Homework 3

Louis Bensard

March 22, 2018

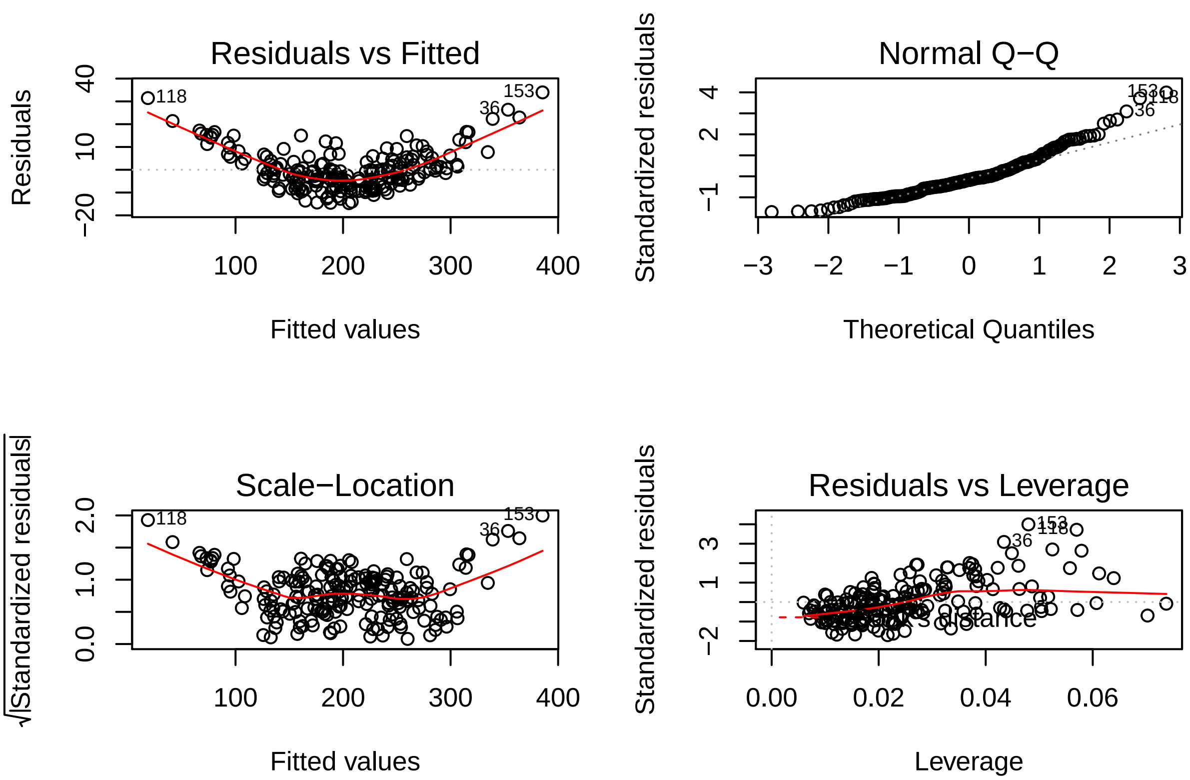
Problem 1:

```
## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-13
```

```
data = read.csv('C:/Users/Louis/Documents/UPMC/M1/Spring 2018/MATH 535/Homework 3/data4.csv',
                header=TRUE)
attach(data)
```

Let's start considering a linear model:

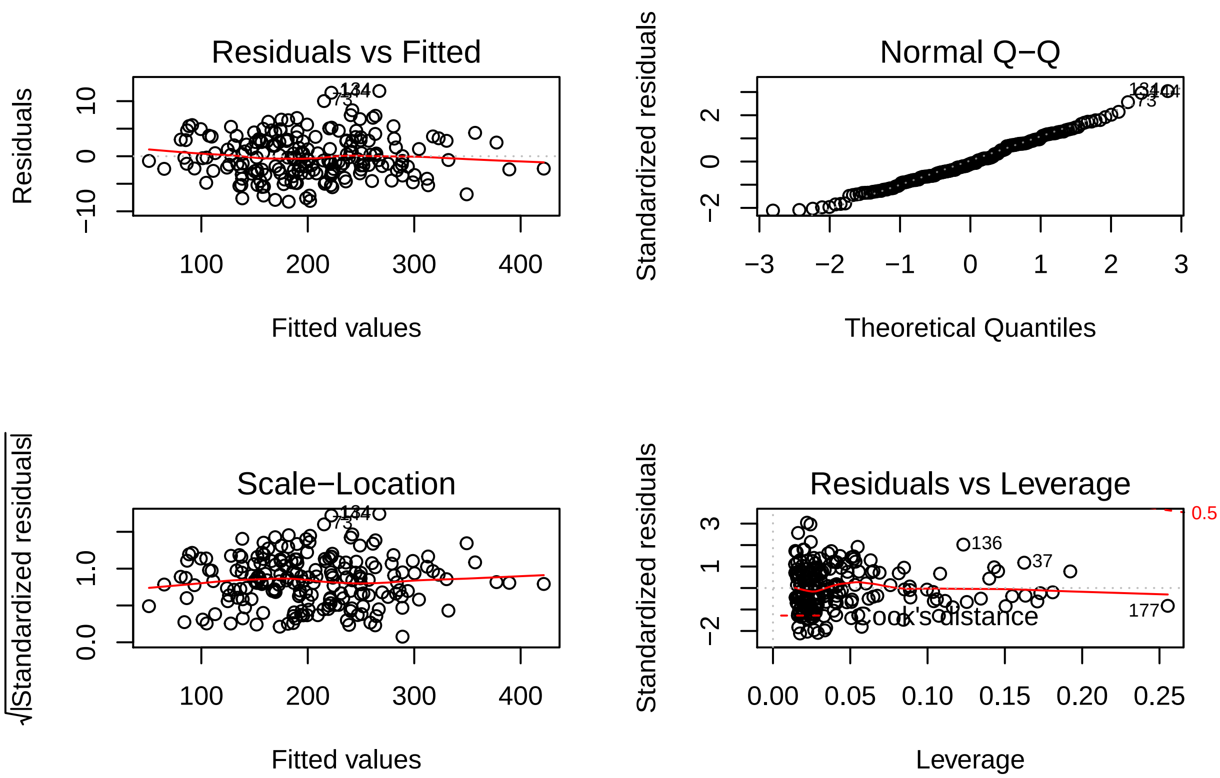
```
model1 = lm(y~ x1 + x2 + x3 + x4)
par(mfrow=c(2,2))
plot(model1)
```



Clearly, the residuals on the Residuals vs Fitted plot are not randomly spread around the 0-horizontal axis, same for the Scale-location plot around the 1-horizontal axis. Plus, the tails of the Normal QQ plot are off. Therefore, assumptions 1,2 and 3 are broken. Therefore the linear model is not valid.

The parabola shape of the Residuals vd Fitted and the "W" shape of the Scale-Location plot lead me to think that the model might be quadratic, let's explore this option:

```
model2 = lm(y~I(x1^2) + x1 + I(x2^2) + x2 + I(x3^2) + x3 + I(x4^2) + x4)
par(mfrow=c(2,2))
plot(model2)
```



This model looks better. The Residuals vs Fitted and the Scale-Location plot look good for the reasons mentionned above. The Normal QQ-plot is aligned. Therefore, this quadratic model is valid.

```
AIC(model1)
```

```
## [1] 1440.377
```

```
AIC(model2)
```

```
## [1] 1126.585
```

A quick run of AIC cnfirms the conclusion that model2 is better than model1.

But we have been pretty generous regarding the number of predictors for model2, we might be overfitting the data. Let's run Lasso to determine whether we can drop some predictors:

```
X = data.frame(x1=x1-mean(x1),x1sq=(x1-mean(x1))^2,x2=x2-mean(x2),x2sq=(x2-mean(x2))^2,
               x3=x3-mean(x3),x3sq=(x3-mean(x3))^2,x4=x4-mean(x4),x4sq=(x4-mean(x4))^2)

lasso = glmnet(as.matrix(X),y,alpha=1,lambda=1:100)
print(lasso)
```

```
##
```

```
## Call:  glmnet(x = as.matrix(X), y = y, alpha = 1, lambda = 1:100)
##
##         Df    %Dev Lambda
##    [1,]  0 0.00000    100
##    [2,]  0 0.00000     99
##    [3,]  0 0.00000     98
##    [4,]  0 0.00000     97
##    [5,]  0 0.00000     96
##    [6,]  0 0.00000     95
##    [7,]  0 0.00000     94
##    [8,]  0 0.00000     93
##    [9,]  0 0.00000     92
##   [10,]  0 0.00000     91
##   [11,]  0 0.00000     90
##   [12,]  0 0.00000     89
##   [13,]  0 0.00000     88
##   [14,]  0 0.00000     87
##   [15,]  0 0.00000     86
##   [16,]  0 0.00000     85
##   [17,]  0 0.00000     84
##   [18,]  0 0.00000     83
##   [19,]  0 0.00000     82
##   [20,]  0 0.00000     81
##   [21,]  0 0.00000     80
##   [22,]  0 0.00000     79
##   [23,]  0 0.00000     78
##   [24,]  0 0.00000     77
##   [25,]  0 0.00000     76
##   [26,]  0 0.00000     75
##   [27,]  0 0.00000     74
##   [28,]  0 0.00000     73
##   [29,]  0 0.00000     72
##   [30,]  0 0.00000     71
##   [31,]  0 0.00000     70
##   [32,]  0 0.00000     69
##   [33,]  0 0.00000     68
##   [34,]  0 0.00000     67
##   [35,]  0 0.00000     66
##   [36,]  0 0.00000     65
##   [37,]  0 0.00000     64
##   [38,]  0 0.00000     63
##   [39,]  0 0.00000     62
##   [40,]  0 0.00000     61
##   [41,]  0 0.00000     60
##   [42,]  0 0.00000     59
##   [43,]  0 0.00000     58
##   [44,]  0 0.00000     57
##   [45,]  0 0.00000     56
##   [46,]  0 0.00000     55
##   [47,]  0 0.00000     54
##   [48,]  1 0.01731     53
##   [49,]  1 0.04254     52
##   [50,]  1 0.06730     51
##   [51,]  1 0.09157     50
```

```
##  [52,]  1 0.11540     49
##  [53,]  1 0.13870     48
##  [54,]  1 0.16150     47
##  [55,]  2 0.18960     46
##  [56,]  2 0.22370     45
##  [57,]  2 0.25710     44
##  [58,]  2 0.28960     43
##  [59,]  2 0.32150     42
##  [60,]  2 0.35260     41
##  [61,]  2 0.38290     40
##  [62,]  2 0.41250     39
##  [63,]  2 0.44130     38
##  [64,]  2 0.46940     37
##  [65,]  2 0.49680     36
##  [66,]  2 0.52340     35
##  [67,]  2 0.54920     34
##  [68,]  2 0.57430     33
##  [69,]  2 0.59860     32
##  [70,]  2 0.62220     31
##  [71,]  2 0.64510     30
##  [72,]  2 0.66720     29
##  [73,]  2 0.68850     28
##  [74,]  2 0.70910     27
##  [75,]  2 0.72900     26
##  [76,]  2 0.74810     25
##  [77,]  2 0.76650     24
##  [78,]  2 0.78410     23
##  [79,]  2 0.80090     22
##  [80,]  2 0.81700     21
##  [81,]  2 0.83240     20
##  [82,]  2 0.84700     19
##  [83,]  2 0.86080     18
##  [84,]  2 0.87400     17
##  [85,]  2 0.88630     16
##  [86,]  2 0.89790     15
##  [87,]  2 0.90880     14
##  [88,]  2 0.91890     13
##  [89,]  2 0.92830     12
##  [90,]  2 0.93690     11
##  [91,]  2 0.94470     10
##  [92,]  2 0.95190      9
##  [93,]  2 0.95820      8
##  [94,]  2 0.96380      7
##  [95,]  2 0.96870      6
##  [96,]  3 0.97540      5
##  [97,]  4 0.98270      4
##  [98,]  4 0.98870      3
##  [99,]  4 0.99290      2
## [100,]  4 0.99550      1
```

```r
print(lasso$beta)
```

```
## 8 x 100 sparse Matrix of class "dgCMatrix"
##    [[ suppressing 100 column names 's0', 's1', 's2' ... ]]
```

```
##
## x1   . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x1sq . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x2   . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x2sq . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x3   . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x3sq . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x4   . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## x4sq . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
##
## x1   . . . . . . . . . . . . 0.2309579 0.5730167 0.9150755 1.257134
## x1sq . . . . . . . . . . . . .         .         .         .
## x2   . . . . . . . . . . . . .         .         .         .
## x2sq . . . . . . . . . . . . .         .         .         .
## x3   . . . . . . . . . . . . .         .         .         .
## x3sq . . . . . . . . . . . . .         .         .         .
## x4   . . . . . . . . . . . . .         .         .         .
## x4sq . . . . . . . . . . . . .         .         .         .
##
## x1   1.599193 1.941252 2.28331 2.5902682 2.8568484 3.1234077 3.3899671
## x1sq .        .        .       .         .         .         .
## x2   .        .        .       0.1089426 0.3431524 0.5773673 0.8115822
## x2sq .        .        .       .         .         .         .
## x3   .        .        .       .         .         .         .
## x3sq .        .        .       .         .         .         .
## x4   .        .        .       .         .         .         .
## x4sq .        .        .       .         .         .         .
##
## x1   3.656526 3.923086 4.189645 4.456204 4.722764 4.989323 5.255883
## x1sq .        .        .        .        .        .        .
## x2   1.045797 1.280012 1.514227 1.748442 1.982657 2.216872 2.451087
## x2sq .        .        .        .        .        .        .
## x3   .        .        .        .        .        .        .
## x3sq .        .        .        .        .        .        .
## x4   .        .        .        .        .        .        .
## x4sq .        .        .        .        .        .        .
##
## x1   5.522442 5.789001 6.055561 6.322120 6.588679 6.855239 7.121798
## x1sq .        .        .        .        .        .        .
## x2   2.685302 2.919516 3.153731 3.387946 3.622161 3.856376 4.090591
## x2sq .        .        .        .        .        .        .
## x3   .        .        .        .        .        .        .
## x3sq .        .        .        .        .        .        .
## x4   .        .        .        .        .        .        .
## x4sq .        .        .        .        .        .        .
##
## x1   7.388357 7.654917 7.921476 8.188035 8.454595 8.721154 8.987714
## x1sq .        .        .        .        .        .        .
## x2   4.324806 4.559021 4.793236 5.027451 5.261666 5.495881 5.730095
## x2sq .        .        .        .        .        .        .
## x3   .        .        .        .        .        .        .
## x3sq .        .        .        .        .        .        .
## x4   .        .        .        .        .        .        .
## x4sq .        .        .        .        .        .        .
```

```
##
## x1   9.254273 9.520832 9.787392 10.053951 10.32051 10.587070 10.85363
## x1sq .        .        .         .         .        .         .
## x2   5.964310 6.198525 6.432740  6.666955  6.90117  7.135385  7.36960
## x2sq .        .        .         .         .        .         .
## x3   .        .        .         .         .        .         .
## x3sq .        .        .         .         .        .         .
## x4   .        .        .         .         .        .         .
## x4sq .        .        .         .         .        .         .
##
## x1   11.120188 11.38675 11.653307 11.91987 12.186426 12.452985 12.719544
## x1sq  .         .        .         .        .         .         .
## x2    7.603815  7.83803  8.072245  8.30646  8.540674  8.774889  9.009104
## x2sq  .         .        .         .        .         .         .
## x3    .         .        .         .        .         .         .
## x3sq  .         .        .         .        .         .         .
## x4    .         .        .         .        .         .         .
## x4sq  .         .        .         .        .         .         .
##
## x1   12.986104 13.252663 13.55503011 13.88224563 14.2135599 14.5448664
## x1sq  .         .         0.07882934  0.16147060  0.2423981  0.3233255
## x2    9.243319  9.477534  9.69406595  9.90181524 10.1082669 10.3147205
## x2sq  .         .         .           0.05401684  0.1196722  0.1853275
## x3    .         .         .           .           .          .
## x3sq  .         .         .           .           .          .
## x4    .         .         .           .           .          .
## x4sq  .         .         .           .           .          .
##
## x1   14.8761729
## x1sq  0.4042529
## x2   10.5211741
## x2sq  0.2509828
## x3    .
## x3sq  .
## x4    .
## x4sq  .
```

Lasso tells us that we can explain 99.56% of the model2 using only 4 predictors, therefore we can easily drop those quasi-useless predictors, and thus we obtain the following final model:

```
bestModel = lm(y~I(x1^2) + x1 + I(x2^2) + x2)
AIC(bestModel)
```

```
## [1] 1123.854
```

$1123.85 = AIC(bestModel) < AIC(model2) = 1126.59$, so we indeed improved our model by dropping 4 predictors. I can now be confident that this last model is the best valid model for predicting y using x_1, x_2, x_3, x_4.

Problem 2:

(a)

First we started with a linear model:

```r
data = na.omit(airquality)
model_a1 = lm(Ozone~Temp, data=data)
```

All the residual plots did not look good, but it seemed that a quadratic model may work:

```r
model_a2 = lm(Ozone~ I(Temp^2) + Temp, data=data)
```

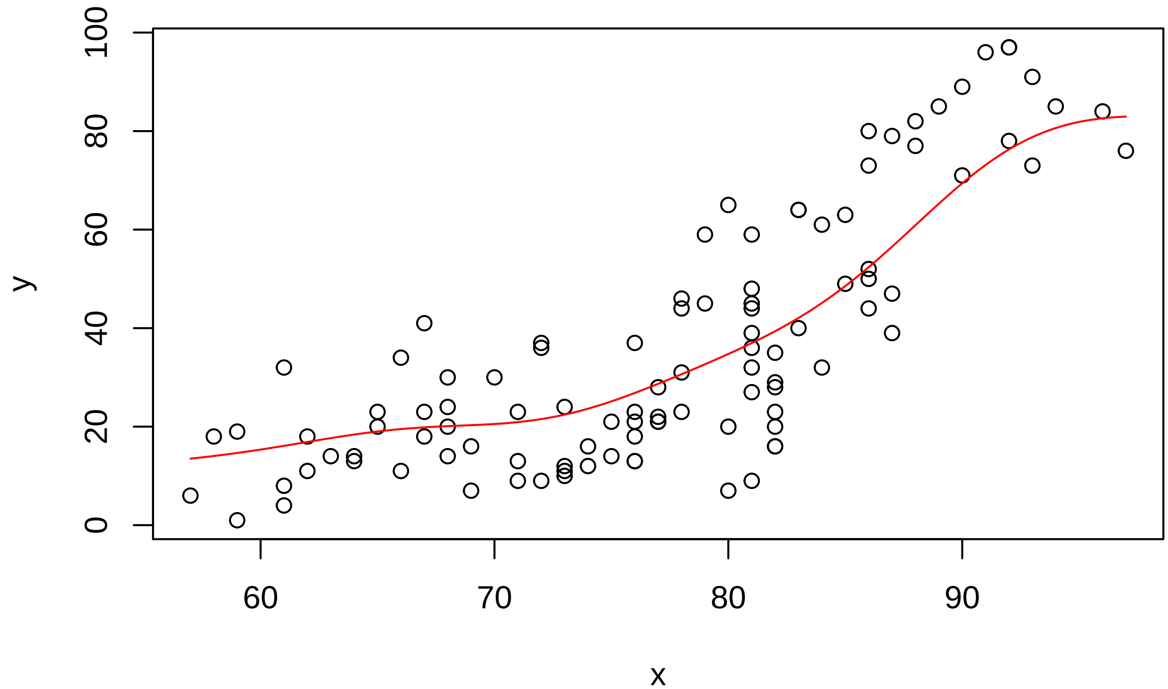
That model fixed the residuals plots except for the Normal QQ plot. Therefore I tried a cubic model and other power models but it doesn't fix the Normal QQ plot, thus we are going to use a non-parametric model obtained by kernel smoothing.

Our model is now $y = \hat{y} + \epsilon$ with:

$$\hat{y} = \frac{\sum_{i=1}^{n} y_i \cdot exp\{\frac{-(x_i-x)^2}{2h^2}\}}{\sum_{i=1}^{n} exp\{\frac{-(x_i-x)^2}{2h^2}\}}$$

If we plot Ozone vs Temp, we notice few outliers (data points where Ozone > 100). Therefore, we decided to get rid of those points so that our non-parametric model is more accurate.

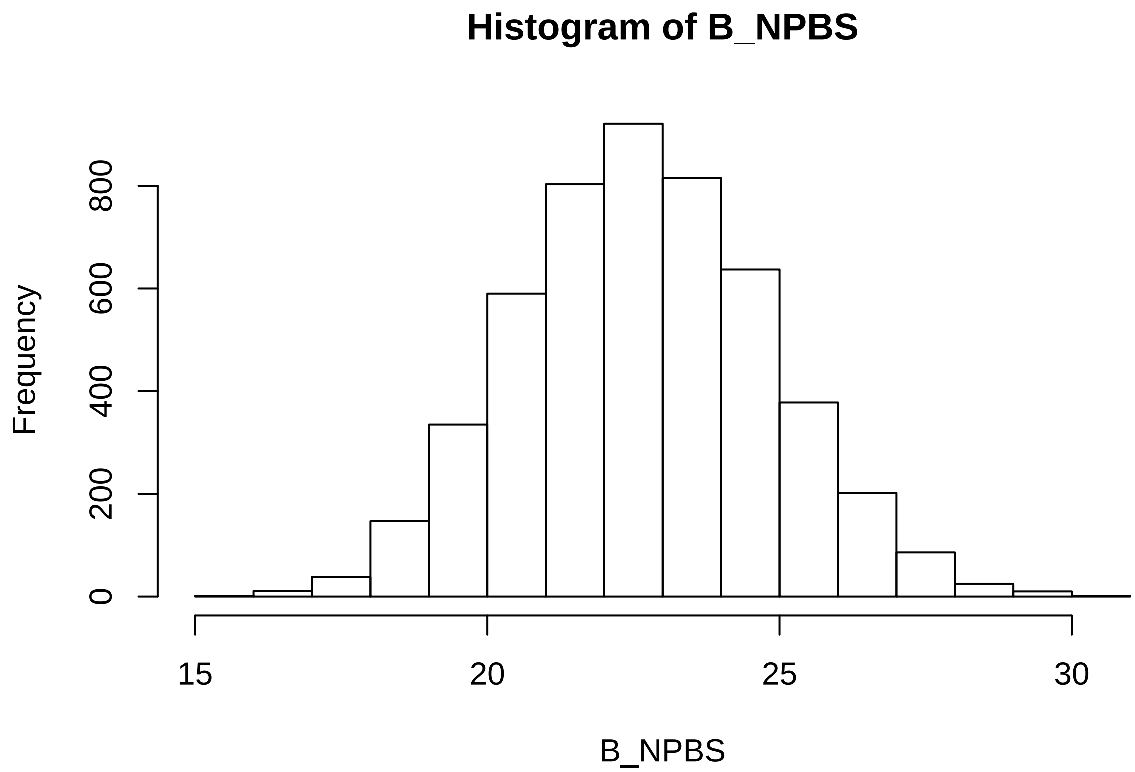
```r
data = data[c(-23, -34,-53,-63,-65,-77,-80),]
attach(data)
y = Ozone
x = Temp
n = length(x)

#Kernel Smoothing Model

#Silverman's rule
h=1.06*sd(x)*(n^(-1/5)) # ~3.9

x1 = seq(min(x),max(x),by=(max(x)-min(x))/(n-1)) ; y1 = rep(0,n) ; y3=rep(0,n)

for(i in 1:n){
  y1[i] = sum(exp(-.5*((x1[i]-x)/h)^2)*y)/sum(exp(-.5*((x1[i]-x)/h)^2))
}

plot(x,y)
lines(x1,y1,col="red")
```



(b)

```
#Note that we replace x1 by x below to get the residuals
for(i in 1:n){
    y3[i] = sum(exp(-.5*((x[i]-x)/h)^2)*y)/sum(exp(-.5*((x[i]-x)/h)^2))
}

#Boostrapping residuals
residuals = y-y3

B_NPBS = rep(0,5000)

for(i in 1:5000){

    #Sample n=length(x) new_x's from the original sample x
    new_x = sample(x,n,replace=T)

    fit_y = rep(0,n) ; new_fit_y = rep(0,n)

    #we get a new y_hat from this new_x vector
    for(j in 1:n){
        fit_y[j] = sum(exp(-.5*((new_x[j]-x)/h)^2)*y)/sum(exp(-.5*((new_x[j]-x)/h)^2))
    }
```

```r
    #we get a new response
    new_y = fit_y + sample(residuals,n,replace=T)

    #we new y_hat using the new response and the new x
    B_NPBS[i] = sum(exp(-.5*((70-new_x)/h)^2)*new_y)/sum(exp(-.5*((70-new_x)/h)^2))

}

hist(B_NPBS)
```




```r
#95% confidence interval of the average y when x=70
L = sort(B_NPBS)[125]
U = sort(B_NPBS)[4875]
cat("\nCI = [",L,",", U,"]")
```

```
##
## CI = [ 18.59812 , 26.97474 ]
```

The histogram of the boostrapped y's looks Normal, as excpeted. We assume the symmetry of the graph then.

Therefore, a 95% confidence interval for the average Ozone on a day in which it's 70 degrees is CI.

(c)

Our model is not a linear model anymore, it is obtained by kernel smoothing. We cannot provide any inference for a data point outside of the range of Temp. $20 < min(Temp) = 57$ is clearly out of the range of Temp. So it wouldn't be appropriate to replicate our solution to part (b) for the average Ozone on a day in which it's 20 degrees.

Note that even with a parametric model this wouldn't be possible because 20 is a clear outlier or the Data Temp.

(d)

To make sure we did not overfit the data, we are going to Cross Validate our non-parametric model, keep track of the residuals during cross validation and then compare r^2's with and without cross validation.

```
residuals1 = c()

for(i in 1:n){

    data1 = data[-i,]
    y2 = data1$Ozone
    x2 = data1$Temp

    predictions1 = sum(exp(-.5*((x[i]-x2)/h)^2)*y2)/sum(exp(-.5*((x[i]-x2)/h)^2))

    test_residuals1 = y[i] - predictions1

    residuals1 = c(residuals1,test_residuals1)

}

SST = sum((y - mean(y))^2)

SSR_CV = sum(residuals1^2)
r2_CV = (SST - SSR_CV)/SST
cat("r2_CV = ",r2_CV,"\n")
```

```
## r2_CV =  0.7122785
```

```
SSR_noCV = sum(residuals^2)
r2_noCV = (SST - SSR_noCV)/SST
cat("r2_noCV = ",r2_noCV)
```

```
## r2_noCV =  0.7326359
```

Before CV, r^2 was 73.3%, after CV r^2 is 71.2%. We lost just 2%, therefore we have not overfit the data. No surpises here, because we use the Silverman's rule for h. Therefore, in one dimension, I would never overfit the data.

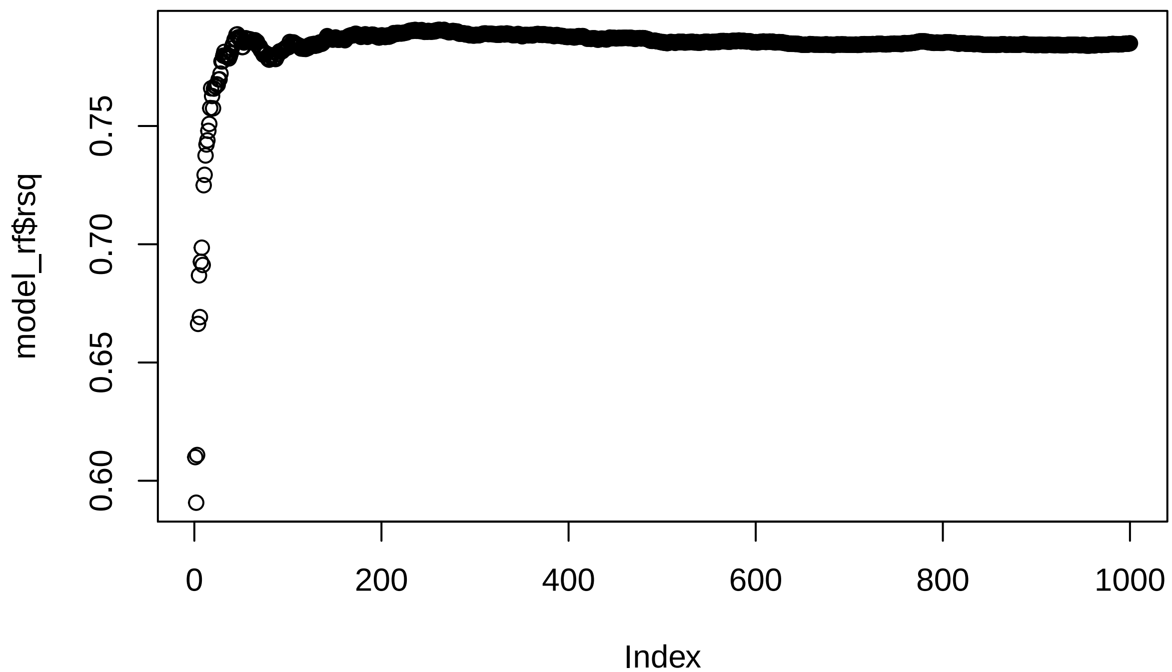
Problem 3:

```
## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.
```

Let's start with a Random Forest with 1000 trees.

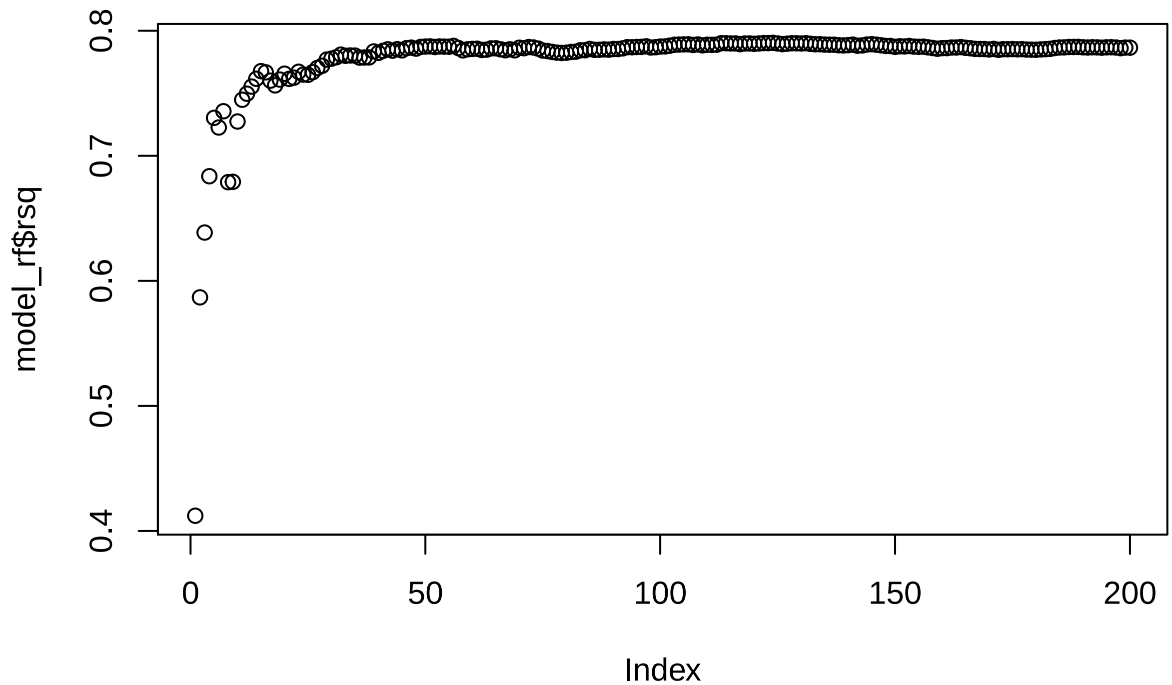
```
n_tree = 1000
model_rf = randomForest(Ozone~Temp+Wind+Solar.R
                        ,data=data,ntree=n_tree,mtry=2,control=rpart.control(minsplit=10,cp=.05))
```

```
plot(model_rf$rsq)
```



We can clearly see r^2 stabilizing around 78%. It seems that 1000 trees is way too many trees, let's use 200 trees instead.

```
n_tree = 200
model_rf = randomForest(Ozone~Temp+Wind+Solar.R
                        ,data=data,ntree=n_tree,mtry=2,control=rpart.control(minsplit=10,cp=.05))

plot(model_rf$rsq)
```



It seems now that r^2 start stabilizing after 100 trees, so let's use this number of trees for our final model.

```
n_tree = 100
model_rf = randomForest(Ozone~Temp+Wind+Solar.R
                        ,data=data,ntree=n_tree,mtry=2,control=rpart.control(minsplit=10,cp=.05))

#r^2 of the model
model_rf$rsq[n_tree]
```

```
## [1] 0.7783792
```

The r^2 of our model is $r^2 = 0.7783792$. Now, let's cross validate to make sure we are not overfitting the data (probbly not since we selected the number of trees carefully).

```
#Cross Validation

#changed to a 8-fold CV because n=104/8=13 so we don't lose data building the matrix
test_index_matrix = matrix(sample(1:n,n,replace=F),nrow=8)

#test_index_matrix has partitioned the indeces of our original data into 8 partitions
#each rpresented by a row in the matrix.

residuals_rf = c()
for(i in 1:8){
    train_data = data[c(test_index_matrix[-i,]),]
    test_data = data[c(test_index_matrix[i,]),]

    model_rf_train = randomForest(Ozone~Temp+Wind+Solar.R ,data=train_data,
```

```r
                                   ntree=n_tree,mtry=2,control=rpart.control(minsplit=10,cp=.05))

    predictions = predict(model_rf_train, test_data)

    test_residuals = test_data$Ozone - predictions
    residuals_rf = c(residuals_rf, test_residuals)
}

SST = sum((y - mean(y))^2)
SSR_rf = sum(residuals_rf^2)
r2CV = (SST - SSR_rf)/SST
print(r2CV)
```

```
## [1] 0.790243
```

$r2CV = 0.790243$ is very close to the r^2 we got above for our random forest model. In fact, it is greater than that r^2. Therefore, we are not overfitting the data.

Note that I couldn't find a number of trees large enough that would overfit the data. Indeed, even for 100 000 trees, the cross validation r^2 is still very close to the original r^2. Above a number of trees large enough, I expected the cross validation r^2CV to get far below the r^2 with no cross validation (say at least 0.1 less), but that never happened even for hundreds of thousands of trees.. Can't figure out why

As a result, we used the graph of the cumulative r^2 to determine the optimal amount of trees and thus, the Random Forest model with 80 trees is a valid model for predicting Ozone as a function of Temp, Wind and Solar.R.