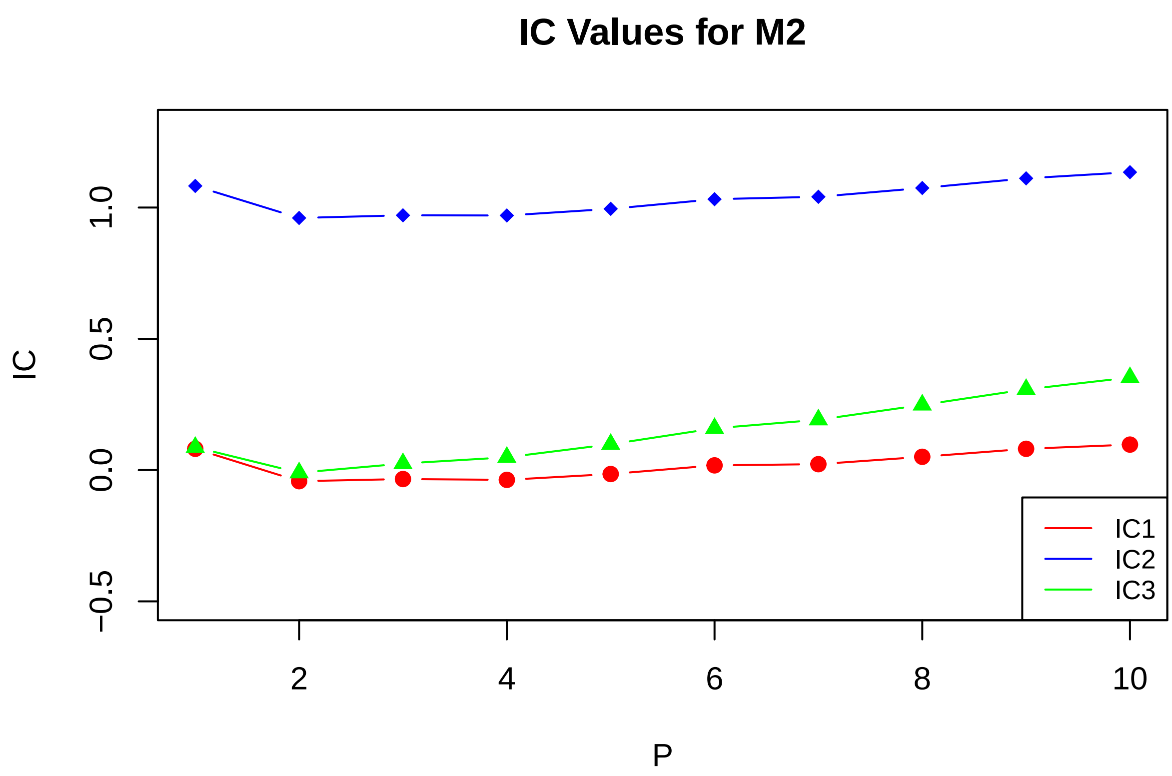
```r
# Values of IC for model M2
M2_IC
```

```
##              IC1       IC2         IC3
##  [1,]  0.08099436 1.0822570  0.08700568
##  [2,] -0.04238235 0.9602229 -0.01003608
##  [3,] -0.03413800 0.9703443  0.02487368
##  [4,] -0.03725311 0.9696913  0.04876140
##  [5,] -0.01505449 0.9949934  0.09830746
##  [6,]  0.01790558 1.0317601  0.15896695
##  [7,]  0.02249788 1.0409311  0.19161827
##  [8,]  0.05056833 1.0744284  0.24811517
##  [9,]  0.08095538 1.1111752  0.30730412
## [10,]  0.09701027 1.1346171  0.35254468
```

```r
# Plotting the three criteria for model M1
plot(P, M1_IC[,1], type = "b", pch = 19, col = 'red', ylim = c(-0.5, 1.3),
     main = "IC Values for M1", ylab = "IC")
lines(P, M1_IC[,2], pch = 18, col = 'blue', type = "b")
lines(P, M1_IC[,3], pch = 17, col = 'green', type = "b")
legend("bottomright", legend=c("IC1", "IC2", "IC3"),
       col=c("red", "blue", "Green"), lty = 1, cex=0.8)
```




```r
# Plotting the three criteria for model M2
plot(P, M2_IC[,1], type = "b", pch = 19, col = 'red', ylim = c(-0.5, 1.3),
     main = "IC Values for M2", ylab = "IC")
lines(P, M2_IC[,2], pch = 18, col = 'blue', type = "b")
lines(P, M2_IC[,3], pch = 17, col = 'green', type = "b")
legend("bottomright", legend=c("IC1", "IC2", "IC3"),
       col=c("red", "blue", "Green"), lty = 1, cex=0.8)
```



Question 2.6

```
set.seed(10)
# Generate 1000 sets of size 100 using model M1
n_samples <- 100
M1_set_100 <- matrix(0, 100, 1000)
for (i in 1:1000) {
  M1 <- rep(0, 100)
  for (j in 6:n_samples) {
    M1[j] <- 0.434 * M1[j-1] + 0.217 * M1[j-2] + 0.145 * M1[j-3] + 0.108 * M1[j-4] +
      0.087 * M1[j-5] + rnorm(1)
  }
  M1_set_100[, i] <- M1
}
```

```
IC_count <- matrix(0, 3, 10)
for (i in 1:1000) {
  IC <- Compute_IC(M1_set_100[, i], n_samples, rep(0, 10), rep(0, 10), rep(0, 10))
  IC_count[1, which(IC[,1] == min(IC[,1]))] <- IC_count[1, which(IC[,1] == min(IC[,1]))] + 1
  IC_count[2, which(IC[,2] == min(IC[,2]))] <- IC_count[2, which(IC[,2] == min(IC[,2]))] + 1
  IC_count[3, which(IC[,3] == min(IC[,3]))] <- IC_count[3, which(IC[,3] == min(IC[,3]))] + 1
}

data.frame(IC_count, row.names = c("IC1", "IC2", "IC3"))
```

```
##      X1  X2  X3  X4 X5 X6 X7 X8 X9 X10
## IC1 15 204 370 180 78 43 34 29 17  30
## IC2 16 232 395 177 78 36 25 19 12  10
## IC3 72 502 335  69 14  5  1  1  1   0
```