a2

Mushi Wang

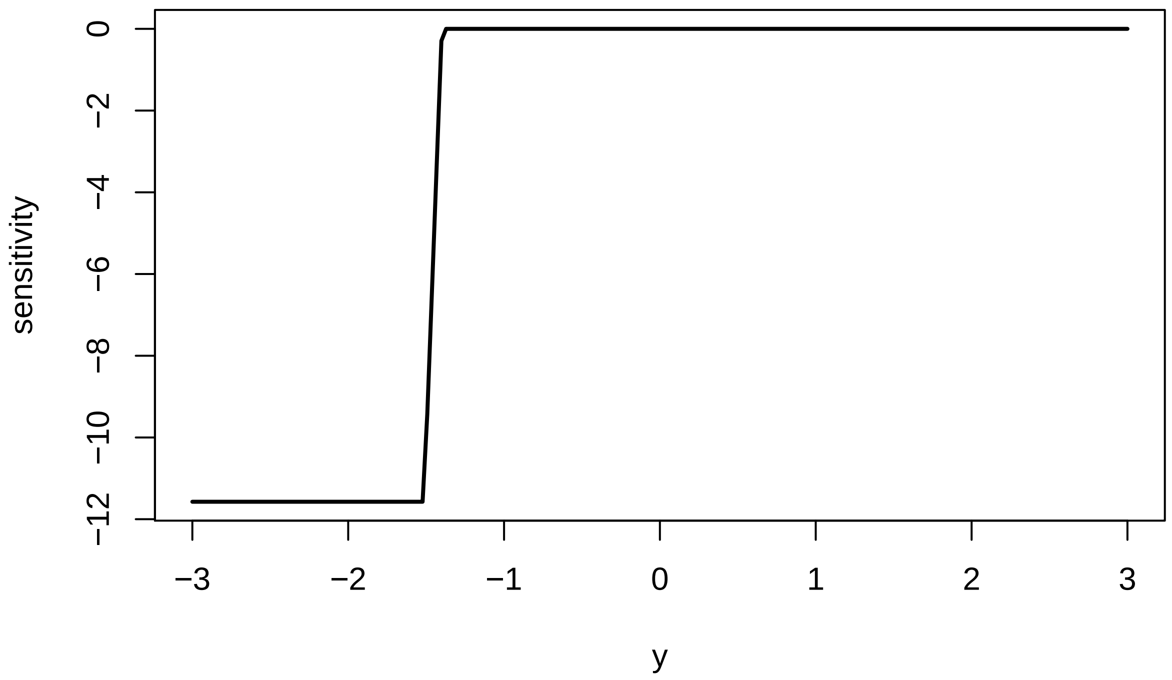
27/06/2020

1.(a) If $y < y_{(k-1)}$, $a(y_1, \ldots, y_{N-1}, y) = y_{(k-1)}$.\ If $y_{(k-1)} \leq y \leq y_{(k)}$, $a(y_1, \ldots, y_{N-1}, y) = y$.\ If $y_{(k)} < y$, $a(y_1, \ldots, y_{N-1}, y) = y_{(k)}$\ Hence,

$$SC(y) = \begin{cases} N(y_{(k-1)} - y_{(k)}) & y < y_{(k-1)} \\ N(y - y_{(k)}) & y_{(k-1)} \leq y \leq y_{(k)} \\ 0 & y_{(k)} < y \end{cases}$$

1.(b)

```r
set.seed(444)
ys = rnorm(100)
N = length(ys) + 1
k = 5
y_ordered = sort(ys)
y_k = y_ordered[k]
y_kminusOne = y_ordered[k - 1]
y = seq(-3 , 3, length.out=200)

sc = function(y, y_kminusOne, y_k) {
  if(y < y_kminusOne) {
    return(N * (y_kminusOne - y_k))
  } else if(y_k < y) {
    return(0)
  } else {
    return(N * (y - y_k))
  }
}

sensitivity = vector("numeric", 200)
for(i in 1:200) {
  sensitivity[i] = sc(y[i], y_kminusOne, y_k)
}

plot(y, sensitivity, type="l", lwd = 2)
```



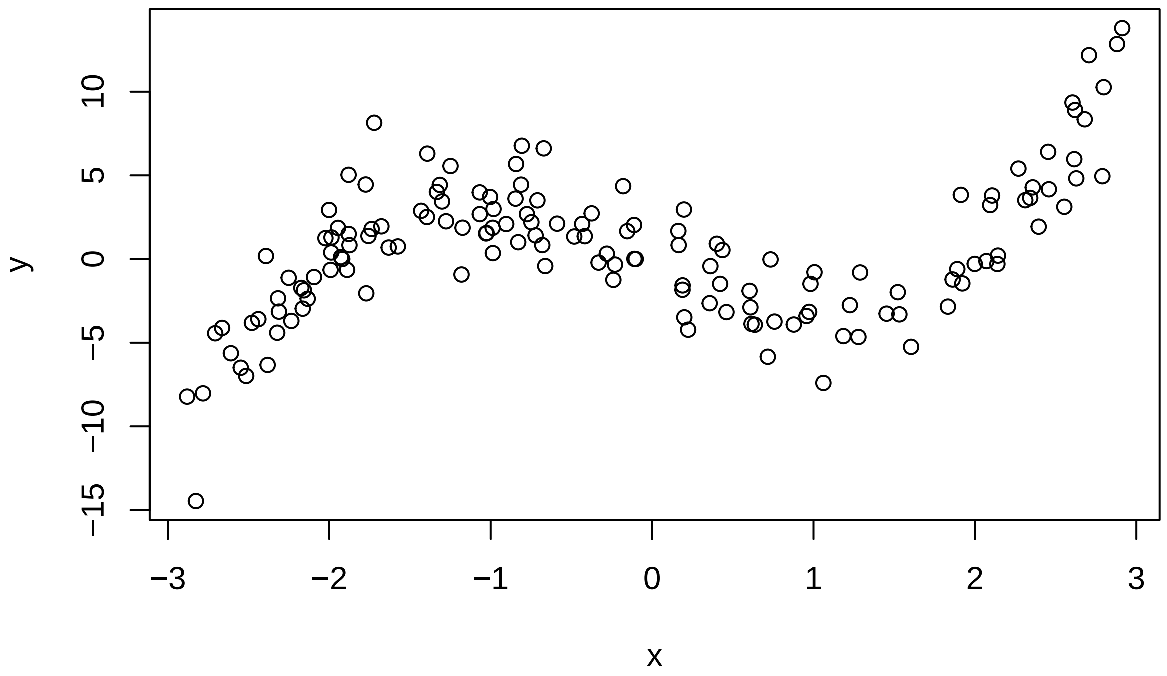
The sensitivity curve is bounded and has a linear increase between $y_{(k-1)}$ and y_k.

1.(c) The break down point is $min\{\frac{k}{N}, 1 - \frac{k}{N}\}$

2.(a) Advantage of K-fold cross validation: Since we are interested in the actual prediction error, not the prediction error of a certain set. By using K-fold cross validation the average of these errors estimates the expected prediction error. Using validation set approach may not result a good representation of the prediction error (variance may be large). Disadvantage of K-fold cross validation: The choice of k. Large k results an approximately unbiased estimator. But it also results high variance. Similarly, small k results low variance but high biasness.

2.(b)

```r
set.seed(444)
x = runif(150,-3,3)
y = (x+2)*(x)*(x-2) + rnorm(150,sd=2)
plot(x, y)
```

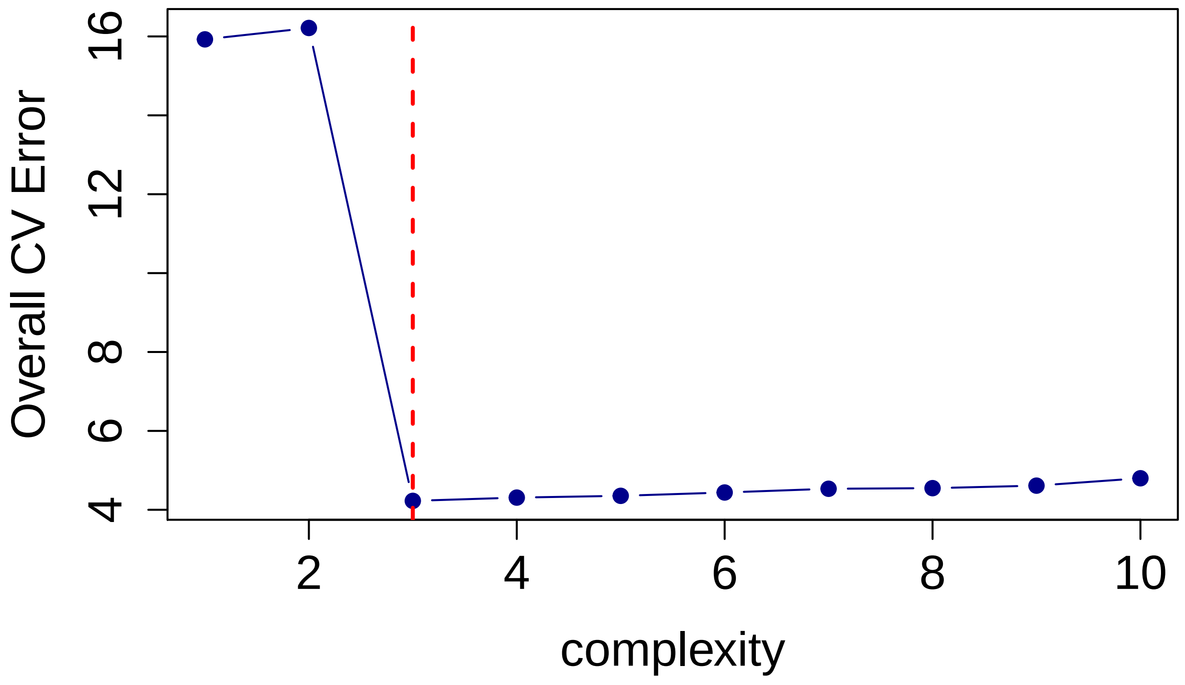


$Y|X = X(X+2)(X-2) + \epsilon$

2.(c)

```r
library(boot)
complexity = c(1:10)
cv.err = vector("numeric", length(complexity))
data = data.frame('x' = x, 'y' = y)
set.seed(444)
for(i in 1:length(complexity)){
  glm.fit = glm(y ~ poly(x, complexity[i]))
  cv.err[i] = cv.glm(data, glm.fit)$delta[1]
}
```

```r
plot(complexity, cv.err, pch=19, col="darkblue", type="b",
     cex.axis = 1.5, cex.lab=1.5, ylab="Overall CV Error")
indx = which.min(cv.err)
abline(v=indx, lty=2, lwd=2, col='red')
```



the LOOCV errors are

```r
cv.err
```

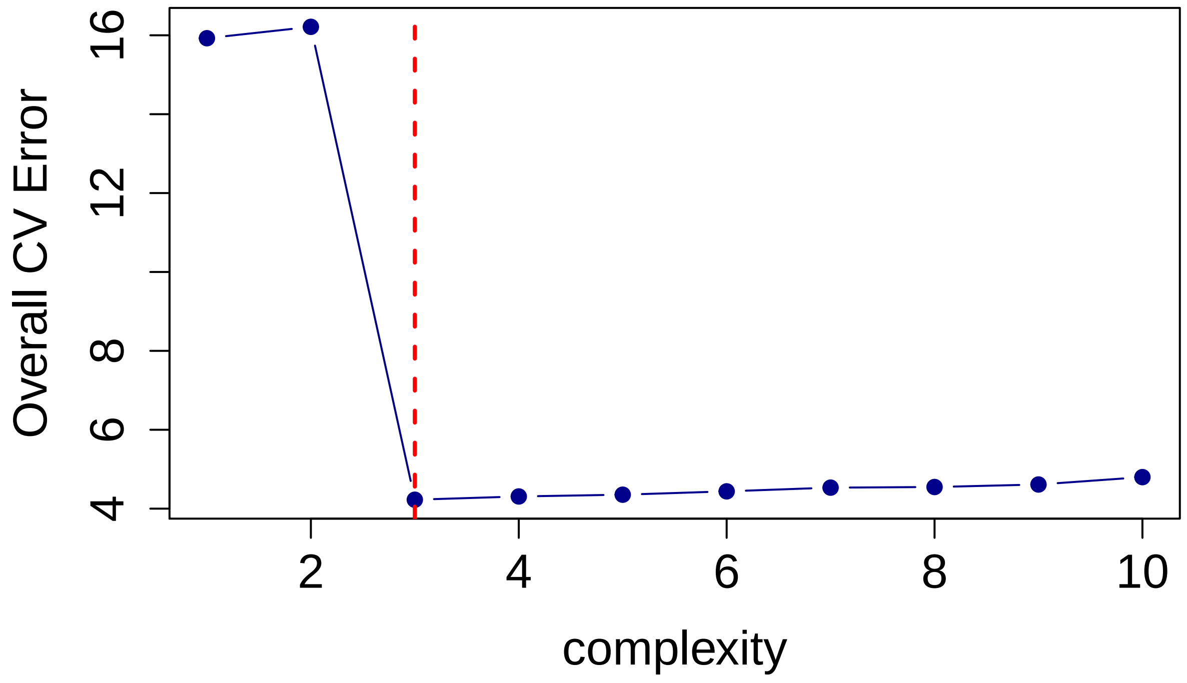
```
##  [1] 15.925694 16.214299  4.226020  4.308672  4.353616  4.437838  4.532520
##  [8]  4.548312  4.611843  4.798791
```

We choose the model with complexity 3.

2.(d)

```r
data = data.frame('x' = x, 'y' = y)
set.seed(844)
for(i in 1:length(complexity)){
  glm.fit = glm(y ~ poly(x, complexity[i]))
  cv.err[i] = cv.glm(data, glm.fit)$delta[1]
}
```

```r
plot(complexity, cv.err, pch=19, col="darkblue", type="b",
     cex.axis = 1.5, cex.lab=1.5, ylab="Overall CV Error")
indx = which.min(cv.err)
abline(v=indx, lty=2, lwd=2, col='red')
```



the LOOCV errors are

```r
cv.err
```

```
##  [1] 15.925694 16.214299  4.226020  4.308672  4.353616  4.437838  4.532520
##  [8]  4.548312  4.611843  4.798791
```

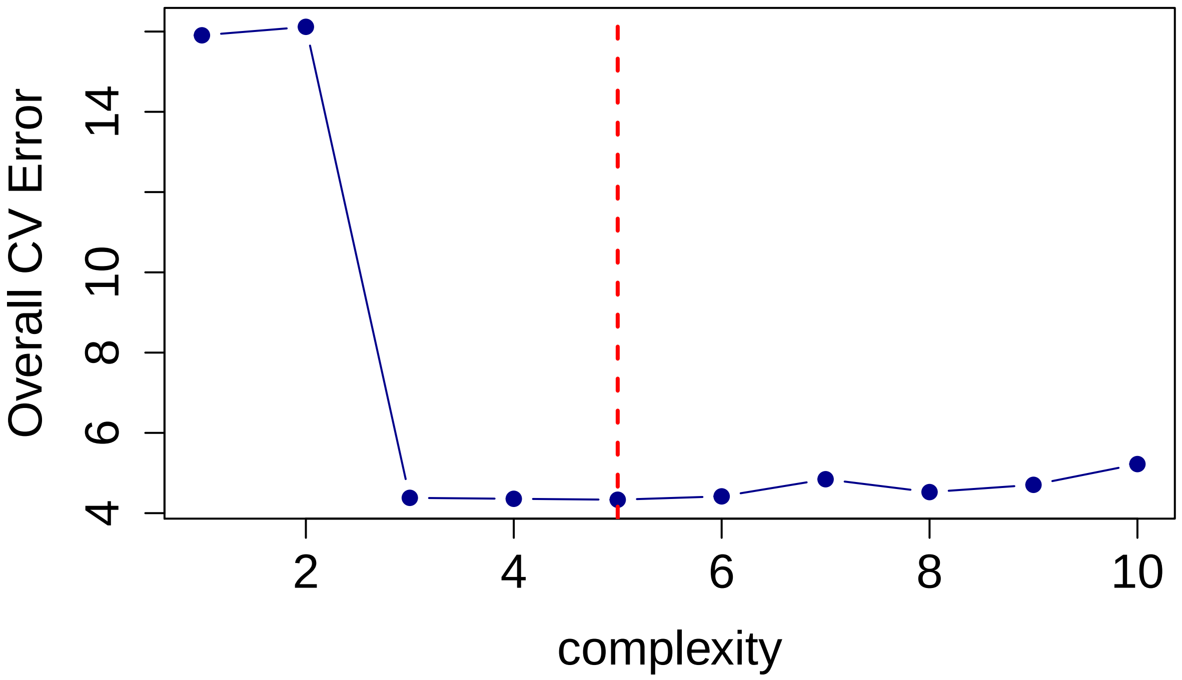
We choose the model with complexity 3.

There is no difference and that is what we would expect. Since we are performing LOOCV, every point is in exactly one test set and every set has size 1. So it does not change the result when we change the seed.

2.(e) 10-fold CV with seed number 444:

```
data = data.frame('x' = x, 'y' = y)
set.seed(444)
for(i in 1:length(complexity)){
  glm.fit = glm(y ~ poly(x, complexity[i]))
  cv.err[i] = cv.glm(data, glm.fit, K = 10)$delta[1]
}
```

```
plot(complexity, cv.err, pch=19, col="darkblue", type="b",
     cex.axis = 1.5, cex.lab=1.5, ylab="Overall CV Error")
indx = which.min(cv.err)
abline(v=indx, lty=2, lwd=2, col='red')
```



```
cv.err
```

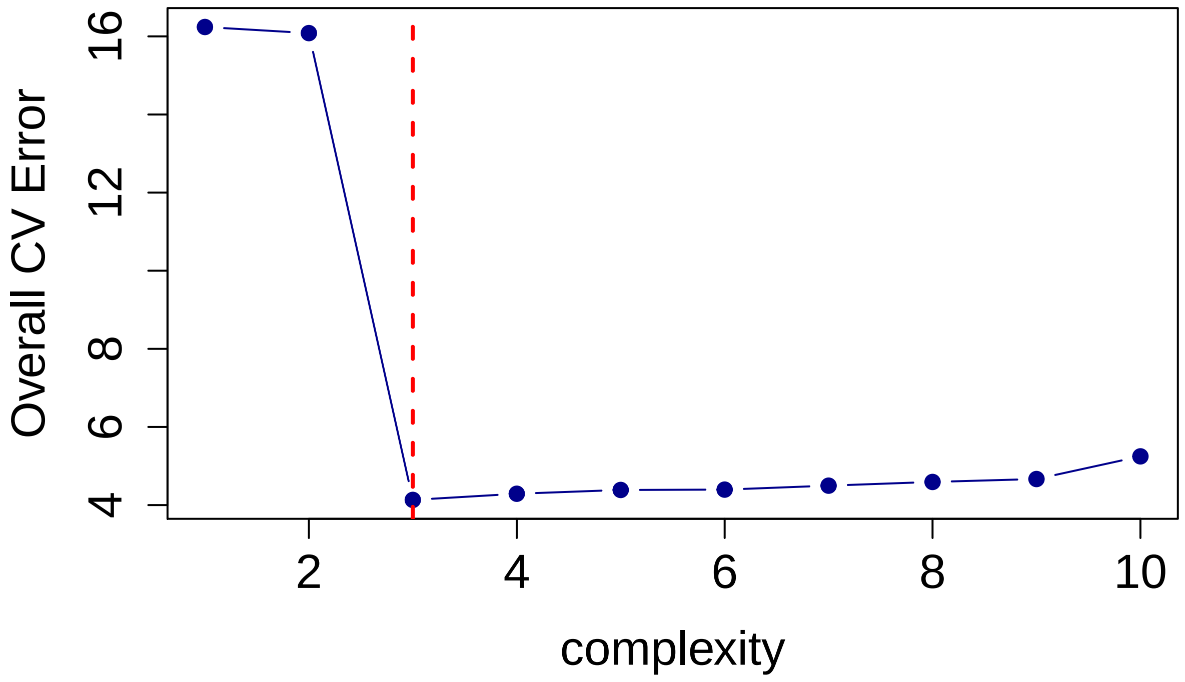
```
##  [1] 15.906805 16.117685  4.381324  4.357353  4.334346  4.418009  4.845047
##  [8]  4.524723  4.705431  5.222740
```

we choose the model with complexity 5.

10-fold CV with seed number 844:

```
data = data.frame('x' = x, 'y' = y)
set.seed(844)
for(i in 1:length(complexity)){
  glm.fit = glm(y ~ poly(x, complexity[i]))
  cv.err[i] = cv.glm(data, glm.fit, K = 10)$delta[1]
}
```

```r
plot(complexity, cv.err, pch=19, col="darkblue", type="b",
     cex.axis = 1.5, cex.lab=1.5, ylab="Overall CV Error")
indx = which.min(cv.err)
abline(v=indx, lty=2, lwd=2, col='red')
```



```r
cv.err
```

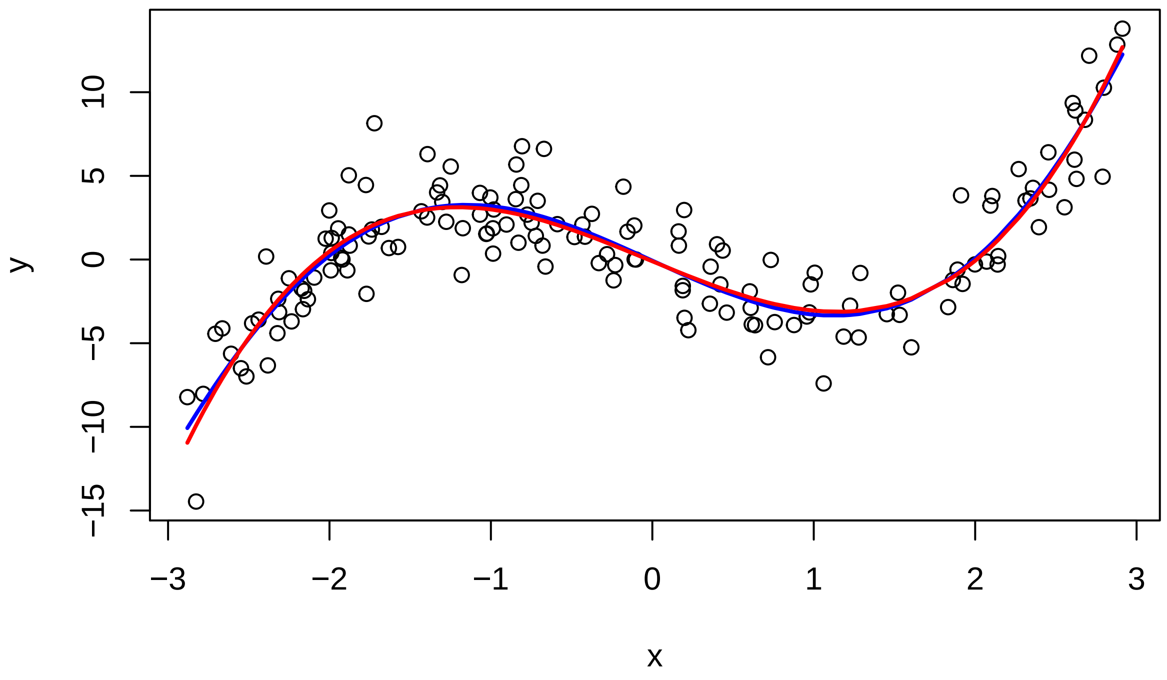
```
## [1] 16.239569 16.083618  4.132356  4.289846  4.386585  4.395705  4.496957
## [8]  4.591881  4.666508  5.247140
```

we choose the model with complexity 3.

changing the seed changes the cv error and that is what we would expect. Since we are performing a 10-fold CV. The random seed affects the elements in test and train sets. So different seed will result a slightly different errors.

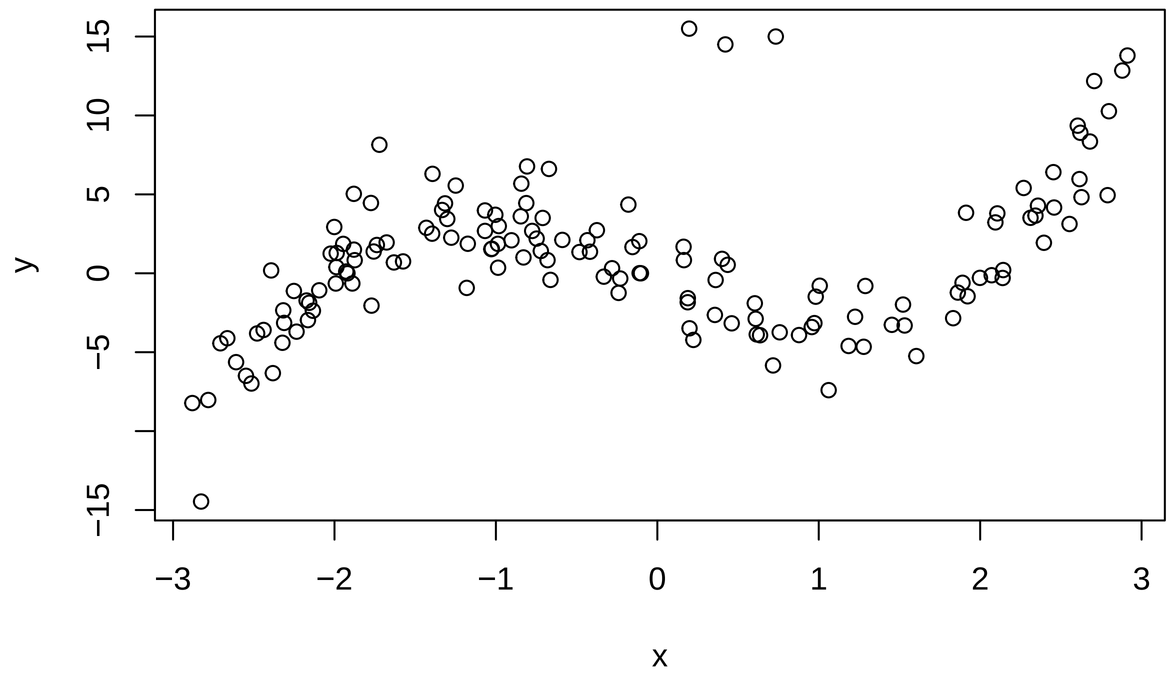
2.(f) All models choose the polynomial with degree 3 except 10-fold CV with seed 444 which agree with the actual model $f(x) = x(x+2)(x-2)$. But if we plot the polynomial with degree 5 and 3, there is not much difference between two of them and the cv errors of degree 3 and 5 are reasonable close.

```r
xorder = order(x)
plot(x, y)
lines(x[xorder], predict(lm(y ~ poly(x, 5)))[xorder], type="l", col="blue", lwd=2)
lines(x[xorder], predict(lm(y ~ poly(x, 3)))[xorder], type="l", col="red", lwd=2)
```



3.(a)

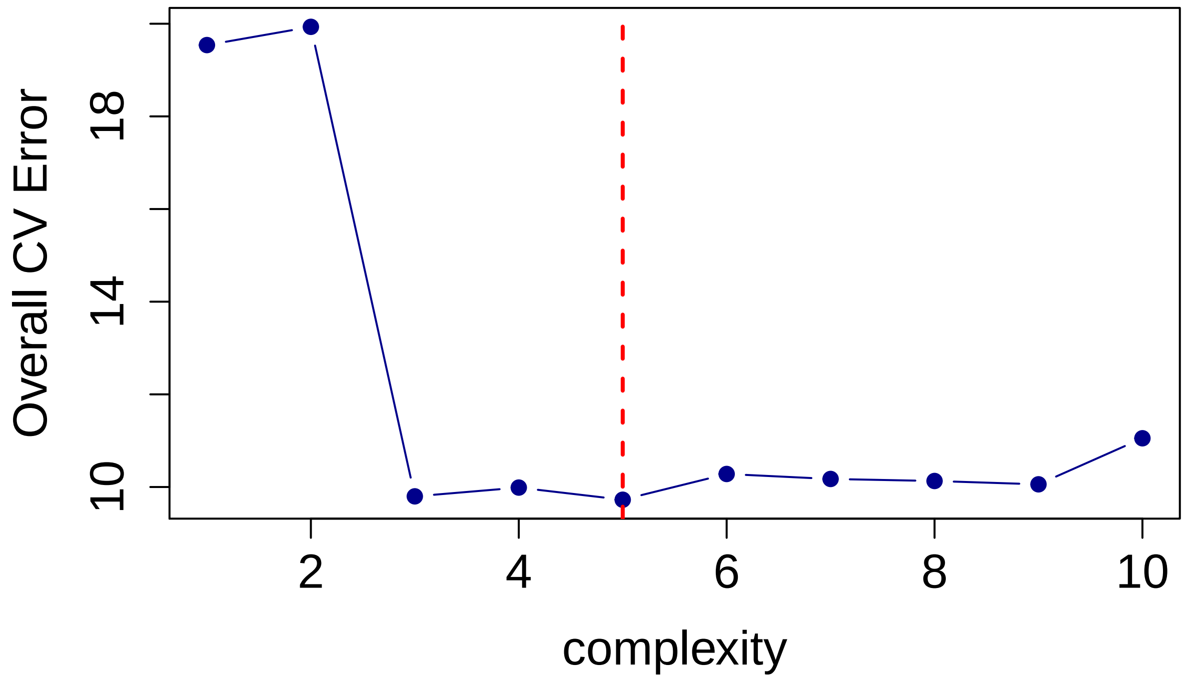
```r
set.seed(444)
x = runif(150,-3,3)
y = (x+2)*(x)*(x-2) + rnorm(150,sd=2)
Extreme.Indx = c(19 , 45 , 124)
y[Extreme.Indx] = c(15.5 , 14.5 , 15)
plot(x, y)
```



3.(b)

```r
data = data.frame('x' = x, 'y' = y)
set.seed(444)
for(i in 1:length(complexity)){
  glm.fit = glm(y ~ poly(x, complexity[i]))
  cv.err[i] = cv.glm(data, glm.fit, K = 10)$delta[1]
}
```

```r
plot(complexity, cv.err, pch=19, col="darkblue", type="b",
     cex.axis = 1.5, cex.lab=1.5, ylab="Overall CV Error")
indx = which.min(cv.err)
abline(v=indx, lty=2, lwd=2, col='red')
```



The CV errors are:

```r
cv.err
```

```
##  [1] 19.539397 19.932891  9.798028  9.987752  9.725404 10.281030 10.173550
##  [8] 10.129751 10.057638 11.051842
```

We choose the model with complexity 5.

The parameter estimates are:

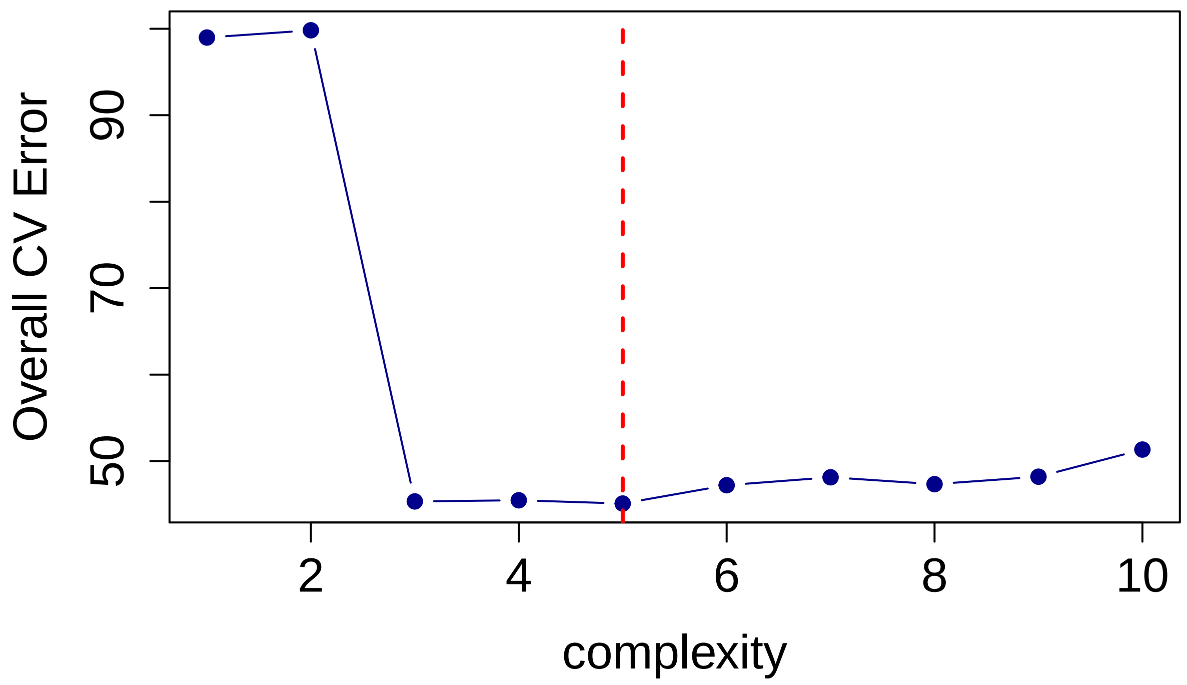
```r
glm(y ~ poly(x, 5))$coefficient
```

```
## (Intercept) poly(x, 5)1 poly(x, 5)2 poly(x, 5)3 poly(x, 5)4 poly(x, 5)5
##   0.8849621  19.7095868  -0.3411478  38.4914044   3.1915946   0.9058450
```

3.(c)

```r
cost = function(y, yhat) {
  r = y - yhat
  sum = 0
  for(i in 1:length(r)) {
    if(abs(r[i]) <= 3) {
      sum = sum + 1 / 2 * r[i] ^ 2
    } else {
      sum = sum + 3 * (abs(r[i]) - 1.5)
    }
  }
  return(sum)
}

data = data.frame('x' = x, 'y' = y)
set.seed(444)
for(i in 1:length(complexity)){
  glm.fit = glm(y ~ poly(x, complexity[i]))
  cv.err[i] = cv.glm(data, glm.fit, cost, K = 10)$delta[1]
}
```

```r
plot(complexity, cv.err, pch=19, col="darkblue", type="b",
     cex.axis = 1.5, cex.lab=1.5, ylab="Overall CV Error")
indx = which.min(cv.err)
abline(v=indx, lty=2, lwd=2, col='red')
```



The cv errors are:

```
cv.err
```

```
##  [1] 98.98340 99.81865 45.33607 45.46929 45.09229 47.21182 48.12844
##  [8] 47.33174 48.19553 51.33455
```

We choose the model with complexity 5.

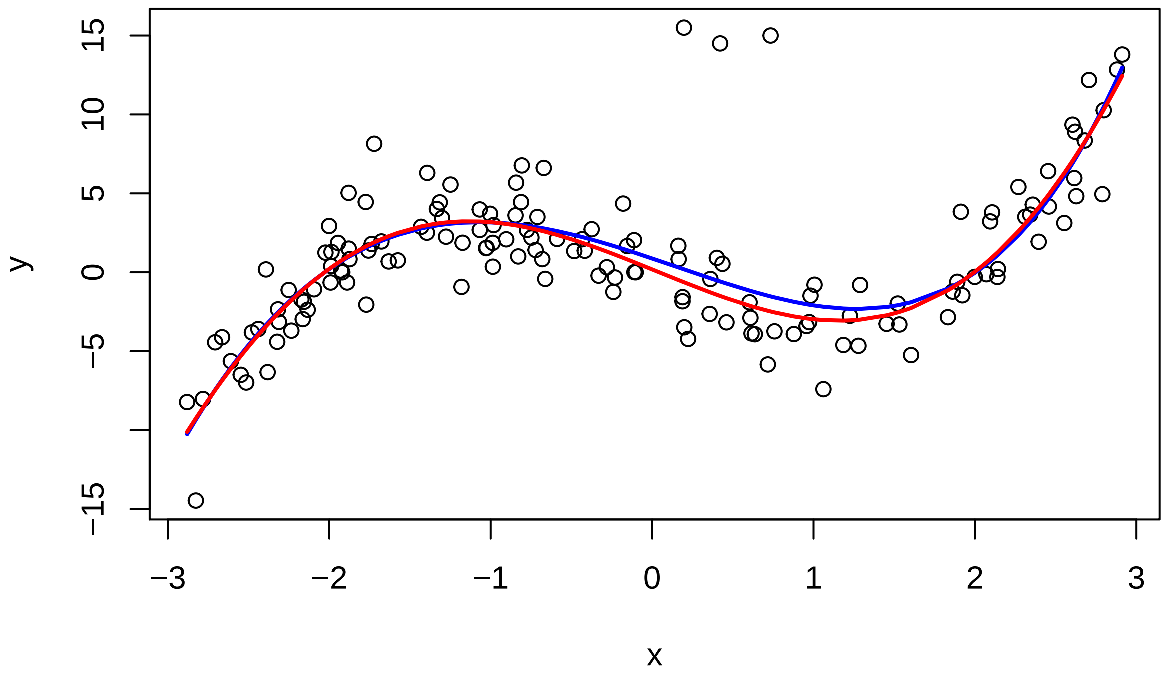
The parameter estimates are:

```
library(MASS)

huberfn = function (u, k = 3) {
  return(pmin(1, k/abs(u)))
}
rlm(y ~ poly(x, 5), psi = huberfn)$coefficient
```

```
## (Intercept) poly(x, 5)1 poly(x, 5)2 poly(x, 5)3 poly(x, 5)4 poly(x, 5)5
##   0.6684345  18.6787377   2.4116282  40.5387255   1.4500114  -1.5255481
```

3.(d)

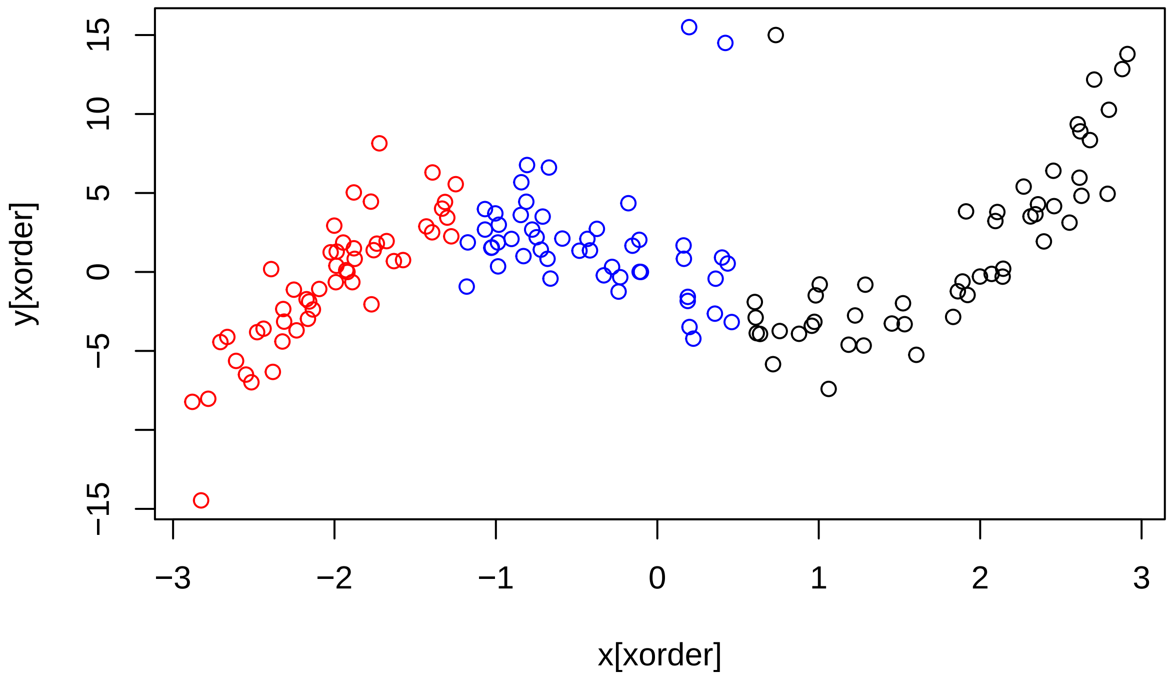
```r
xorder = order(x)
plot(x, y)
lines(x[xorder], predict(lm(y ~ poly(x, 5)))[xorder], type="l", col="blue", lwd=2)
lines(x[xorder], predict(rlm(y ~ poly(x, 5), psi = huberfn))[xorder], type="l", col="red", lwd=2)
```



Both lines are reasonable close to each other except the range (-1, 1). Since robust fit is less sensitive to outliers.

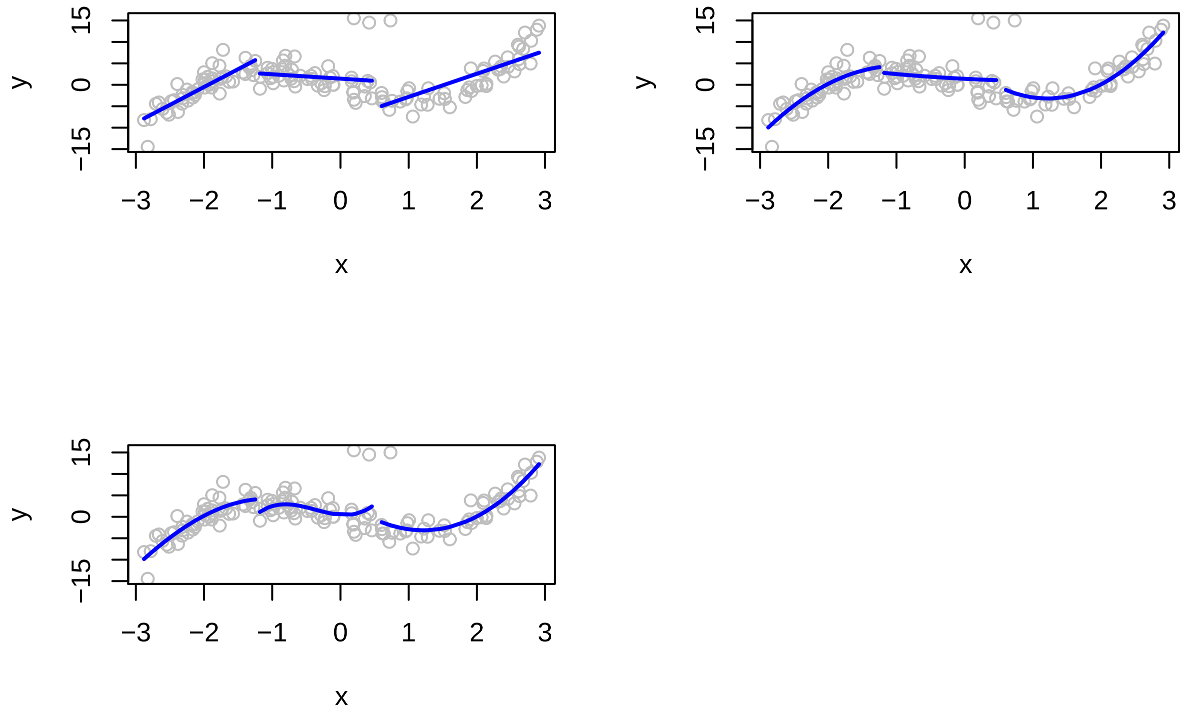
4.(a)

```
clr = c(rep('red', 50), rep('blue', 50), rep('black', 50))
plot(x[xorder], y[xorder], col = clr)
```



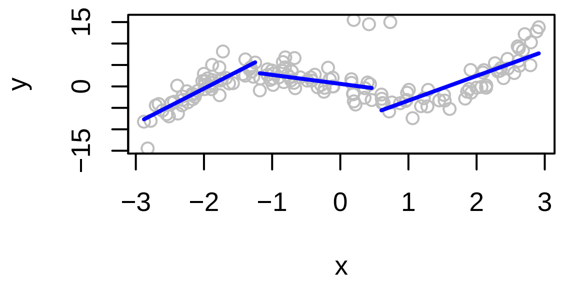
4.(b)

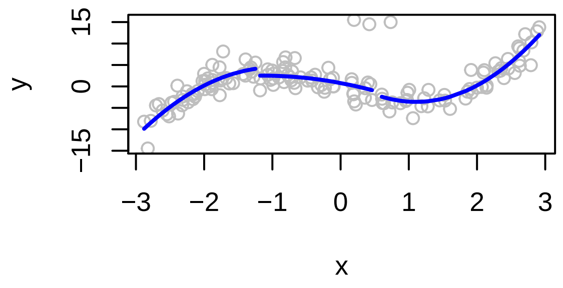
```r
par(mfrow = c(2, 2))
complexity = c(1:3)
for(i in 1:length(complexity)){
  plot(x, y, col = 'grey')
  model1 = lm(y[xorder[1:50]] ~ poly(x[xorder[1:50]], i))
  model2 = lm(y[xorder[51:100]] ~ poly(x[xorder[51:100]], i))
  model3 = lm(y[xorder[101:150]] ~ poly(x[xorder[101:150]], i))
  lines(x[xorder[1:50]], predict(model1), col = 'blue', lwd = 2)
  lines(x[xorder[51:100]], predict(model2), col = 'blue', lwd = 2)
  lines(x[xorder[101:150]], predict(model3), col = 'blue', lwd = 2)
}
```

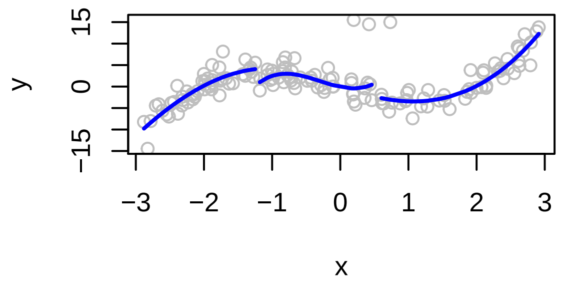


4.(c)

```r
par(mfrow = c(2, 2))
for(i in 1:length(complexity)){
  plot(x, y, col = 'grey')
  model1 = rlm(y[xorder[1:50]] ~ poly(x[xorder[1:50]], i), psi = huberfn)
  model2 = rlm(y[xorder[51:100]] ~ poly(x[xorder[51:100]], i), psi = huberfn)
  model3 = rlm(y[xorder[101:150]] ~ poly(x[xorder[101:150]], i), psi = huberfn)
  lines(x[xorder[1:50]], predict(model1), col = 'blue', lwd = 2)
  lines(x[xorder[51:100]], predict(model2), col = 'blue', lwd = 2)
  lines(x[xorder[101:150]], predict(model3), col = 'blue', lwd = 2)
}
```

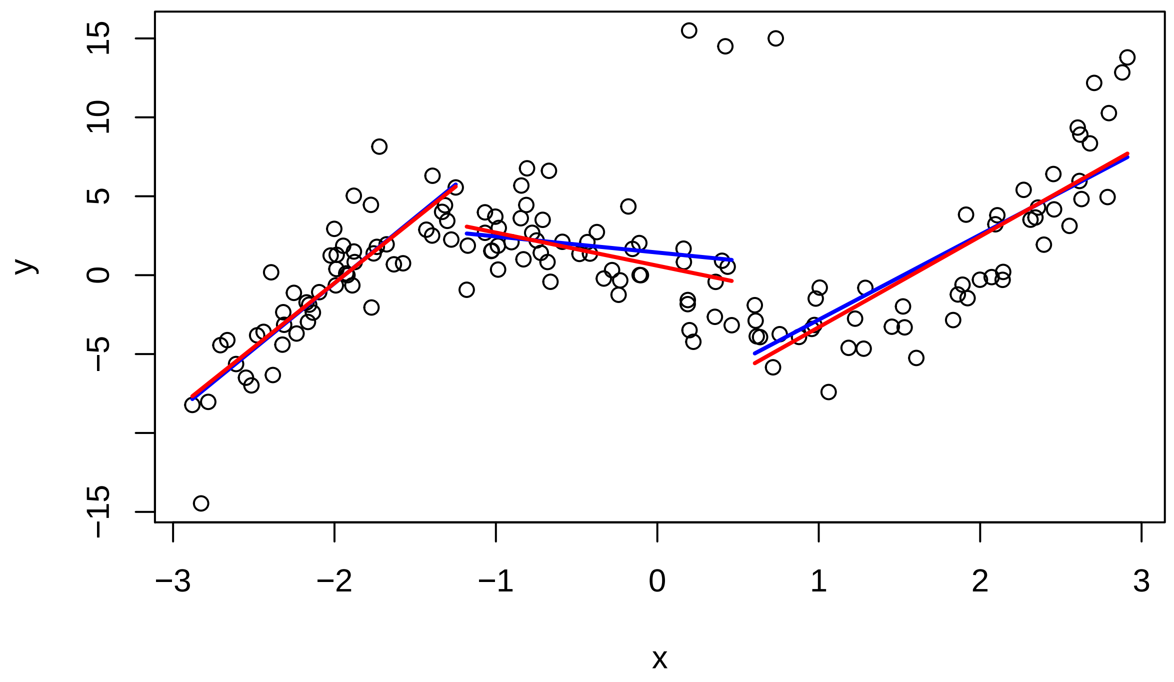


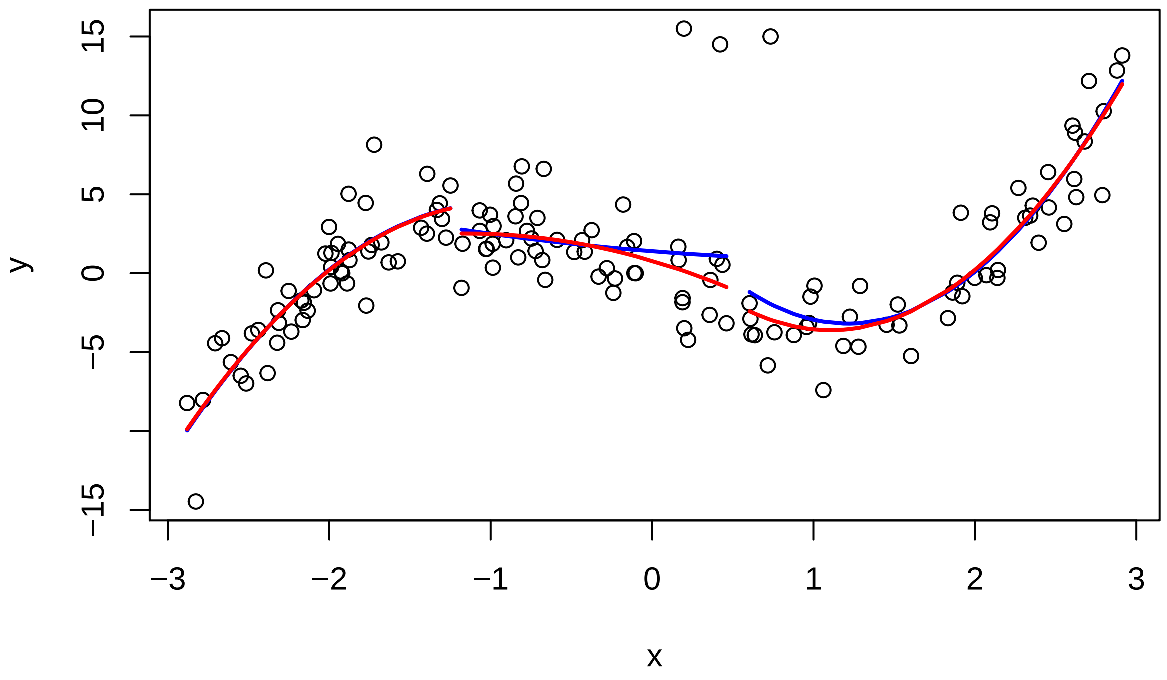


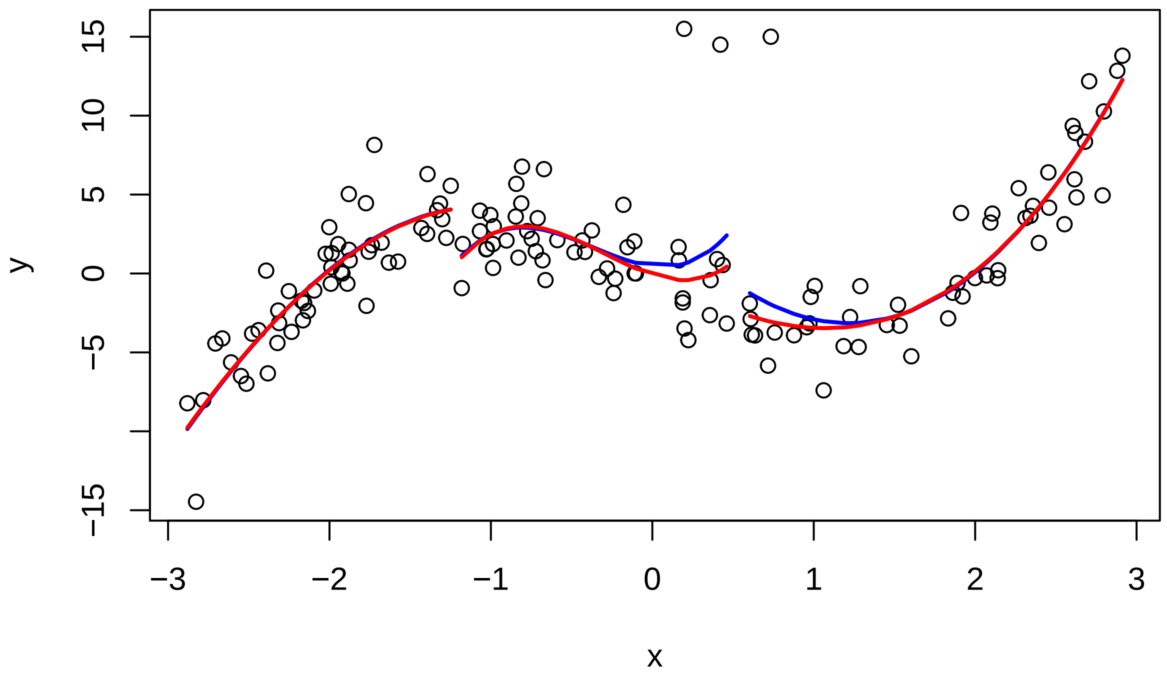


4.(d)

```r
for(i in 1:length(complexity)){
  plot(x, y)
  model1 = lm(y[xorder[1:50]] ~ poly(x[xorder[1:50]], i))
  model2 = lm(y[xorder[51:100]] ~ poly(x[xorder[51:100]], i))
  model3 = lm(y[xorder[101:150]] ~ poly(x[xorder[101:150]], i))

  rbmodel1 = rlm(y[xorder[1:50]] ~ poly(x[xorder[1:50]], i), psi = huberfn)
  rbmodel2 = rlm(y[xorder[51:100]] ~ poly(x[xorder[51:100]], i), psi = huberfn)
  rbmodel3 = rlm(y[xorder[101:150]] ~ poly(x[xorder[101:150]], i), psi = huberfn)

  lines(x[xorder[1:50]], predict(model1), col = 'blue', lwd = 2)
  lines(x[xorder[51:100]], predict(model2), col = 'blue', lwd = 2)
  lines(x[xorder[101:150]], predict(model3), col = 'blue', lwd = 2)

  lines(x[xorder[1:50]], predict(rbmodel1), col = 'red', lwd = 2)
  lines(x[xorder[51:100]], predict(rbmodel2), col = 'red', lwd = 2)
  lines(x[xorder[101:150]], predict(rbmodel3), col = 'red', lwd = 2)
}
```




y
x
15
10
5
0
−5
−15
−3
−2
−1
0
1
2
3



There isn't much different between the Huber's robust loss function and least squares on the first polynomial. For the rest of the polynomials, since Huber's robust loss function is less sensitive to outliers, the Huber's robust loss fit is further away than the least squares fit.