Homework 5

Sarah Adilijiang

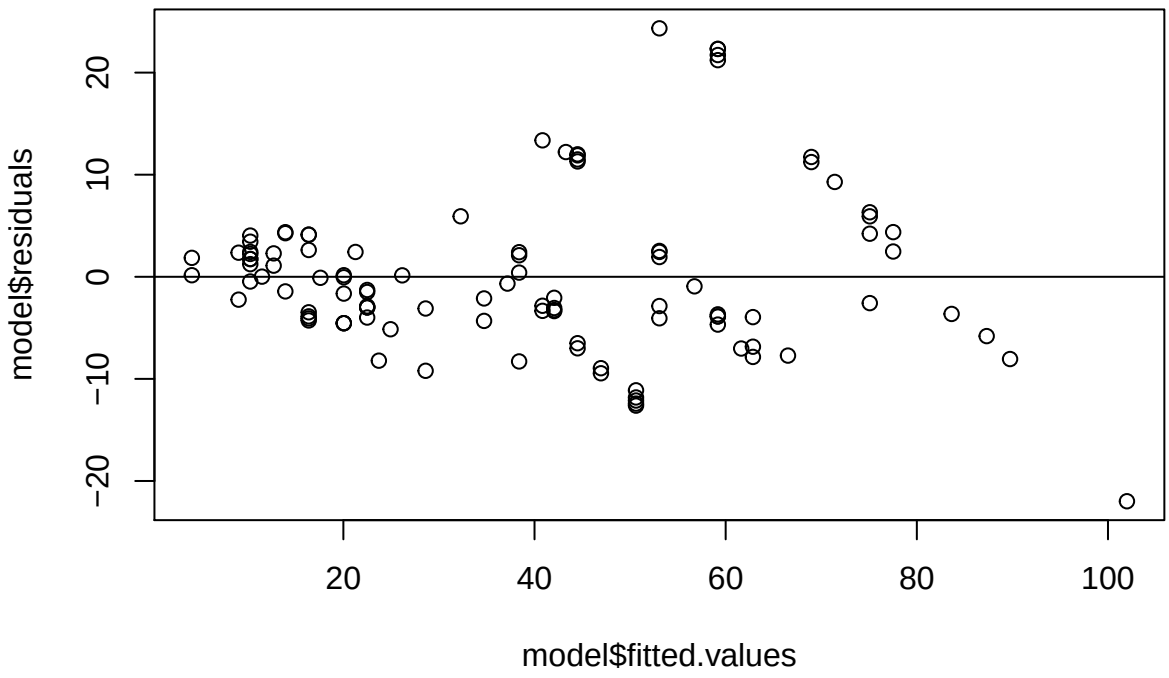
Problem 1

(a) Nonconstant Variance

```
library(faraway)
data(pipeline)
model = lm(Lab~Field, pipeline)
summary(model)
```

```
##
## Call:
## lm(formula = Lab ~ Field, data = pipeline)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.985  -4.072  -1.431   2.504  24.334
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.96750    1.57479  -1.249    0.214
## Field        1.22297    0.04107  29.778   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.865 on 105 degrees of freedom
## Multiple R-squared:  0.8941, Adjusted R-squared:  0.8931
## F-statistic: 886.7 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
# check for nonconstant variance using plots
plot(model$fitted.values, model$residuals, abline(h=0))
```




```
# use a formal test: Breusch-Pagan test to check the heteroscedasticity
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
bptest(model)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model
## BP = 16.045, df = 1, p-value = 6.185e-05
```

Answer:

(1) In the residuals vs fitted values plot, the variances of errors seem to increase for larger fitted values, which indicates nonconstant variance (heteroscedasticity) in the model.

(2) The Breusch-Pagan test's Null Hypothesis is homoscedasticity of the regression model, the Alternative being a heteroscedastic model. Here the Breusch-Pagan test has a p-value = 6.185e-05, so we Reject Null Hypothesis (homoscedasticity). Therefore, there is significant evidence for heteroscedasticity in this model.

(b) WLS

```
i = order(pipeline$Field)
npipe = pipeline[i,]
ff = gl(12,9)[-108]
meanfield = unlist(lapply(split(npipe$Field, ff), mean))
varlab = unlist(lapply(split(npipe$Lab, ff), var))

# regress log(varlab) on log(meanfield) to estimate a0 and a1
model2 = lm(log(varlab)~log(meanfield))
a0 = exp( summary(model2)$coefficients[1,1] ); a0
```

```
## [1] 0.7020209
```

```
a1 = summary(model2)$coefficients[2,1]; a1
```

```
## [1] 1.12442
```

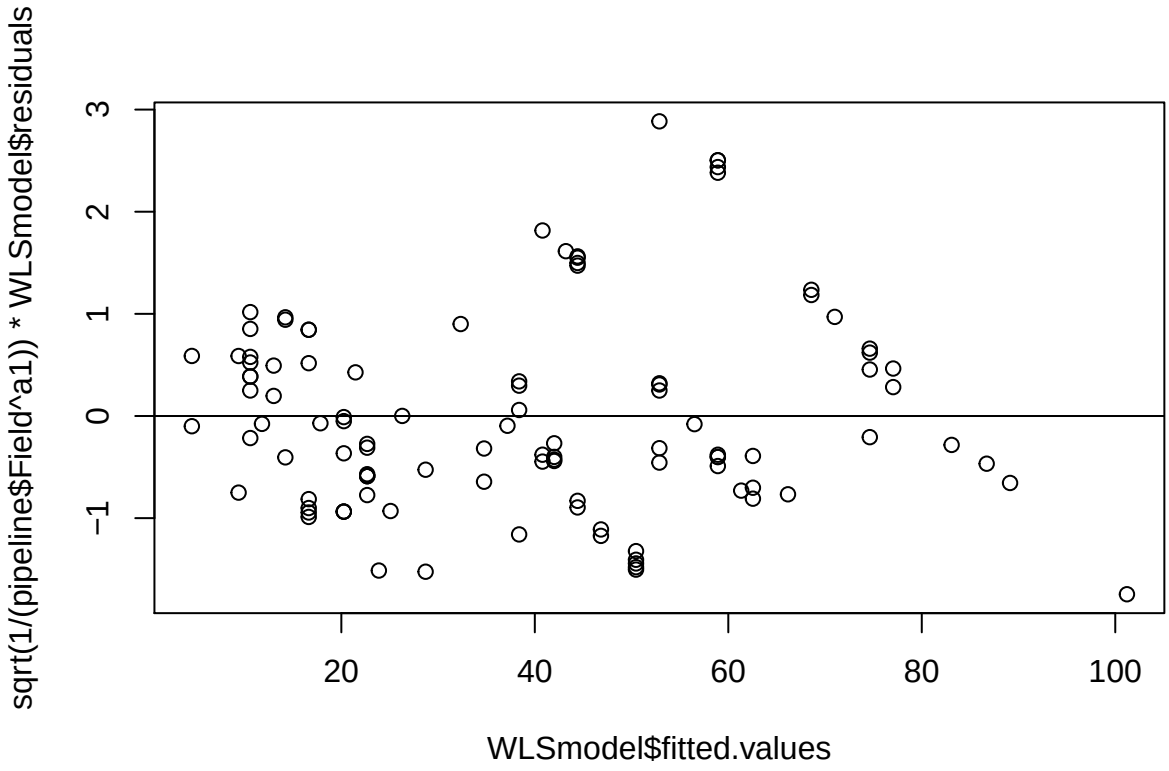
Answer:

Since $var(Lab) = a_o Field^{a_1}$ i.e. $\sigma_i^2 \propto a_o x_i^{a_1} \propto x_i^{a_1}$, so we can set the weights $w_i \propto \frac{1}{\sigma_i^2} \propto \frac{1}{x_i^{a_1}}$.

```
WLSmodel = lm(Lab~Field, pipeline, weights = 1/(Field^a1))
summary(WLSmodel)
```

```
##
## Call:
## lm(formula = Lab ~ Field, data = pipeline, weights = 1/(Field^a1))
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7450 -0.6789 -0.2672  0.5205  2.8847
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.49436    0.90707  -1.647    0.102
## Field        1.20828    0.03488  34.637   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9795 on 105 degrees of freedom
## Multiple R-squared:  0.9195, Adjusted R-squared:  0.9188
## F-statistic:  1200 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
# check for nonconstant variance using plots
plot(WLSmodel$fitted.values, sqrt(1/(pipeline$Field^a1))* WLSmodel$residuals, abline(h=0))
```



Answer:

Here we see the Adjusted R-squared value of the WLS model is higher than that of the previous Least Squares model. And in the residuals vs fitted values plot, the adjusted errors ($\sqrt{w_i}\hat{\varepsilon}_i$) look much better (more like having a constant variance) than the previous LS model.

(c) Transformations

```
cor(pipeline$Lab, pipeline$Field)
```

```
## [1] 0.9455819
```

Since the response "Lab" and the predictor "Field" are highly positively correlated, there is no need to try inverse transformations.

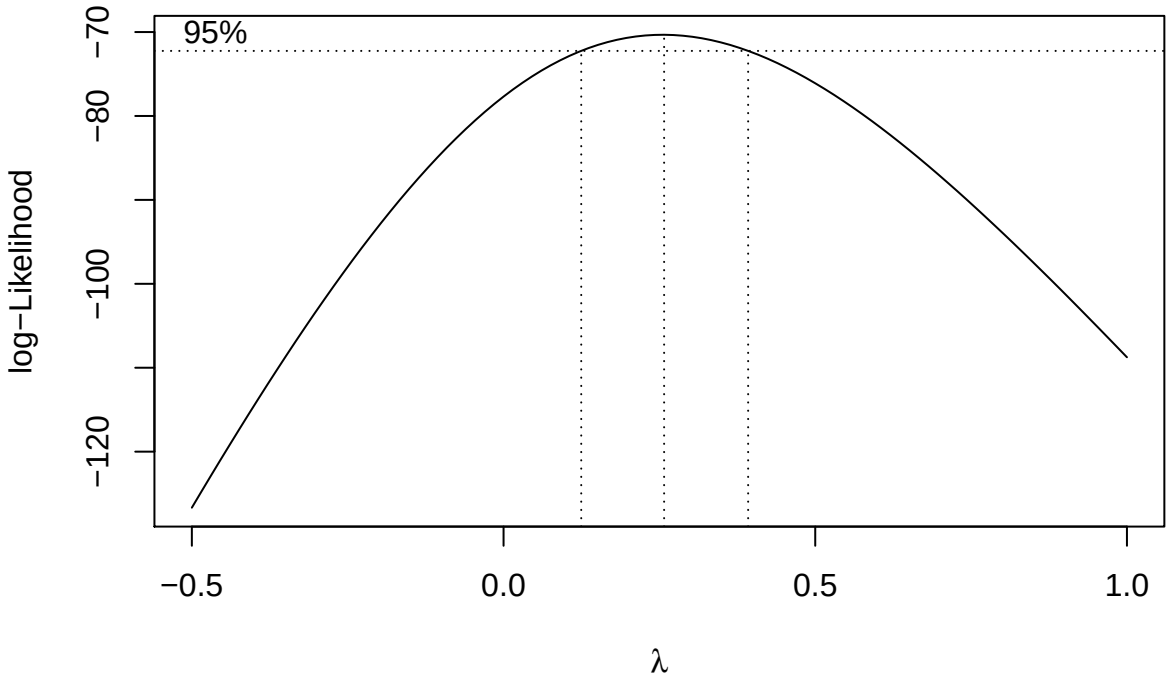
1) Therefore, first, we try to take the square root of the predictor "Field" and look for the appropriate transformation of the predictor "Field" in this case.

```
# fit Lab on sqrt(Field)
model = lm(Lab~sqrt(Field), pipeline)
summary(model)
```

```
##
## Call:
## lm(formula = Lab ~ sqrt(Field), data = pipeline)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.706  -5.843  -1.343   5.538  22.652
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -36.1385     2.8573  -12.65   <2e-16 ***
## sqrt(Field)  13.5486     0.4931   27.48   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.446 on 105 degrees of freedom
## Multiple R-squared:  0.8779, Adjusted R-squared:  0.8767
## F-statistic:   755 on 1 and 105 DF,  p-value: < 2.2e-16
```

```r
# use Box-Cox method to find the appropriate lambda for the response
library(MASS)
boxcox(model, plotit = TRUE, lambda = seq(-0.5,1.0,by=0.1))
```



Answer:

According to the Box-Cox method, the λ should be around 0.25. If we only consider log or square root transformations, then λ can be either 0 or 0.5 in this case, which indicates that we should either take the log or square root of response "Lab".

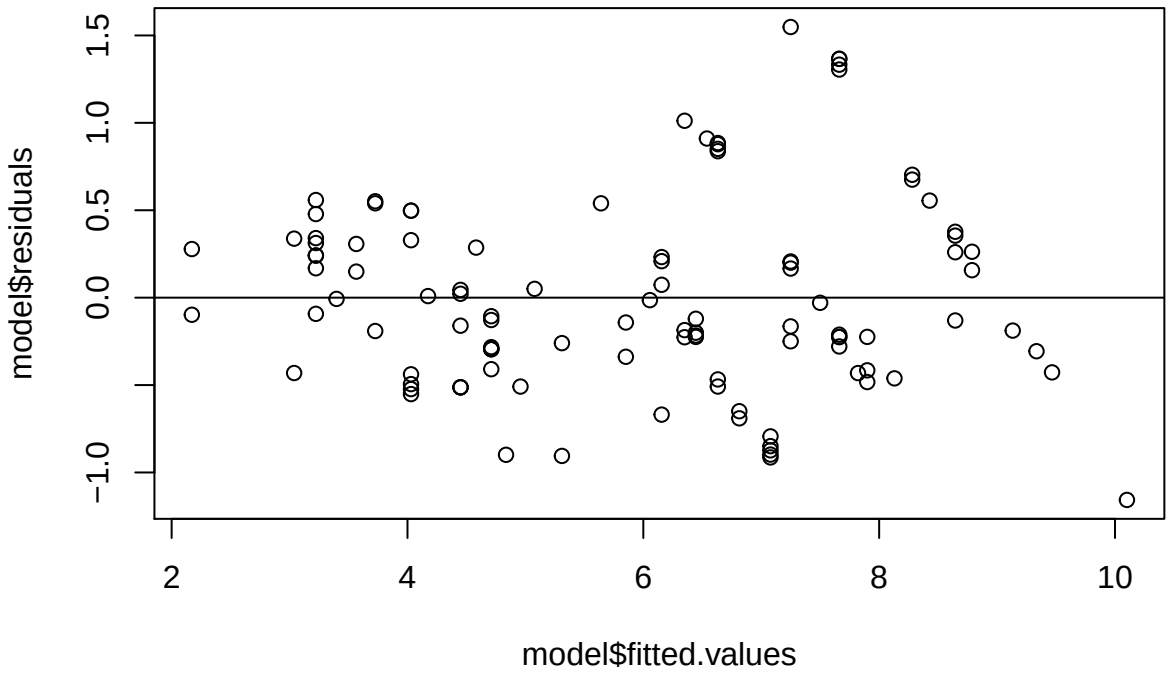
So now we try both of these two models.

```r
# fit sqrt(Lab) on sqrt(Field)
model = lm(sqrt(Lab)~sqrt(Field), pipeline)
summary(model)
```

```
##
```

```
## Call:
## lm(formula = sqrt(Lab) ~ sqrt(Field), data = pipeline)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1570 -0.4125 -0.1209  0.3098  1.5481
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.36773    0.18815  -1.954   0.0533 .
## sqrt(Field)  1.13553    0.03247  34.973   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5561 on 105 degrees of freedom
## Multiple R-squared:  0.9209, Adjusted R-squared:  0.9202
## F-statistic:  1223 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
# check for nonconstant variance using plots
plot(model$fitted.values, model$residuals, abline(h=0))
```




```
# use a formal test: Breusch-Pagan test to check the heteroscedasticity
library(lmtest)
bptest(model)
```

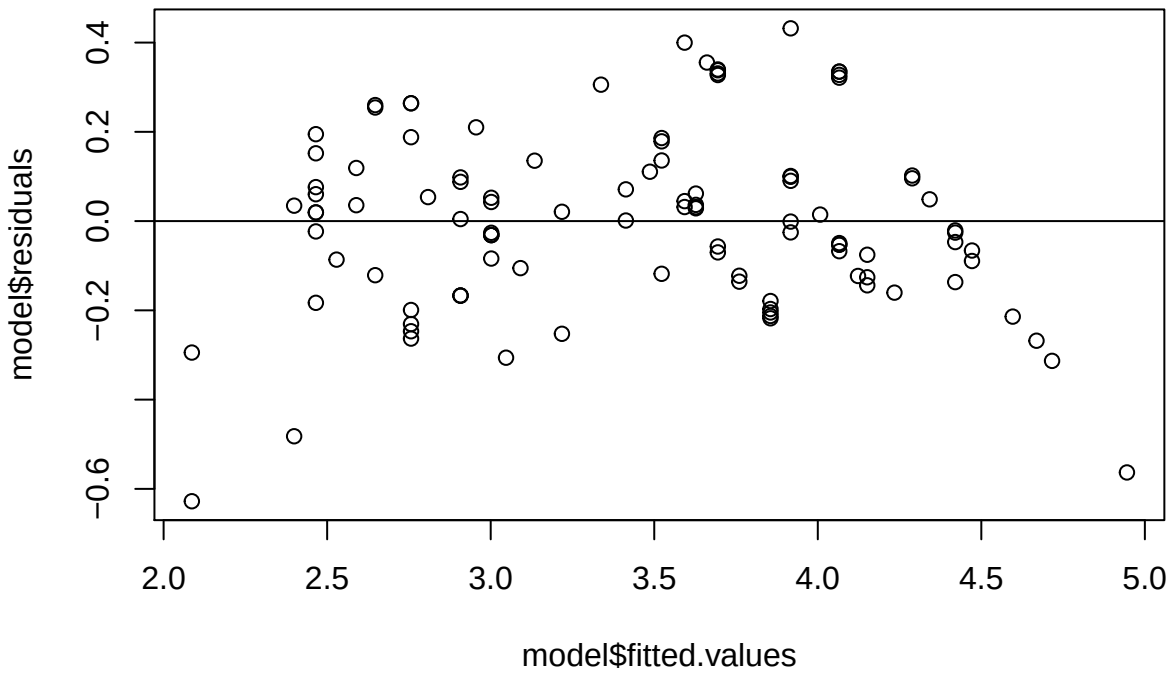
```
##
##  studentized Breusch-Pagan test
```

```
##
## data:  model
## BP = 8.3055, df = 1, p-value = 0.003952
```

```
# fit log(Lab) on sqrt(Field)
model = lm(log(Lab)~sqrt(Field), pipeline)
summary(model)
```

```
##
## Call:
## lm(formula = log(Lab) ~ sqrt(Field), data = pipeline)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.62771 -0.12443  0.00125  0.10157  0.43186
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.17093    0.06824   17.16   <2e-16 ***
## sqrt(Field)  0.40938    0.01178   34.76   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2017 on 105 degrees of freedom
## Multiple R-squared:  0.9201, Adjusted R-squared:  0.9193
## F-statistic:  1208 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
# check for nonconstant variance using plots
plot(model$fitted.values, model$residuals, abline(h=0))
```




```
# use a formal test: Breusch-Pagan test to check the heteroscedasticity
library(lmtest)
bptest(model)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model
## BP = 0.02032, df = 1, p-value = 0.8866
```

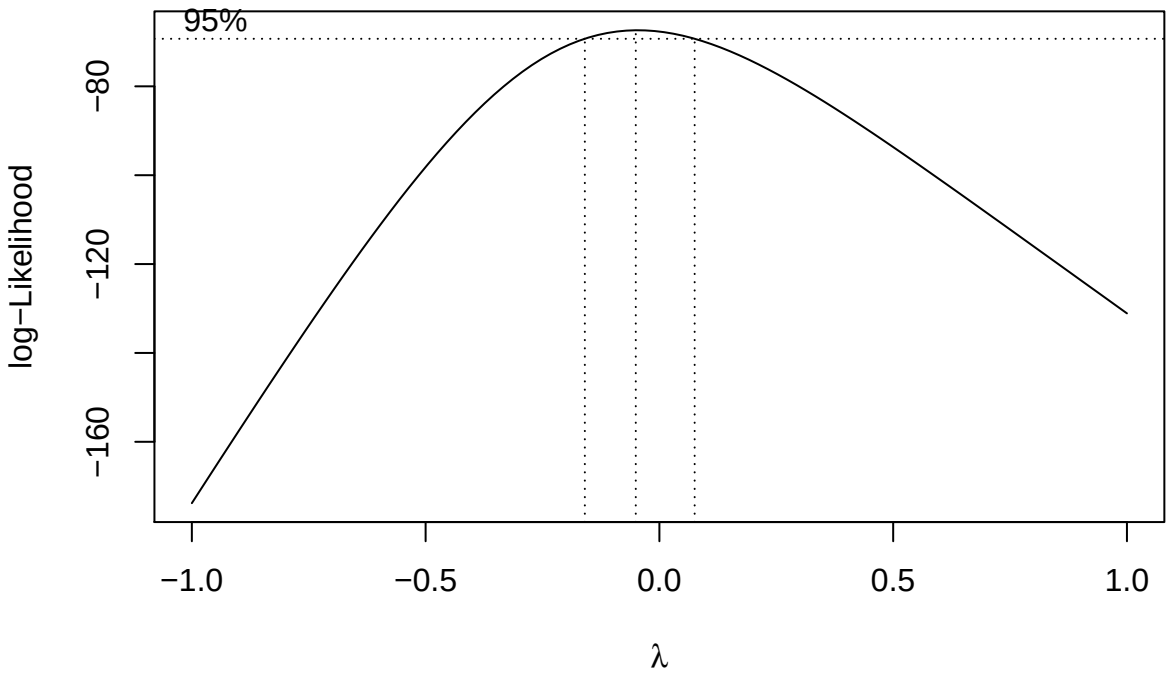
2) Second, we try to take the log of the predictor "Field" and look for the appropriate transformation of the predictor "Field" in this case.

```
# fit Lab on log(Field)
model = lm(Lab~log(Field), pipeline)
summary(model)
```

```
##
## Call:
## lm(formula = Lab ~ log(Field), data = pipeline)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.984  -8.435  -3.022   7.089  24.342
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -72.068      5.272  -13.67   <2e-16 ***
```

```
## log(Field)    33.382      1.554   21.48   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.41 on 105 degrees of freedom
## Multiple R-squared:  0.8146, Adjusted R-squared:  0.8129
## F-statistic: 461.4 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
# use Box-Cox method to find the appropriate lambda for the response
library(MASS)
boxcox(model, plotit = TRUE, lambda = seq(-1.0,1.0,by=0.1))
```



Answer:

According to the Box-Cox method, the λ should be around -0.9. If we only consider log or square root transformations, then λ should be 0 in this case, which indicates that we should either take the log of response "Lab".

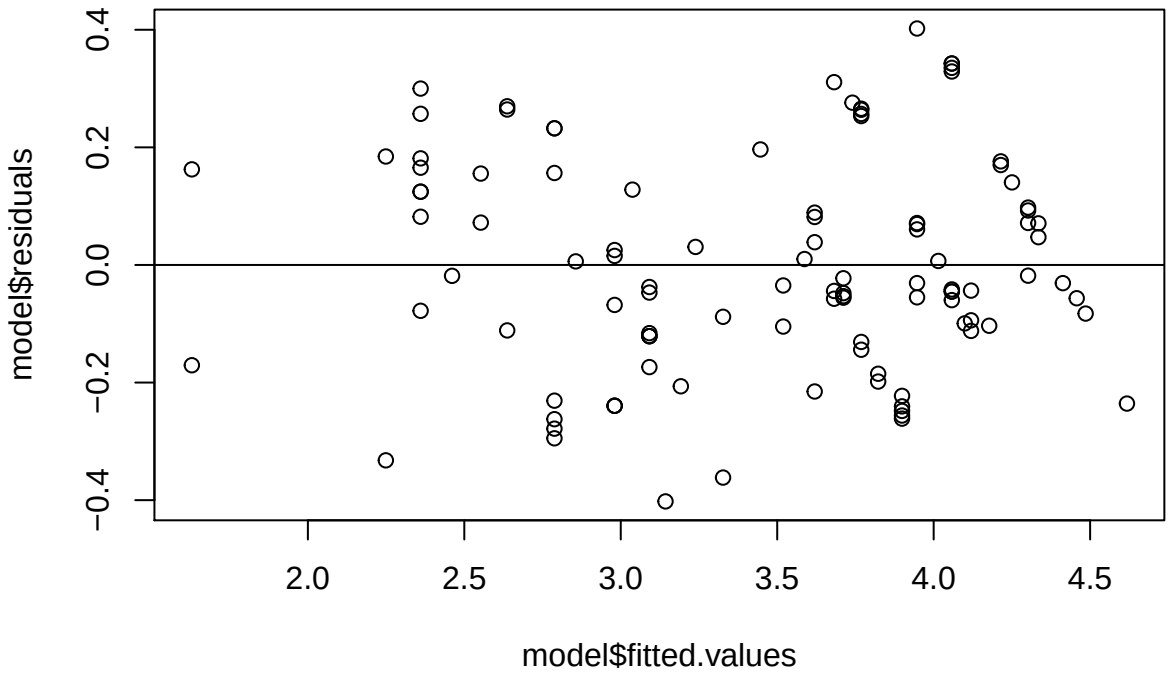
So now we try this model.

```
# fit log(Lab) on log(Field)
model = lm(log(Lab)~log(Field), pipeline)
summary(model)
```

```
##
## Call:
## lm(formula = log(Lab) ~ log(Field), data = pipeline)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -0.40212 -0.11853 -0.03092  0.13424  0.40209
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.06849    0.09305  -0.736    0.463
## log(Field)   1.05483    0.02743  38.457   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1837 on 105 degrees of freedom
## Multiple R-squared:  0.9337, Adjusted R-squared:  0.9331
## F-statistic:  1479 on 1 and 105 DF,  p-value: < 2.2e-16
```

```
# check for nonconstant variance using plots
plot(model$fitted.values, model$residuals, abline(h=0))
```




```
# use a formal test: Breusch-Pagan test to check the heteroscedasticity
library(lmtest)
bptest(model)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model
## BP = 1.1252, df = 1, p-value = 0.2888
```

Answer:

Compare the above three models:

(1) The first model (regressing $\sqrt{Lab}$ on $\sqrt{Field}$) has a Breusch-Pagan test p-value of 0.003952, so we Reject Null Hypothesis (homoscedasticity). Therefore, there is still a significant evidence for heteroscedasticity in this model.

(2) The second and third models both have Breusch-Pagan test p-value that are larger than 0.1, so there are no significant evidences for heteroscedasticity in both of these two models. However, the third model (regressing $\log(Lab)$ on $\log(Field)$) has higher Multiple R-squared value and higher Adjusted R-squared value than the second model (regressing $\log(Lab)$ on $\sqrt{Field}$). Also, in the residuals vs fitted values plot, the third model has better distribution of residuals which looks more like having a constant variance.

Therefore, in conclusion, we should regress $\log(Lab)$ on $\log(Field)$ to get an approximately linear relationship with constant variance.

Problem 2

(a) Least Squares

```
library(faraway)
data(stackloss)
model_ls = lm(stack.loss~., stackloss)
summary(model_ls)
```

```
##
## Call:
## lm(formula = stack.loss ~ ., data = stackloss)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.2377 -1.7117 -0.4551  2.3614  5.6978
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -39.9197    11.8960  -3.356  0.00375 **
## Air.Flow      0.7156     0.1349   5.307  5.8e-05 ***
## Water.Temp    1.2953     0.3680   3.520  0.00263 **
## Acid.Conc.   -0.1521     0.1563  -0.973  0.34405
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.243 on 17 degrees of freedom
## Multiple R-squared:  0.9136, Adjusted R-squared:  0.8983
## F-statistic:  59.9 on 3 and 17 DF,  p-value: 3.016e-09
```

(b) Least Absolute Deviations (LAD)

```
library(quantreg)
```

```
## Loading required package: SparseM
```

```
##
## Attaching package: 'SparseM'
```

```
## The following object is masked from 'package:base':
##
##     backsolve
```

```r
model_lad = rq(stack.loss~Air.Flow+Water.Temp+Acid.Conc., data = stackloss)
summary(model_lad)
```

```
##
## Call: rq(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
##     data = stackloss)
##
## tau: [1] 0.5
##
## Coefficients:
##             coefficients lower bd  upper bd
## (Intercept) -39.68986    -41.61973 -29.67754
## Air.Flow      0.83188      0.51278   1.14117
## Water.Temp    0.57391      0.32182   1.41090
## Acid.Conc.   -0.06087     -0.21348  -0.02891
```

(c) Huber method

```r
library(MASS)
model_huber = rlm(stack.loss~Air.Flow+Water.Temp+Acid.Conc., data = stackloss)
summary(model_huber)
```

```
##
## Call: rlm(formula = stack.loss ~ Air.Flow + Water.Temp + Acid.Conc.,
##     data = stackloss)
## Residuals:
##      Min       1Q   Median       3Q      Max
## -8.91753 -1.73127  0.06187  1.54306  6.50163
##
## Coefficients:
##             Value    Std. Error t value
## (Intercept) -41.0265   9.8073    -4.1832
## Air.Flow      0.8294   0.1112     7.4597
## Water.Temp    0.9261   0.3034     3.0524
## Acid.Conc.   -0.1278   0.1289    -0.9922
##
## Residual standard error: 2.441 on 17 degrees of freedom
```

(d) Least Trimmed Squares (LTS)

```r
library(MASS)
model_lts = ltsreg(stack.loss~Air.Flow+Water.Temp+Acid.Conc., data = stackloss)
summary(model_lts)
```

```
##               Length Class      Mode
## crit           1     -none-     numeric
## sing           1     -none-     character
## coefficients   4     -none-     numeric
## bestone        4     -none-     numeric
## fitted.values 21     -none-     numeric
## residuals     21     -none-     numeric
## scale          2     -none-     numeric
## terms          3     terms      call
## call           4     -none-     call
## xlevels        0     -none-     list
```

```
## model          4     data.frame list
```

```
coef(model_lts)
```

```
##  (Intercept)     Air.Flow   Water.Temp   Acid.Conc.
## -33.50000000   0.75000000   0.35483871  -0.03225806
```

Answer:

Compare the four models:

When using LAD/Huber/LTS models, the numerical values of the coefficient for predictor "Air.Flow" change a relatively small amount comparing with the LS model. However, the numerical values of predictors "Water.Temp" and "Acid.Conc." change a relatively larger amount comparing with the LS model.

The LS works well when there are normal errors, but perform relatively poorly when having outliers and highly influential points. However, the robust regression methods (LAD/Huber/LTS) are less sensitive thus more robust to outliers.

(e) Outliers and Influential points

Outliers

```
# find potential outliers
jack <- rstudent(model_ls)
jack[which.max(abs(jack))]
```

```
##        21
## -3.330493
```

```
# Here we use 5% significance level to perform the t-test
alpha = 0.05
n = nrow(stackloss)
p = length(model_ls$coefficients)

# t-test without Bonferroni correction
t = qt(1-alpha/2, df = n-p-1)
jack[abs(jack) > t]
```

```
##        21
## -3.330493
```

```
# t-test with Bonferroni correction
t = qt(1-(alpha/2)/n, df = n-p-1)
jack[abs(jack) > t]
```

```
## named numeric(0)
```

```
# outlier test
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following objects are masked from 'package:faraway':
##
##     logit, vif
```

```r
outlierTest(model_ls)
```

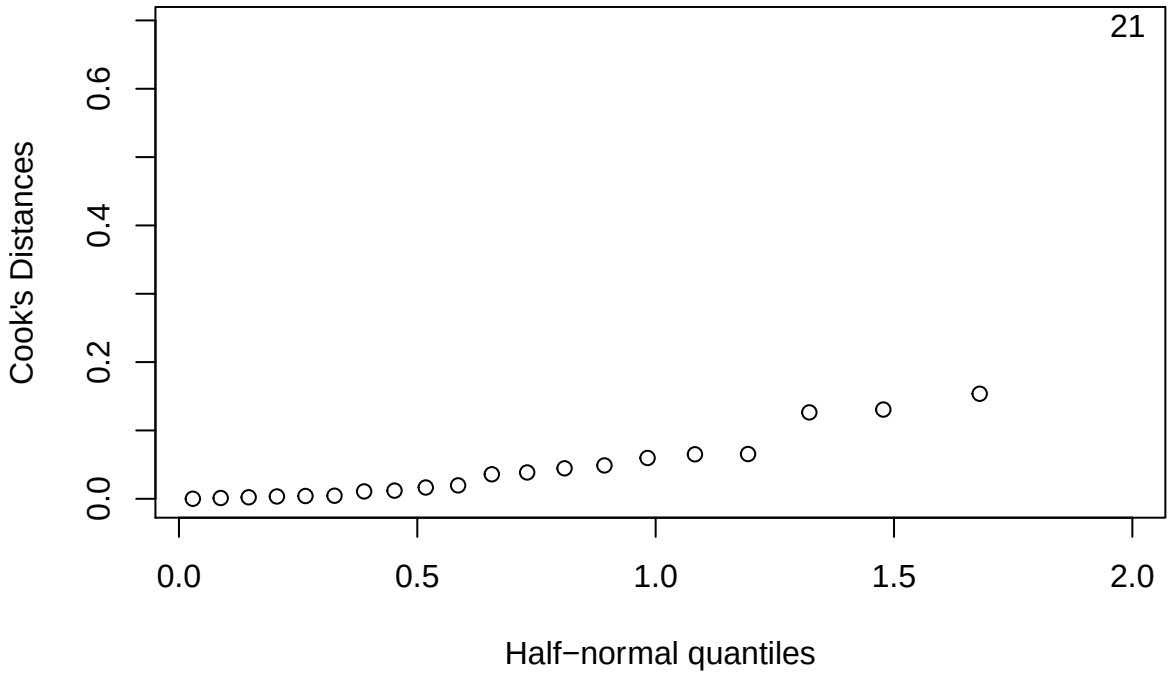
```
## No Studentized residuals with Bonferonni p < 0.05
## Largest |rstudent|:
##     rstudent unadjusted p-value Bonferonni p
## 21 -3.330493           0.004238     0.088999
```

Influential points

```r
# find influential points with large Cook's Distance
cook = cooks.distance(model_ls)
n = nrow(stackloss)
cook[cook > 4/n]
```

```
##        21
## 0.6919999
```

```r
# find influential points with large Cook's Distance via half-normal plot
halfnorm(cook, ylab = "Cook's Distances", nlab = 1)
```



Answer:

The 21th observation (row) seem to be an outlier to the least squares regression model under the looser measure without Bonferroni correction.

And the 21th observation (row) also have a large Cook's Distance thus have a large influence on the fitted least squares model.

Therefore, we remove the 21th observation (row) to re-fit the least squares regression.

```r
stackloss_new = stackloss[-21,]
model_ls_new = lm(stack.loss~., stackloss_new)
summary(model_ls_new)
```

```
##
## Call:
## lm(formula = stack.loss ~ ., data = stackloss_new)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0449 -2.0578  0.1025  1.0709  6.3017
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -43.7040     9.4916  -4.605 0.000293 ***
## Air.Flow      0.8891     0.1188   7.481 1.31e-06 ***
## Water.Temp    0.8166     0.3250   2.512 0.023088 *
## Acid.Conc.   -0.1071     0.1245  -0.860 0.402338
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.569 on 16 degrees of freedom
## Multiple R-squared:  0.9488, Adjusted R-squared:  0.9392
## F-statistic: 98.82 on 3 and 16 DF,  p-value: 1.541e-10
```

Answer:

Compare the new LS model with previous four models:

In the new LS model, the the numerical values of the coefficient for predictor "Air.Flow" still change a relatively small amount comparing with previous four models. However, the numerical values of predictors "Water.Temp" and "Acid.Conc." change a relatively larger amount comparing with the previous LS model which includes the ourlier, and now their values fall in the range of these two coefficient values in LAD/Huber/LTS models.

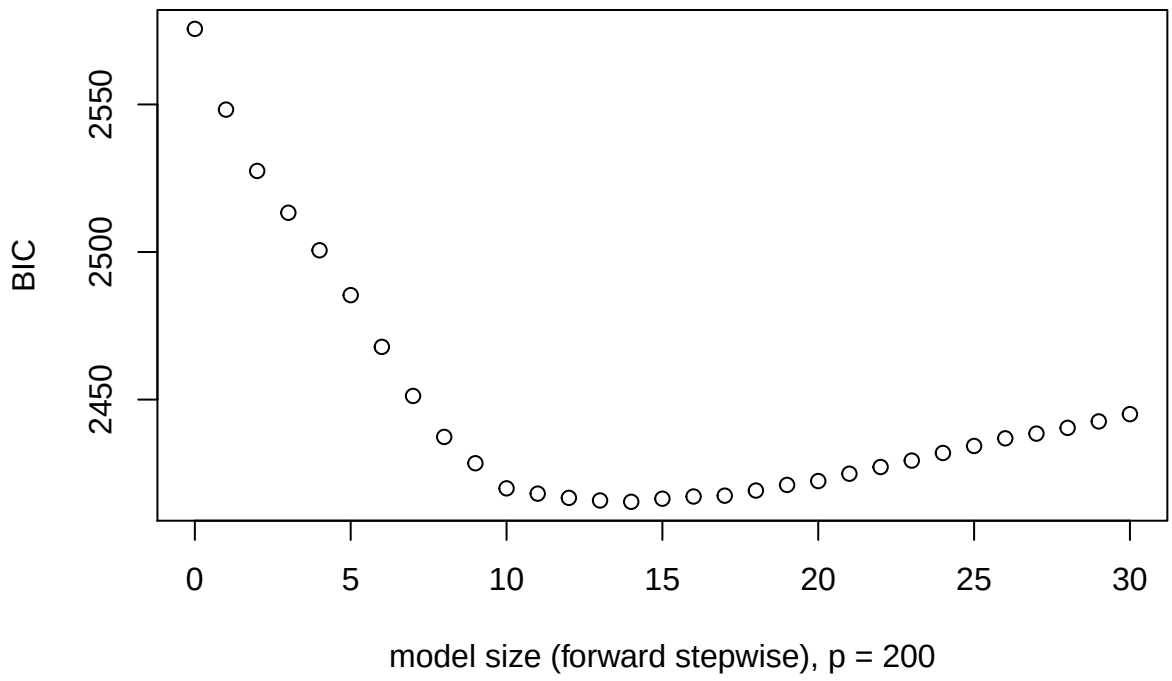
Therefore, the LS works well when there are normal errors, but perform relatively poorly when having outliers and highly influential points. And performing LS model after removing the outliers has similar effect of robust regression methods (LAD/Huber/LTS) which are less sensitive thus more robust to outliers.

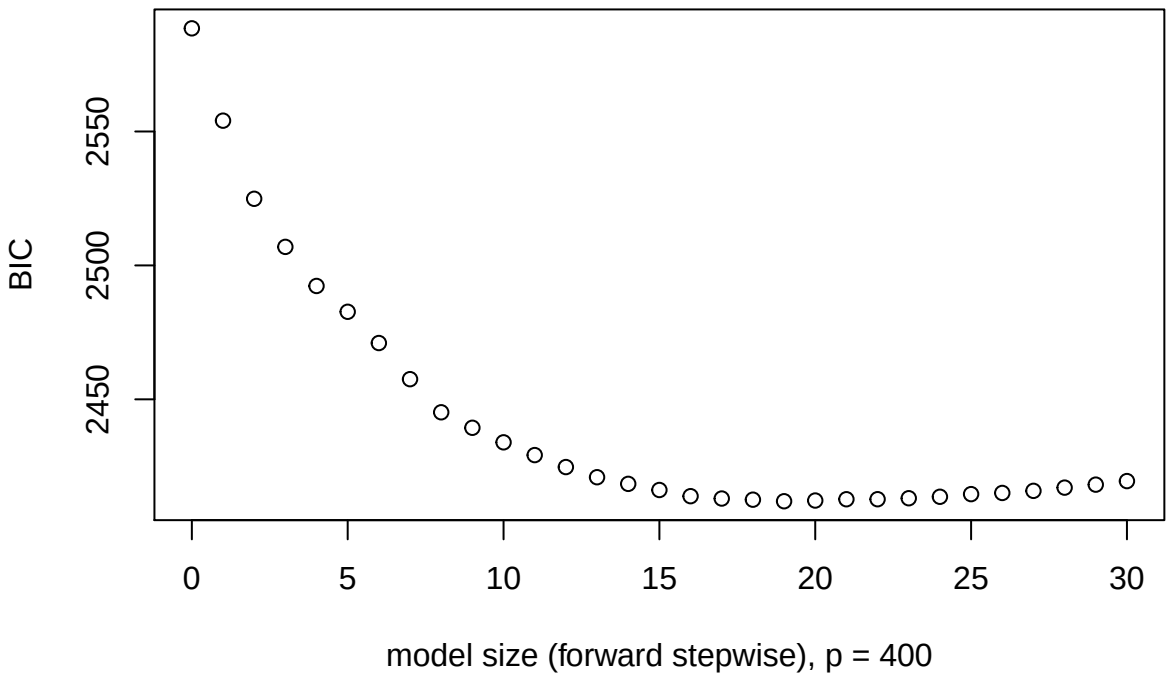
Problem 3

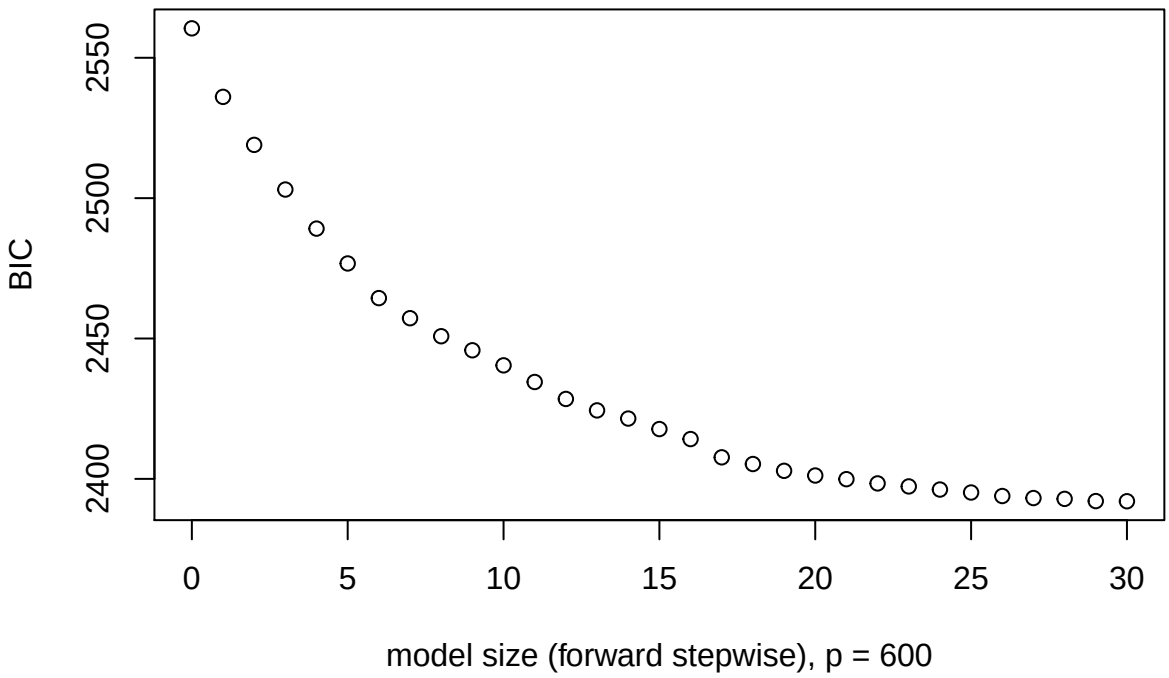
(a) Forward stepwise - BIC

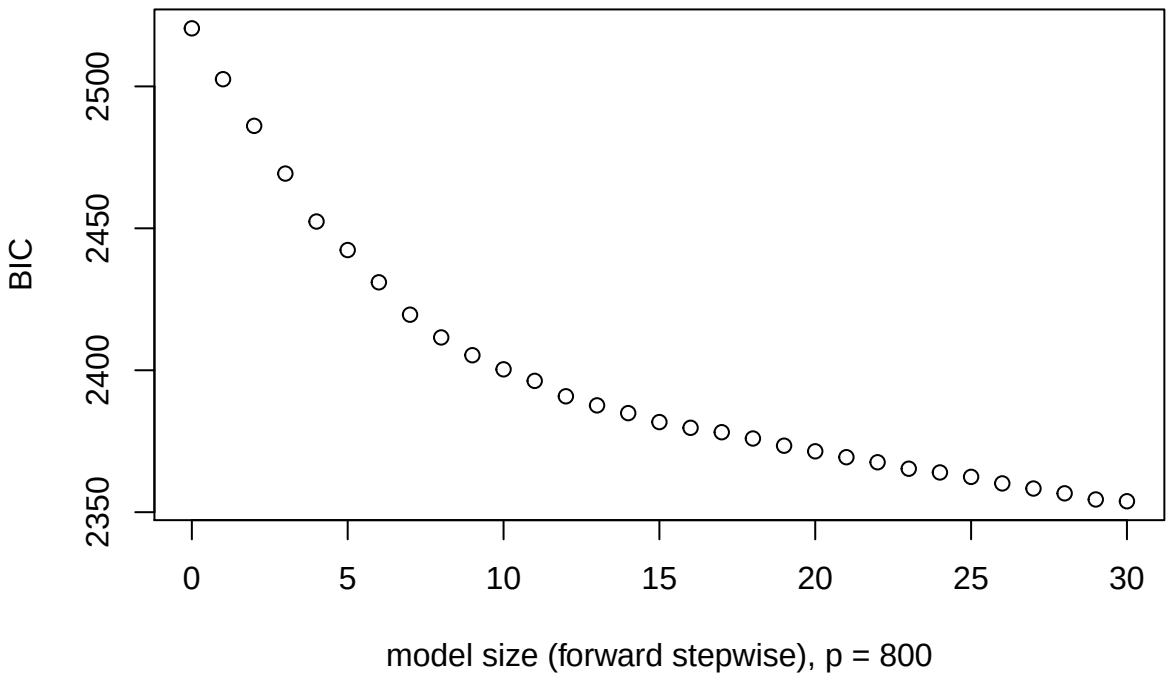
```r
set.seed(1)
p = c(200,400,600,800)

for (i in 1:length(p)){
    # generate simulated data set
    n = 400
    X = matrix(rnorm(n*p[i], mean=0, sd=1), nrow=n, ncol=p[i])
    X = X %*% diag( 1/sqrt(colSums(X^2)) )  #or X = scale(X, center=FALSE, scale=sqrt(colSums(X^2)))
    Beta = c(rep(5,10),rep(0,p[i]-10))
    noise = rnorm(n, mean=0, sd=1)
    Y = X %*% Beta + noise

    # run forward stepwise without threshold of p-values
    S = NULL
```

```
    store_RSS = rep(0,31)
    store_RSS[1] = sum( (Y-lm(Y~1)$fitted.values)^2 )
    pvalues = NULL

    for (k in 1:30){
        S_else = setdiff(1:p[i],S)

        for (j in 1:length(S_else)){
            model = lm( Y ~ X[,c(S,S_else[j])] )
            pvalues[j] = summary(model)$coefficients[nrow(summary(model)$coefficients),4]
        }
        add_ind = S_else[which.min(pvalues)]
        S = c(S,add_ind)
        XS = X[,S,drop=FALSE]
        store_RSS[k+1] = sum( (Y-lm(Y~XS)$fitted.values)^2 )
    }

    BIC = n*log(store_RSS) + (0:30)*log(n)
    plot(0:30,BIC,xlab=paste0('model size (forward stepwise), p = ',p[i]),ylab='BIC')
}
```




BIC
2450
2500
2550
0
5
10
15
20
25
30
model size (forward stepwise), p = 400


BIC
2400
2450
2500
2550
0
5
10
15
20
25
30
model size (forward stepwise), p = 600



Answer:

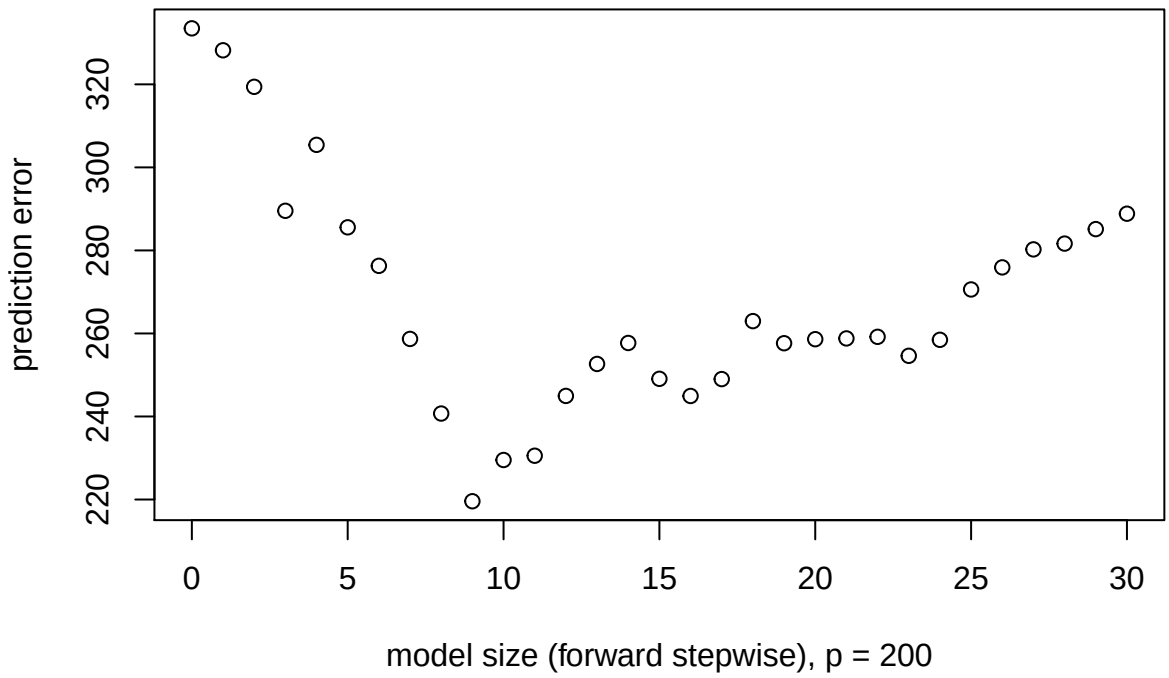
When p=200 < n=400, BIC has the lowest value around model size 14, which is closer to the true model size 10 though still having some false positive results. When p=400 = n=400, BIC has the lowest value around model size 19, which is more larger than true model size 10 and having more false positive results than before. Therefore, when p is smaller, BIC does a relatively good job of picking an appropriate model size, but gets more false positive results as p increases since BIC don't correct for multiple testing issue.

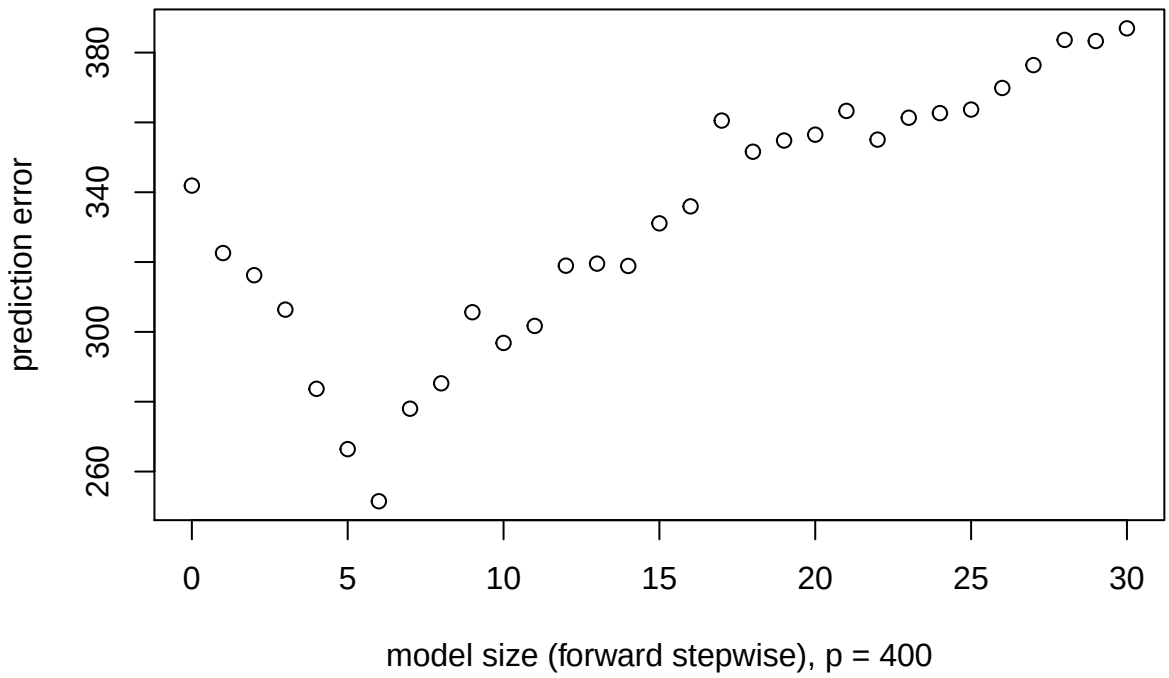
When p=600 or 800 > n=400, BIC keeps decreasing and has the lowest value at the model size 30 and can get even lower if keeps increasing the model size. In this case, BIC does not do a good job of picking an appropriate model size any more, since BIC don't correct for multiple tesing issue and having a large number of covariates will generate lots of false positive results for BIC.

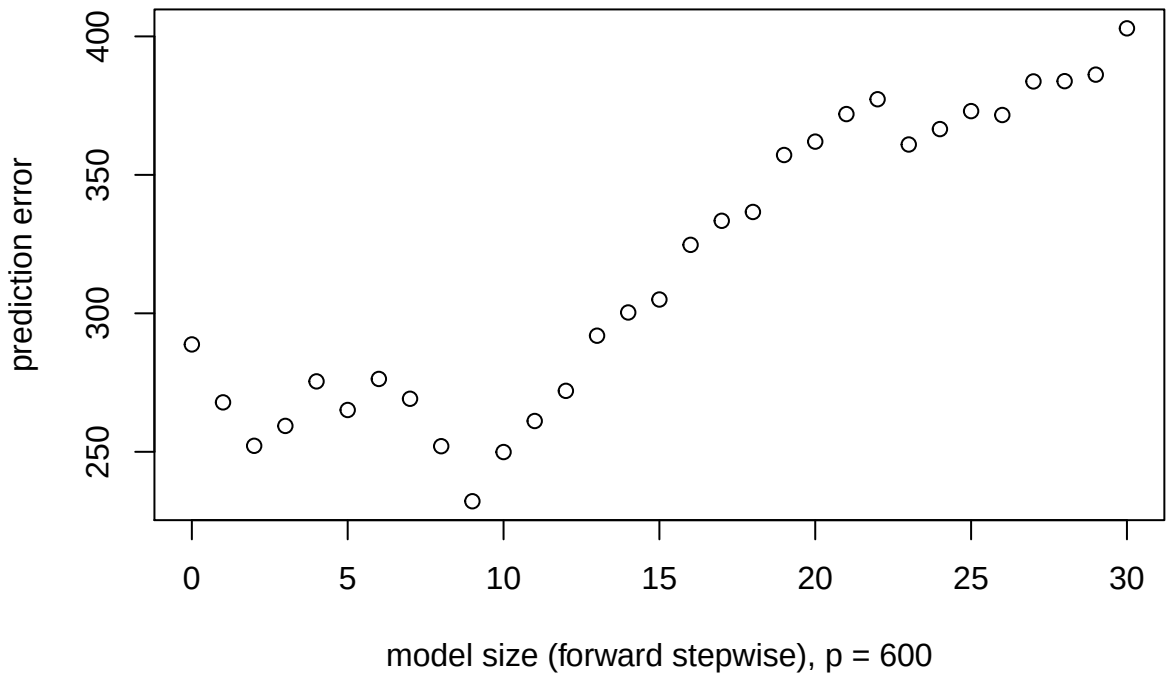
(b) Forward stepwise - prediction error in the validation set

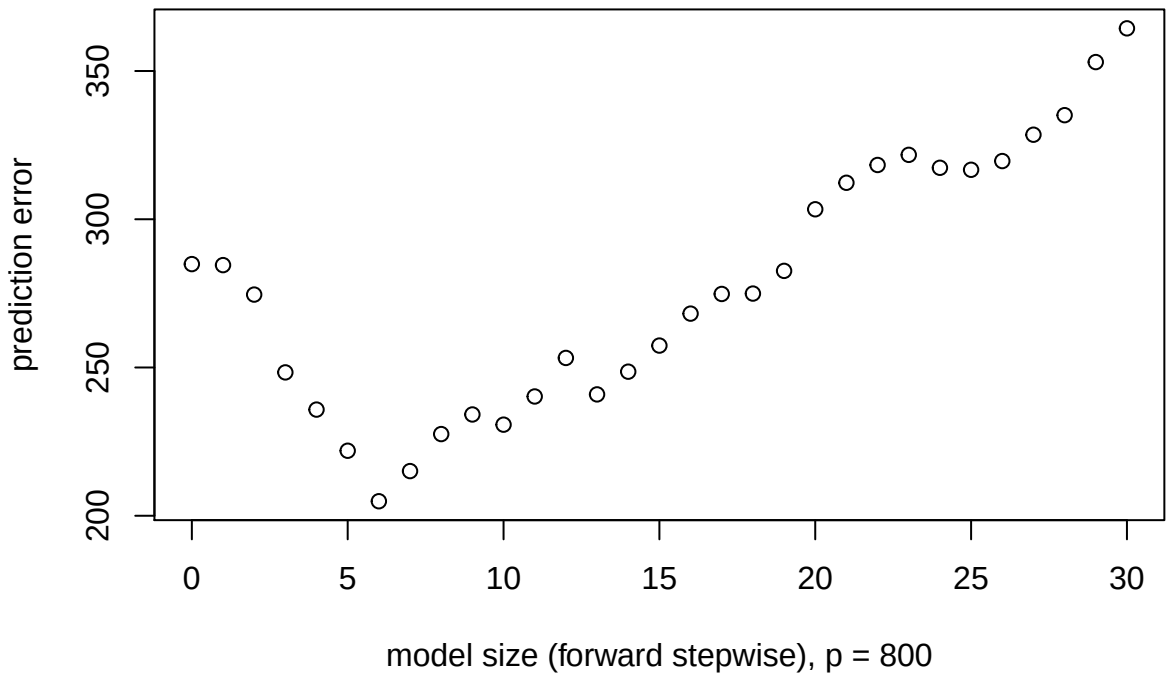
```
set.seed(1)
p = c(200,400,600,800)

for (i in 1:length(p)){
    # generate simulated data set
    n = 400
    X = matrix(rnorm(n*p[i], mean=0, sd=1), nrow=n, ncol=p[i])
    X = X %*% diag( 1/sqrt(colSums(X^2)) )  #or X = scale(X, center=FALSE, scale=sqrt(colSums(X^2)))
    Beta = c(rep(5,10),rep(0,p[i]-10))
    noise = rnorm(n, mean=0, sd=1)
    Y = X %*% Beta + noise

    # randomly separate the data into two subsets: training & validation sets
```

```
    idx = sample(1:n, size=200, replace=FALSE)
    X_train = X[idx, ]
    Y_train = Y[idx, ]
    X_val = X[-idx, ]
    Y_val = Y[-idx, ]

    # run forward stepwise without threshold of p-values
    S = NULL
    store_pred_err = rep(0,31)
    Beta0_hat = summary(lm(Y_train~1))$coefficients[1,1]
    store_pred_err[1] = sum( (Y_val-Beta0_hat)^2 )
    pvalues = NULL

    for (k in 1:30){
        S_else = setdiff(1:p[i],S)

        for (j in 1:length(S_else)){
            model = lm( Y_train ~ X_train[,c(S,S_else[j])] )
            pvalues[j] = summary(model)$coefficients[nrow(summary(model)$coefficients),4]
        }
        add_ind = S_else[which.min(pvalues)]
        S = c(S,add_ind)
        XS_train = X_train[,S,drop=FALSE]

        Beta0_hat = summary(lm(Y_train~XS_train))$coefficients[1,1]
        Beta_hat = summary(lm(Y_train~XS_train))$coefficients[-1,1]
        XS_val = X_val[,S,drop=FALSE]
        Y_pred = Beta0_hat + XS_val %*% Beta_hat

        store_pred_err[k+1] = sum( (Y_val-Y_pred)^2 )
    }

    plot(0:30,store_pred_err,xlab=paste0('model size (forward stepwise), p = ',p[i]),ylab='prediction e
}
```


prediction error
220
240
260
280
300
320
0
5
10
15
20
25
30
model size (forward stepwise), p = 200


prediction error
260
300
340
380
0
5
10
15
20
25
30
model size (forward stepwise), p = 400


prediction error
250
300
350
400
0
5
10
15
20
25
30
model size (forward stepwise), p = 600



Answer:

When p=200 < n=400, prediction error has the lowest value around model size 9. When p=400 = n=400, prediction error has the lowest value around model size 6. When p=600 or 800 > n=400, prediction error also can get the lowest value around model size 9 and 6. They are all close to the true model size 10 though still having some false negative results.

Therefore, in general, using validation method will solve the multiple tesing issue of BIC and is a much better method for large number of covariates to pick an appropriate model.

Problem 4

see next page

STAT 34300 HW5

Problem 4:

(a) $w_i = \frac{\ell_i'(Y_i - X_i^T\hat{\beta})}{Y_i - X_i^T\hat{\beta}} = \frac{1}{\sqrt{|Y_i - X_i^T\hat{\beta}|}} \Rightarrow \ell_i'(Y_i - X_i^T\hat{\beta}) = \frac{Y_i - X_i\hat{\beta}}{|Y_i - X_i\hat{\beta}|^{\frac{1}{2}}}$

$\Rightarrow \ell'(t) = \frac{t}{|t|^{\frac{1}{2}}} = \begin{cases} t^{\frac{1}{2}}, & t>0 \\ -(-t)^{\frac{1}{2}}, & t<0 \end{cases} = \text{sign}(t)\cdot|t|^{\frac{1}{2}}$

$\Rightarrow \ell(t) = \begin{cases} \frac{2}{3}t^{\frac{3}{2}}, & t>0 \\ \frac{2}{3}(-t)^{\frac{3}{2}}, & t<0 \end{cases} = \frac{2}{3}|t|^{\frac{3}{2}}$

so loss function $\ell(t) = \frac{2}{3}|t|^{\frac{3}{2}}$. i.e. $\hat{\beta} = \underset{\beta\in R^p}{\text{argmin}} \sum_i \frac{2}{3}|Y_i - X_i^T\beta|^{\frac{3}{2}}$

(b)
- LS : $\ell(t) = t^2/2$ $\quad \ell'(t) = t$
- LAD : $\ell(t) = |t|$ $\quad \ell'(t) = \text{sign}(t)$
- Huber : $\ell(t) = \rho(t)$ $\quad \ell'(t) = \rho'(t)$

when there is a large residuat $t = Y_i - X_i^T\hat{\beta}$, the gradient of loss function $\ell'(t)$ will change model for $\ell(t) = \frac{2}{3}|t|^{\frac{3}{2}}$ than LS : $\ell(t) = t^2/2$, but change less for $\ell(t) = \frac{2}{3}|t|^{\frac{3}{2}}$ than LAD : $\ell(t) = \text{sign}(t)$ and Huber : $\ell(t) = \rho(t)$

$\Rightarrow \ell(t) = \frac{2}{3}|t|^{\frac{3}{2}}$ is more sensitive (less robust) to outliers than LAD & Huber

$\Rightarrow \ell(t) = \frac{2}{3}|t|^{\frac{3}{2}}$ is less sensitive (more robust) to outliers than LS