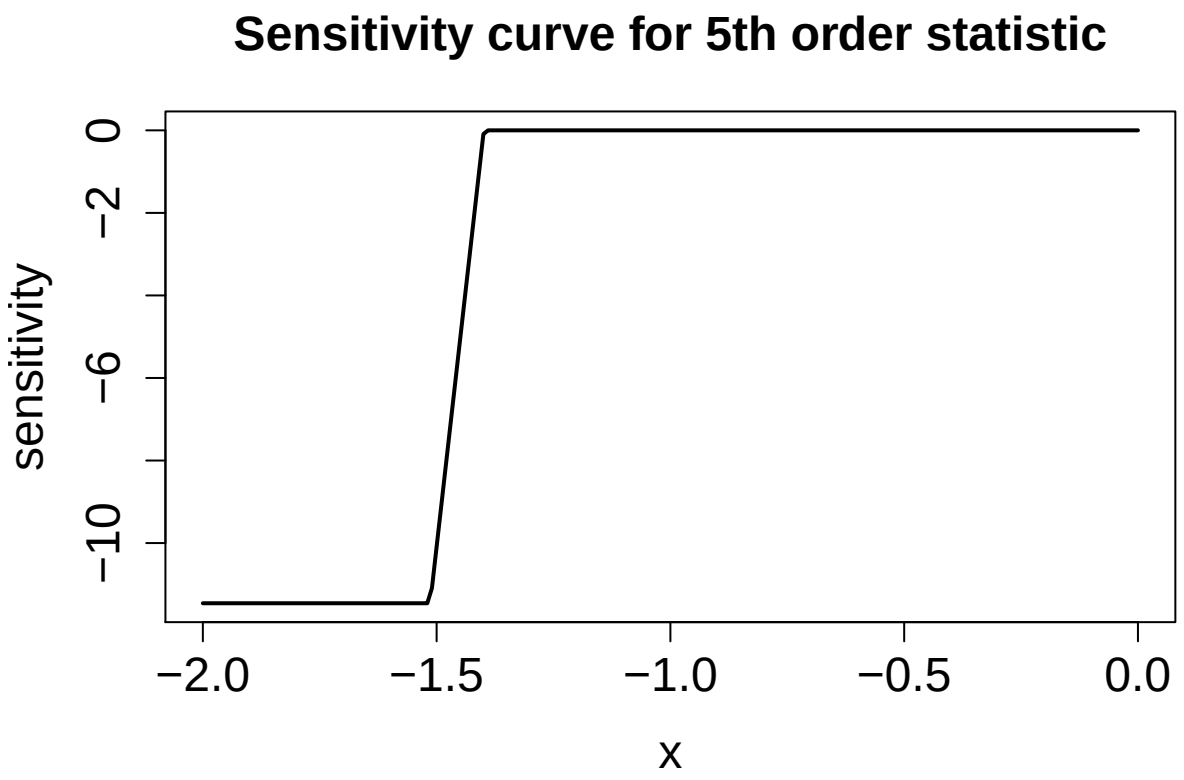
a1

1(a)

if $y \leq y_{k-1}$ then $SC(y) = \frac{y_{(k-1)} - y_{(k)}}{1/N}$
if $y \in (y_{(k-1)}, y_{(k)})$ then $SC(y) = \frac{y - y_{(k)}}{1/N}$
else $y \in (y_{(k)}, \infty)$ then $SC(y) = 0$

1(b)

```
set.seed(444)
y=rnorm(100)
y=sort(y)
sc <- function(ynew, y, k){
  N = length(y)
  if (ynew < y[k-1]){
    return (N*(y[k-1]-y[k]))
  }
  else if(ynew<y[k]){
    return(N*(ynew-y[k]))
  }else
    return(0)
}
x=seq(-2,0,0.01)
b<-function(x) sc(x,y,5)

sc_x=lapply(x, b)
plot(x, sc_x, type="l", lwd = 2,
     main="Sensitivity curve for 5th order statistic",
     ylab="sensitivity" , cex.lab=1.5,cex.axis=1.5, cex.main=1.5)
```



1(c)

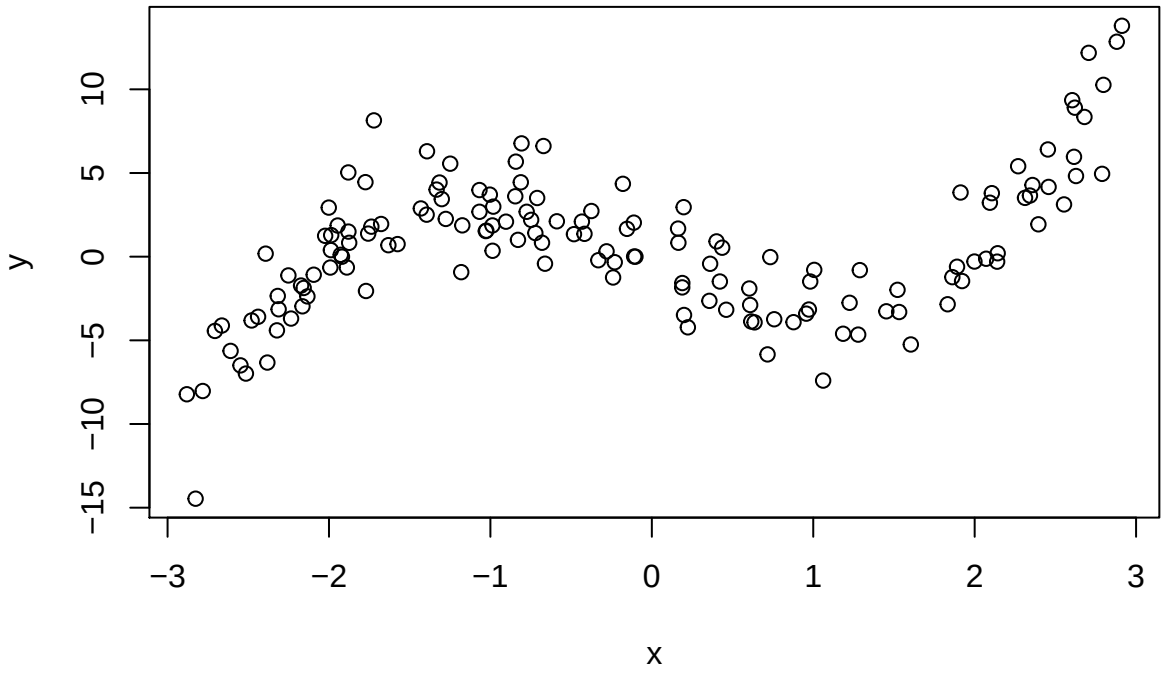
We need to change N-k+1 data to infinity so breakdown point it $\frac{N-k+1}{N}$

2(a) k-fold cv has disadvantage of computation time and advantage of reduced bias and being "fair" to all points as we use each point as test set once and as training set k times

2(b)

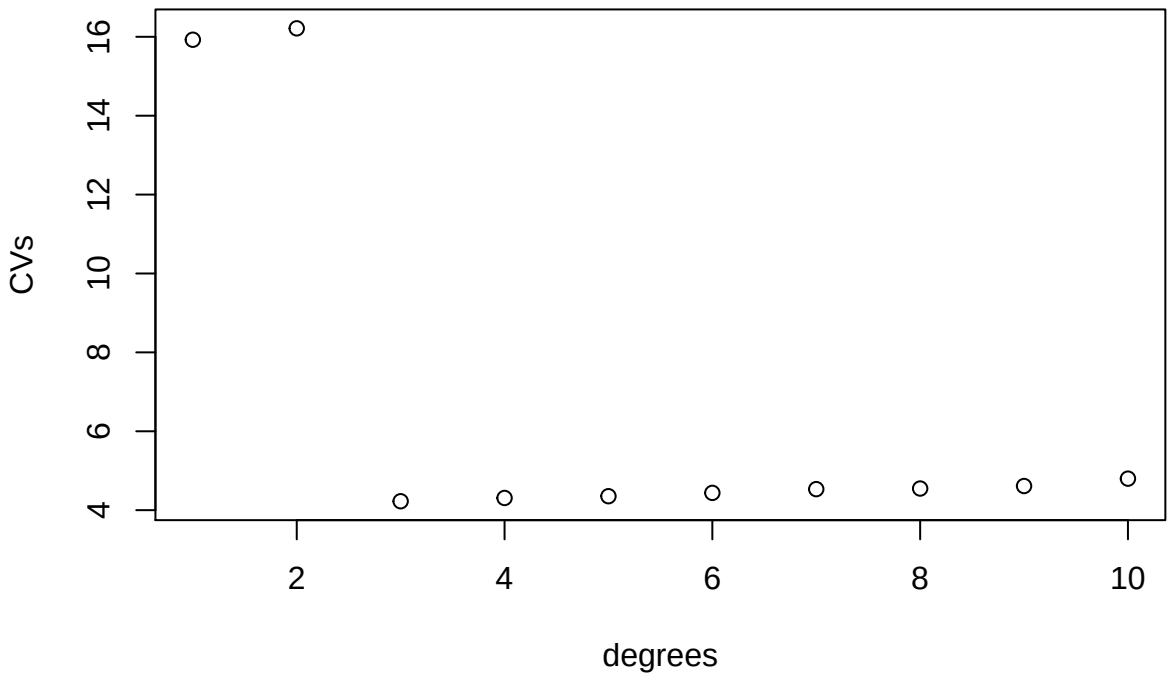
$$y = x^3 - 4x$$

```
set.seed(444)
x = runif(150,-3,3)

y = (x+2)*(x)*(x-2) + rnorm(150,sd=2)
plot(x,y)
```



2(c)

```
library(boot)
data = data.frame(x,y)
CVs=c()
set.seed(444)
for (i in 1:10){
  glm.model=glm(y~poly(x,i), data=data)
  CV=cv.glm(data, glm.model)
  CVs[i]=CV$delta[1]
}
degrees=seq(1,10)
plot(degrees,CVs)
```




```
which.min(CVs)
```

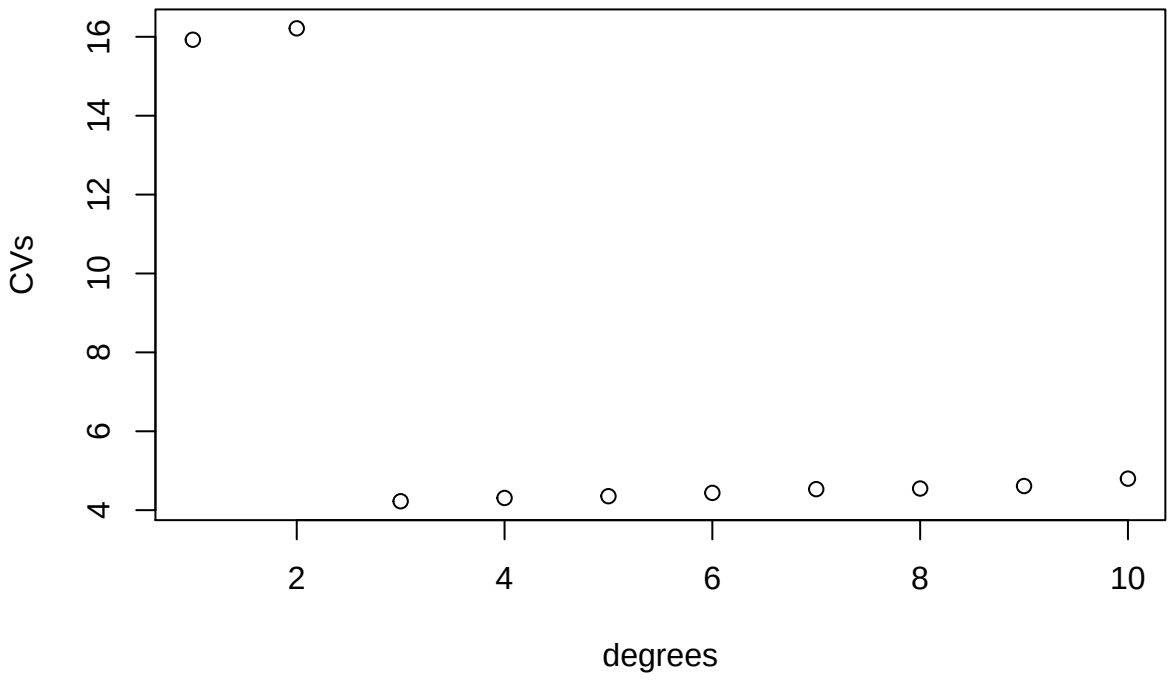
```
## [1] 3
```

```
# help(cv.glm)
# help(glm)
```

we see that degree 3 is the best model

2(d)

```
set.seed(844)
for (i in 1:10){
  glm.model=glm(y~poly(x,i), data=data)
  CV=cv.glm(data, glm.model)
  CVs[i]=CV$delta[1]
}
degrees=seq(1,10)
plot(degrees,CVs)
```




```
which.min(CVs)
```

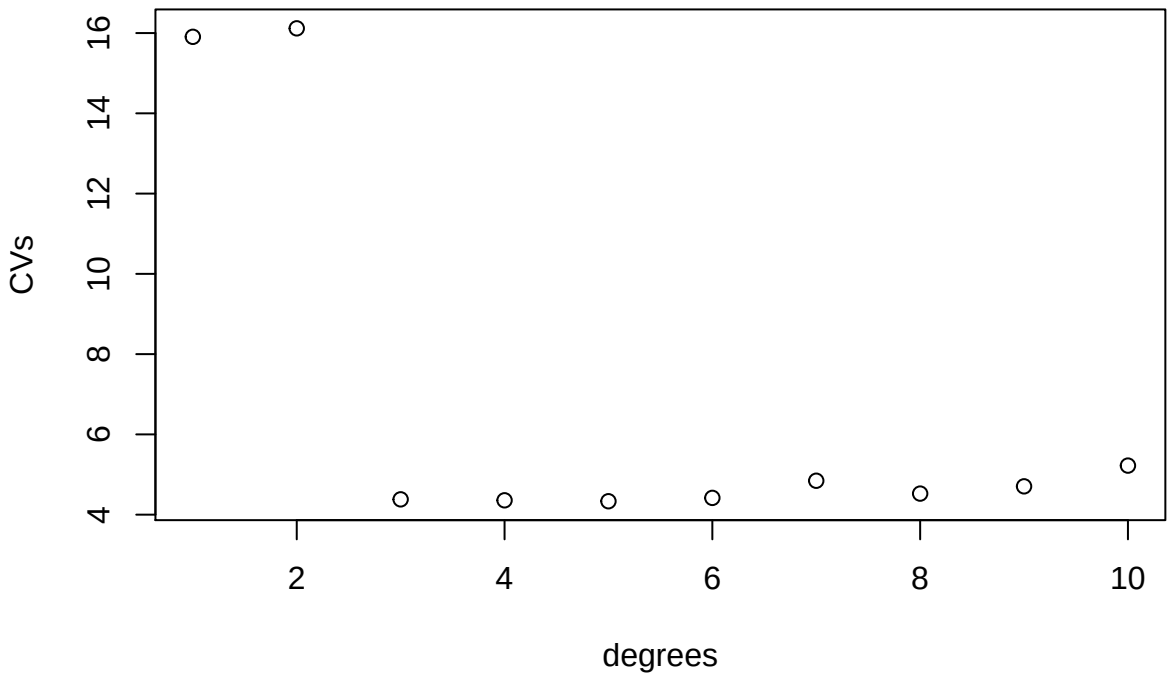
```
## [1] 3
```

We see the best is still degree 3

There is no difference as expected, as no matter what ramdom number generator is, LOOCV is forming the same set of testsets and trainingsets. It does not affect how each set is composed of as k-fold CV.

2(e)

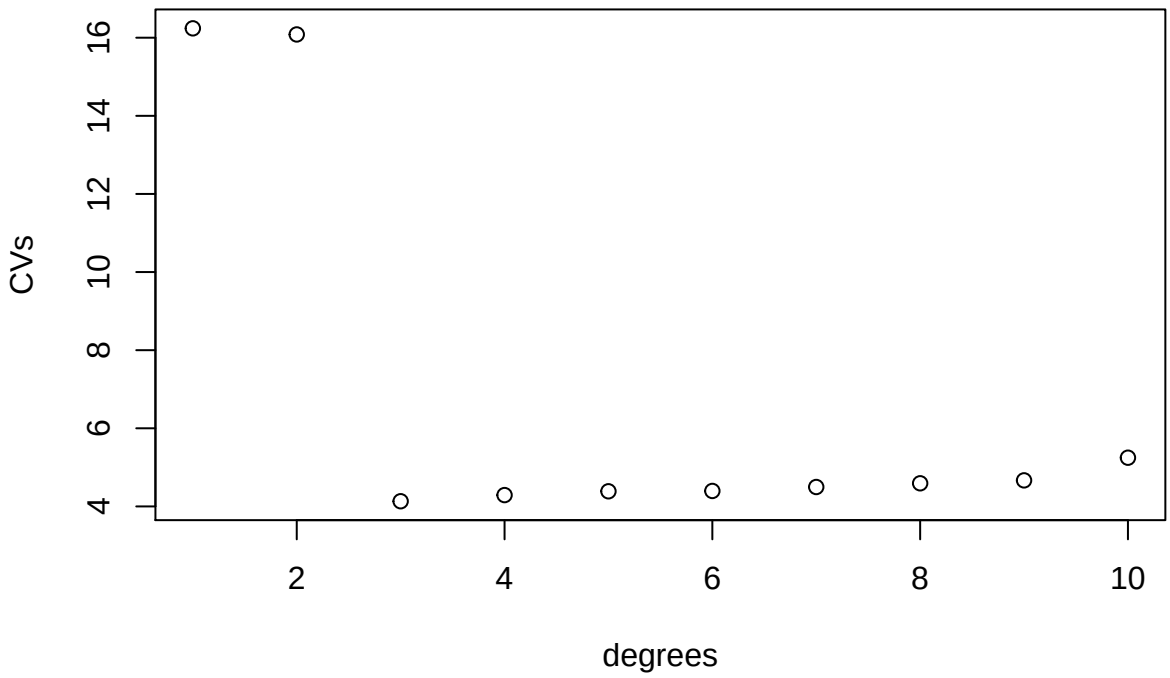
```
set.seed(444)
for (i in 1:10){
  glm.model=glm(y~poly(x,i), data=data)
  CV=cv.glm(data, glm.model,K=10)
  CVs[i]=CV$delta[1]
}
degrees=seq(1,10)
plot(degrees,CVs)
```




```
which.min(CVs)
```

```
## [1] 5
```

```
set.seed(844)
for (i in 1:10){
  glm.model=glm(y~poly(x,i), data=data)
  CV=cv.glm(data, glm.model,K=10)
  CVs[i]=CV$delta[1]
}
degrees=seq(1,10)
plot(degrees,CVs)
```




```
which.min(CVs)
```

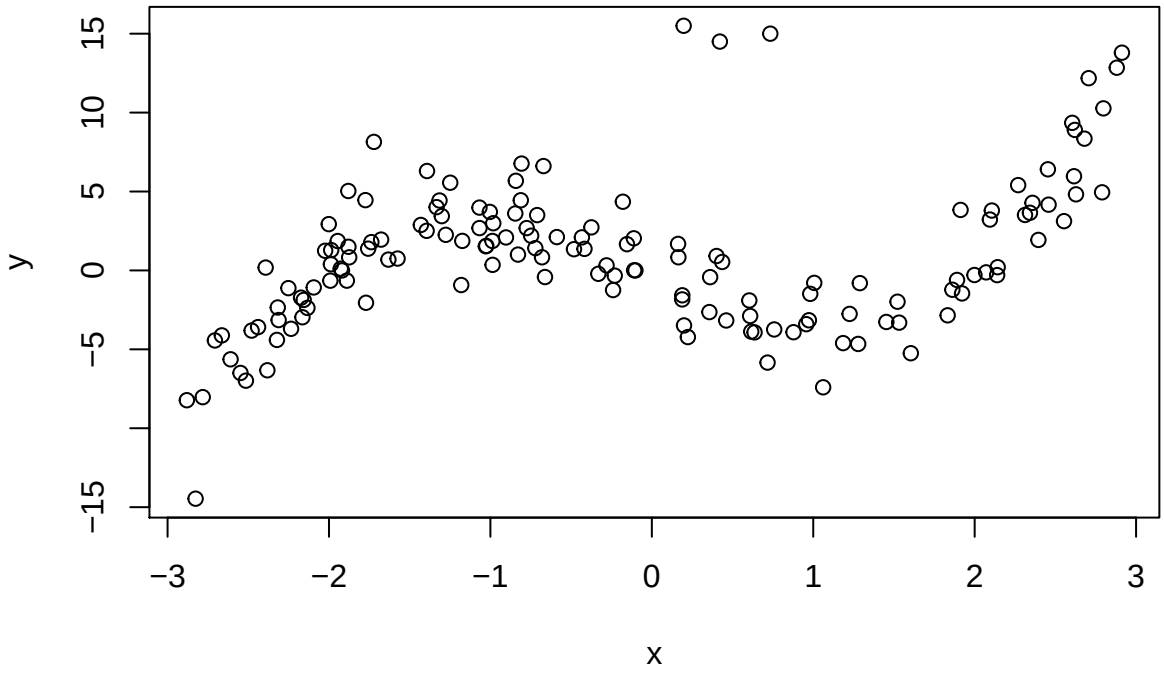
```
## [1] 3
```

We choose degree 5 at seed 444 and degree 3 at seed 844. As expected, changing random seed change the value of CV_{10}. This is because the 10 groups, which are rollingly used as training and test sets are composed by different datapoint in the two random seed. Ex: If data in a group is more correlated (in the wrong way) than expected, we may have bad CV score.

2(f) we find that although we get the right degree (3) for most seeds and k, above degree 3, the CV scores are not very much different. This agree with our model as the true degree is 3 but noise is very much. Because of the large noice, the punishment of over fitting is not very significant, the real data may look just like the overfitted polynomial.

3(a)

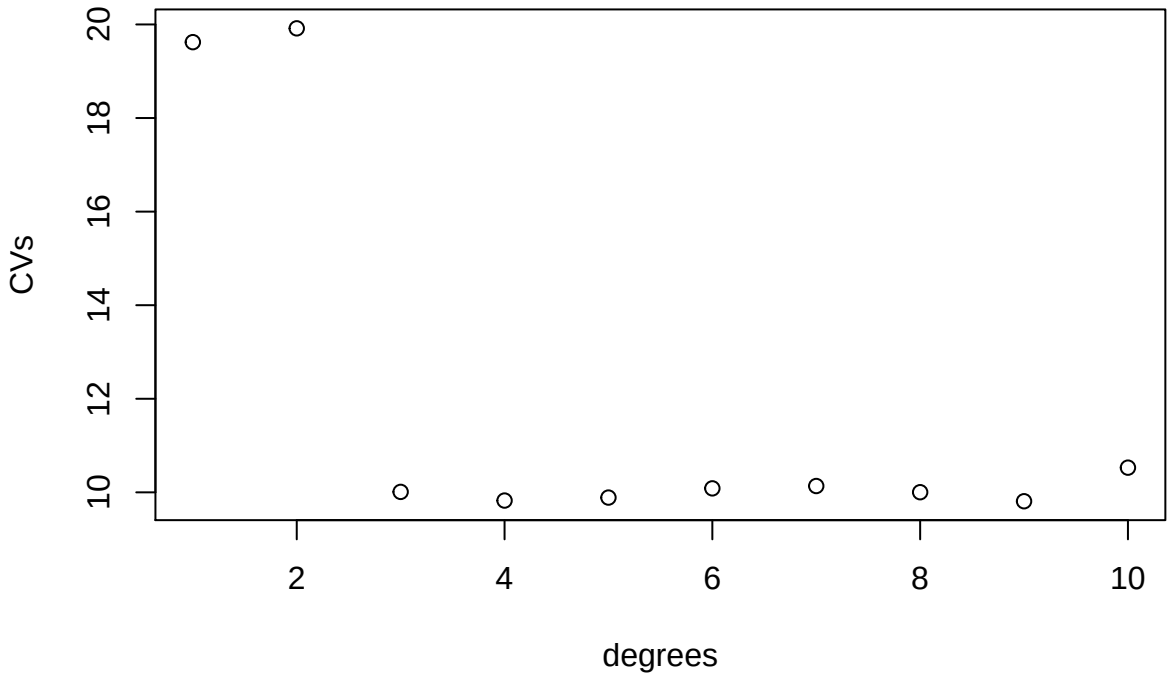
```
set.seed(444)
x = runif(150,-3,3)
y = (x+2)*(x)*(x-2) + rnorm(150,sd=2)
Extreme.Indx = c(19 , 45 , 124)
y[Extreme.Indx] = c(15.5 , 14.5 , 15)
index=order(x)
x=x[index]
y=y[index]
plot(x,y)
```




```
data = data.frame(x,y)
```

3(b)

```
set.seed(444)
CVs=c()
for (i in 1:10){
  glm.model=glm(y~poly(x,i), data=data)
  CV=cv.glm(data, glm.model,K=10)
  CVs[i]=CV$delta[1]
}
degrees=seq(1,10)
plot(degrees,CVs)
```



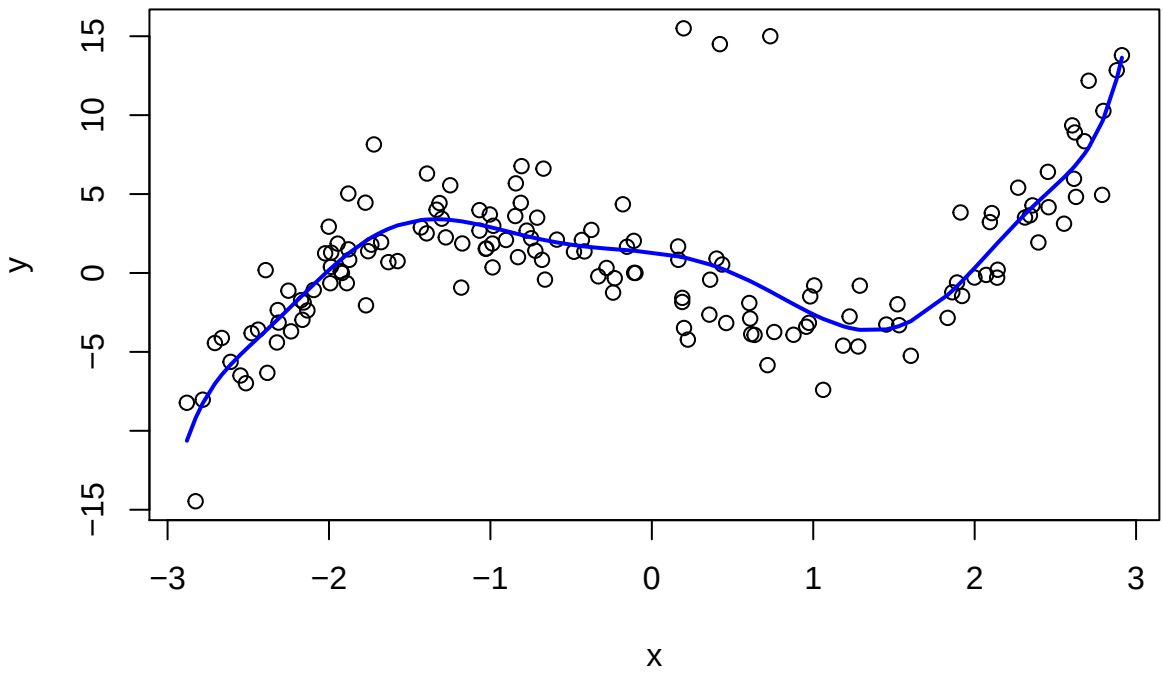

```
which.min(CVs)
```

```
## [1] 9
```

```
glm_9=glm(y~poly(x,9), data=data)
glm_9$coefficients
```

```
## (Intercept) poly(x, 9)1 poly(x, 9)2 poly(x, 9)3 poly(x, 9)4 poly(x, 9)5
##   0.8849621  19.7095868  -0.3411478  38.4914044   3.1915946   0.9058450
## poly(x, 9)6 poly(x, 9)7 poly(x, 9)8 poly(x, 9)9
##  -1.6214789  -3.1036242   2.4593905   5.5129819
```

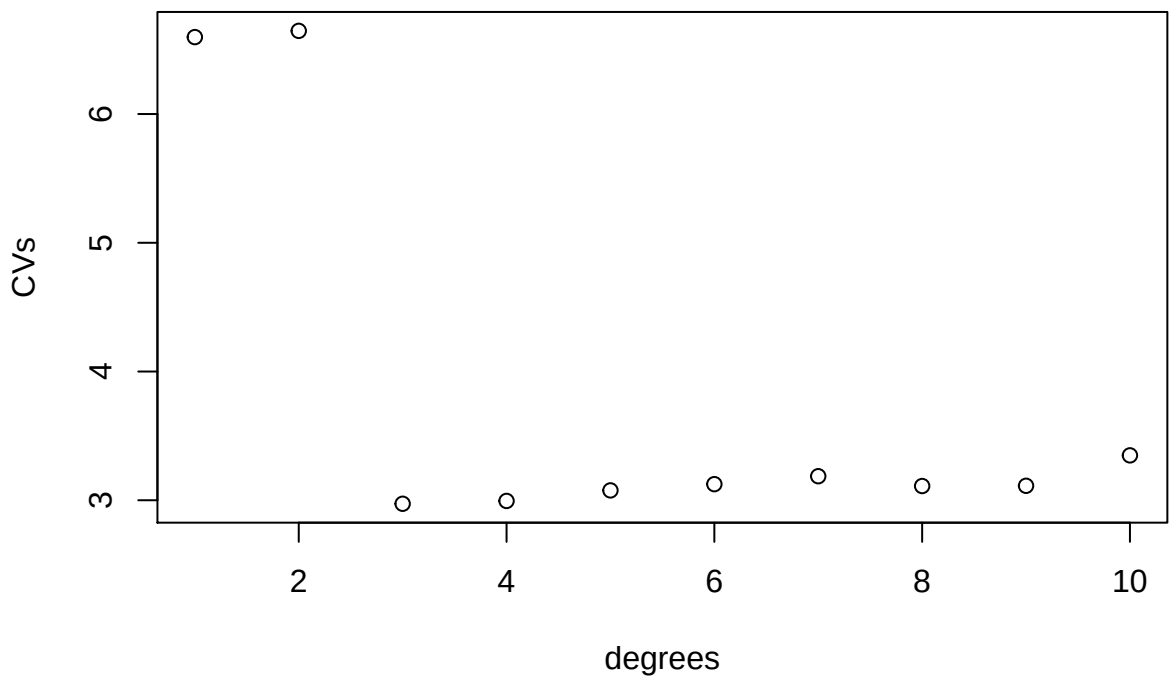
```
plot(x,y)
lines(x, predict(glm_9), type="l", col="blue", lwd=2)
```



3(c)

```
set.seed(444)
huber<- function(x, k=3){
  if (abs(x)<=3){
    return ((x)^2/2)
  }else
    return(3*(abs(x)-1.5))
}

cost <- function(obs, pred){
  resid = obs-pred
  weighted = lapply(resid, function(x) huber(x,k=3))
  weighted = unlist(weighted)
  return(mean(weighted))
}




for (i in 1:10){
  # rlm.model = rlm(y~poly(x,i), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k2 =3, maxit=10
  glm.model = glm(y~poly(x, i), data=data)
  CV=cv.glm(data, glm.model,K=10, cost=cost)
  CVs[i]=CV$delta[1]
}
degrees=seq(1,10)
```

```
plot(degrees,CVs)
```

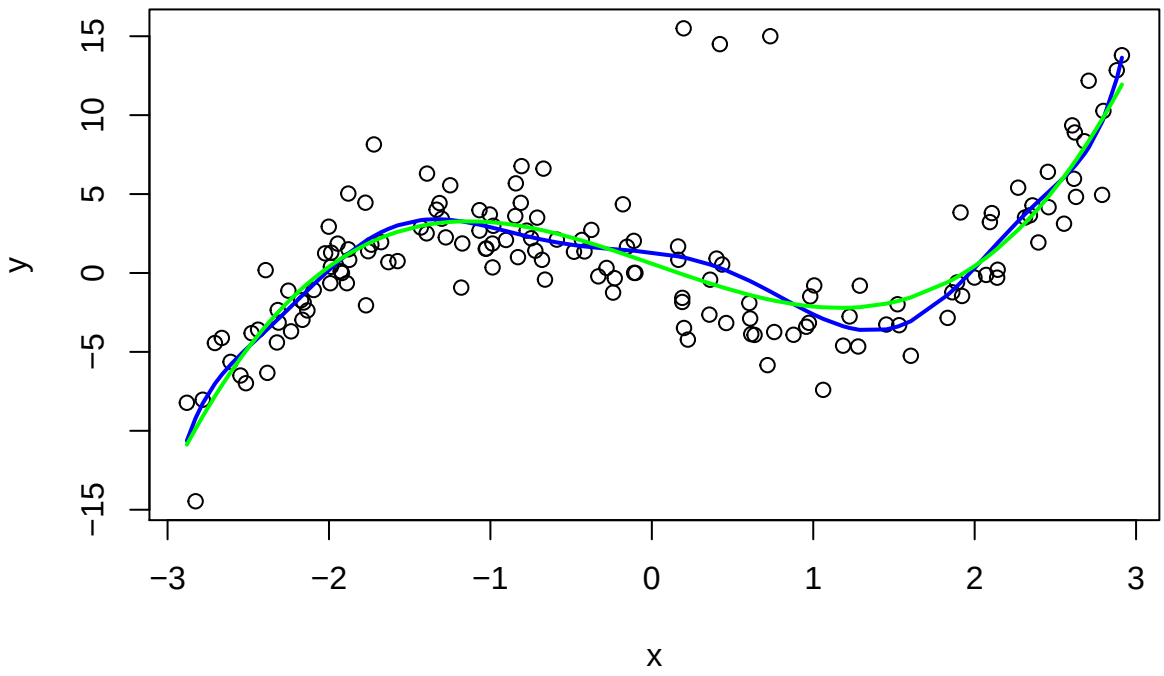



```
which.min(CVs)
```

```
## [1] 3
```

(d)

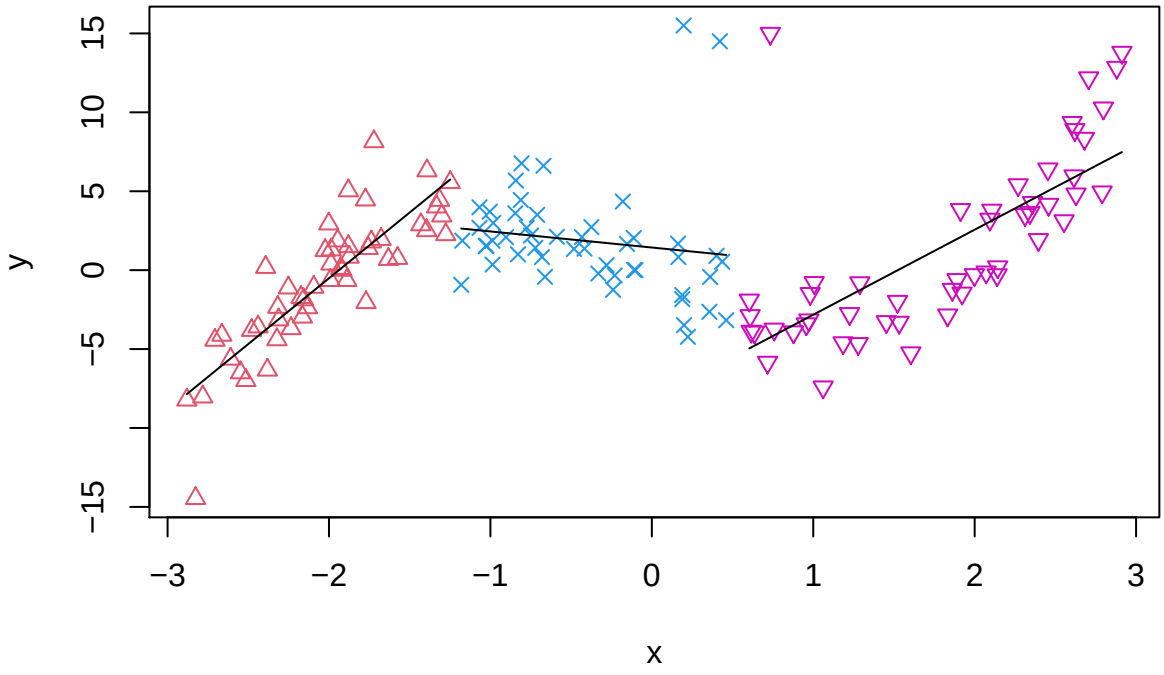
```
model_b=glm(y~poly(x, 9), data=data)
model_c=glm(y~poly(x, 3), data=data)
plot(x,y)
lines(x, predict(model_b),col="blue", lwd=2)
lines(x, predict(model_c),col="green", lwd=2)
```

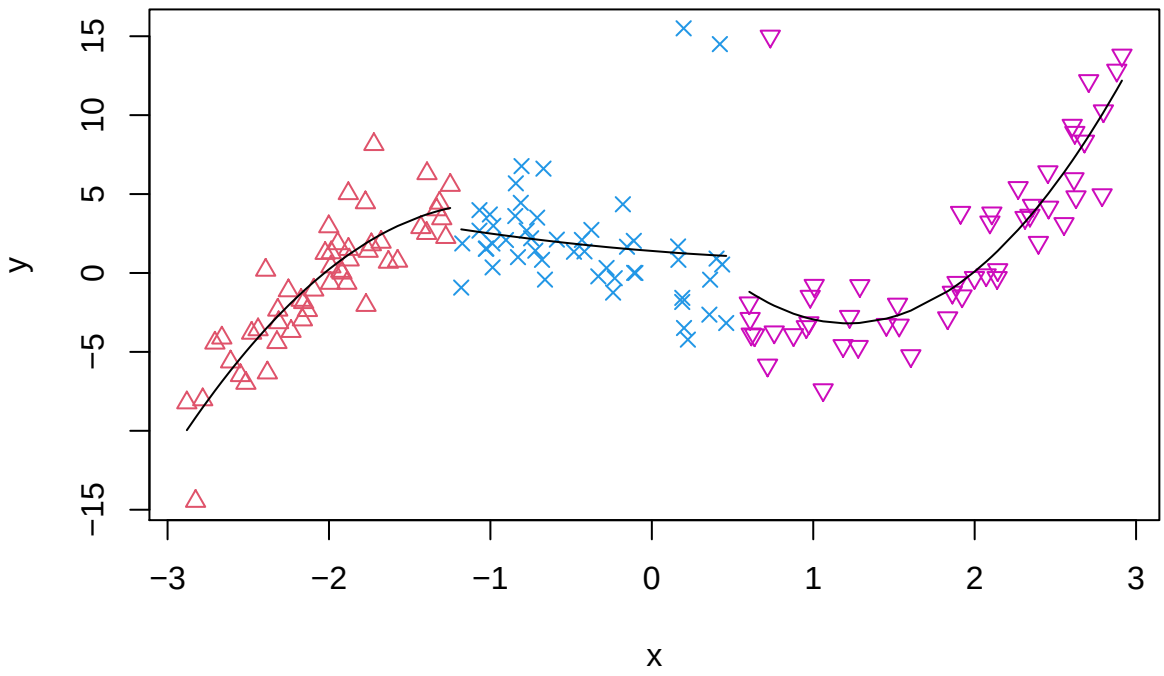


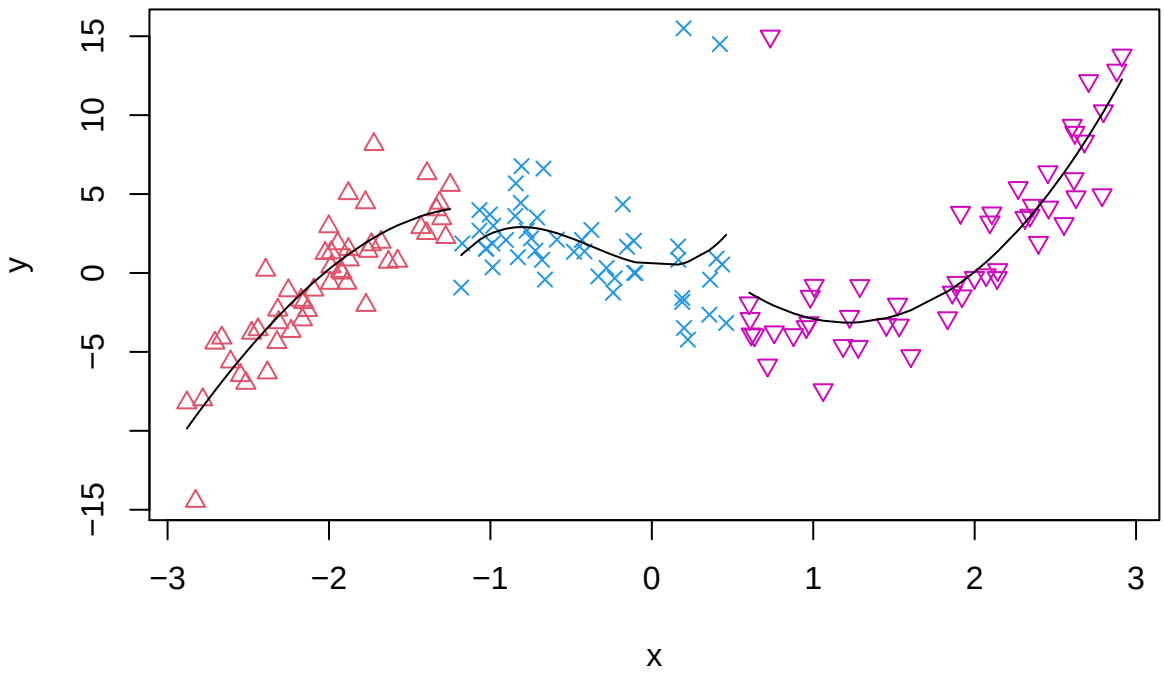
we observe using Huber as cost function, the regression try less hard to fit extreme values. That is because it's less expensive to have large $y - \hat{y}$.

4(ab)

```
# because we've already sorted the data...
point_type = c(rep(2,50), rep(4,50), rep(6,50))
col_type = c(rep(2,50), rep(4,50), rep(6,50))


plot(x,y,pch=point_type, col=col_type)
zone1.x=x[1:50]
zone1.y=y[1:50]
zone2.x=x[51:100]
zone2.y=y[51:100]
zone3.x=x[101:150]
zone3.y=y[101:150]
for (d in 1:3){
  plot(x,y,pch=point_type, col=col_type)
  model = predict(glm(zone1.y~poly(zone1.x,degree = d)))
  lines(zone1.x, model)
  model = predict(glm(zone2.y~poly(zone2.x,degree = d)))
  lines(zone2.x, model)
  model = predict(glm(zone3.y~poly(zone3.x,degree = d)))
  lines(zone3.x, model)

}
```




y
15
10
5
0
−5
−15
−3
−2
−1
0
1
2
3
x



4(c)

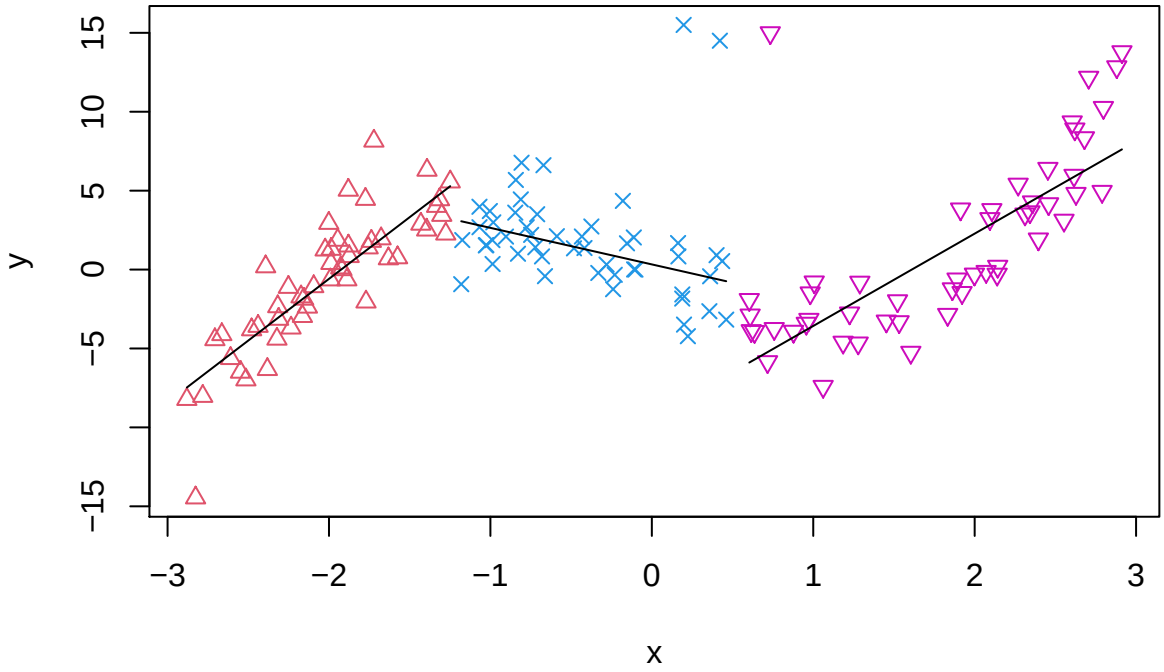
```
point_type = c(rep(2,50), rep(4,50), rep(6,50))
col_type = c(rep(2,50), rep(4,50), rep(6,50))

plot(x,y,pch=point_type, col=col_type)
zone1.x=x[1:50]
zone1.y=y[1:50]
zone2.x=x[51:100]
zone2.y=y[51:100]
zone3.x=x[101:150]
zone3.y=y[101:150]
library(MASS)
```

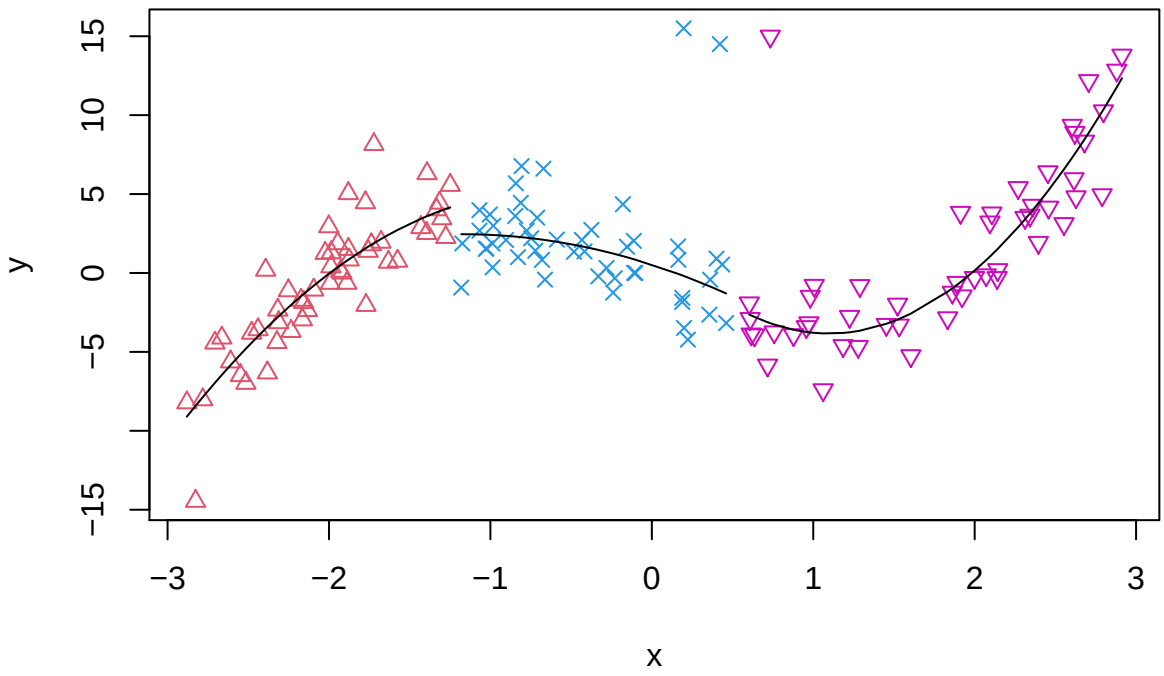
```
##
## Attaching package: 'MASS'
```

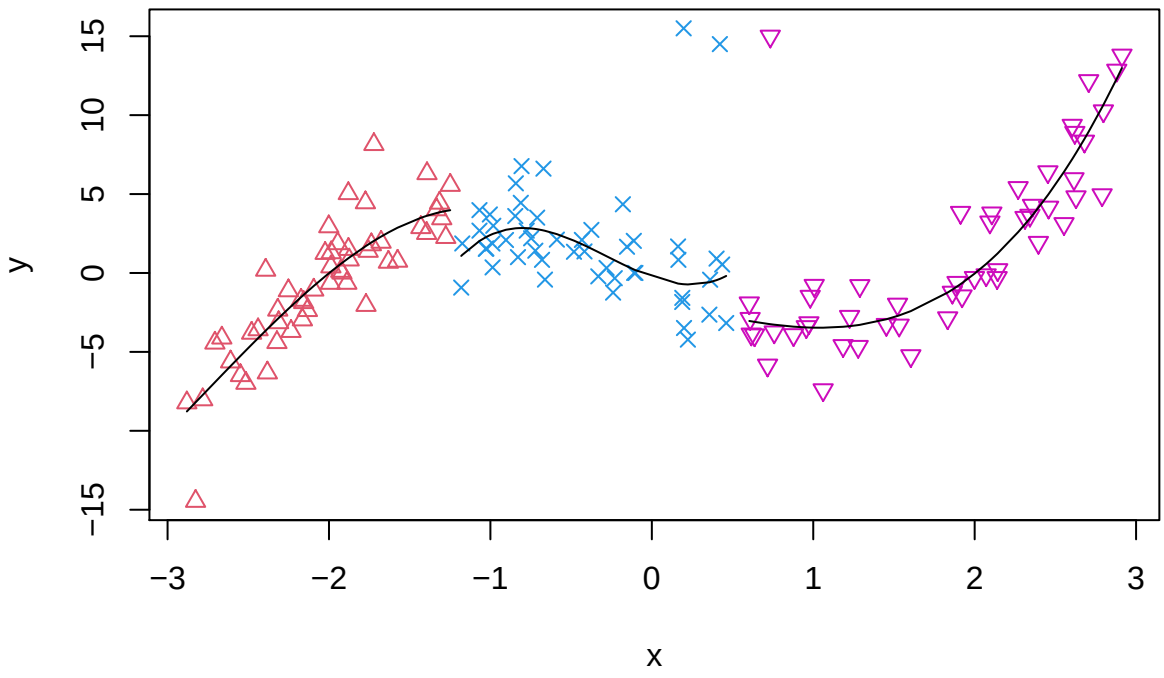
```
## The following object is masked _by_ '.GlobalEnv':
##
##     huber
```

```
for (d in 1:3){
  plot(x,y,pch=point_type, col=col_type)
  model = predict(rlm(zone1.y~poly(zone1.x,d), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k
  lines(zone1.x, model)
  model = predict(rlm(zone2.y~poly(zone2.x,d), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k
  lines(zone2.x, model)
  model = predict(rlm(zone3.y~poly(zone3.x,d), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k
```

```
  lines(zone3.x, model)

}
```



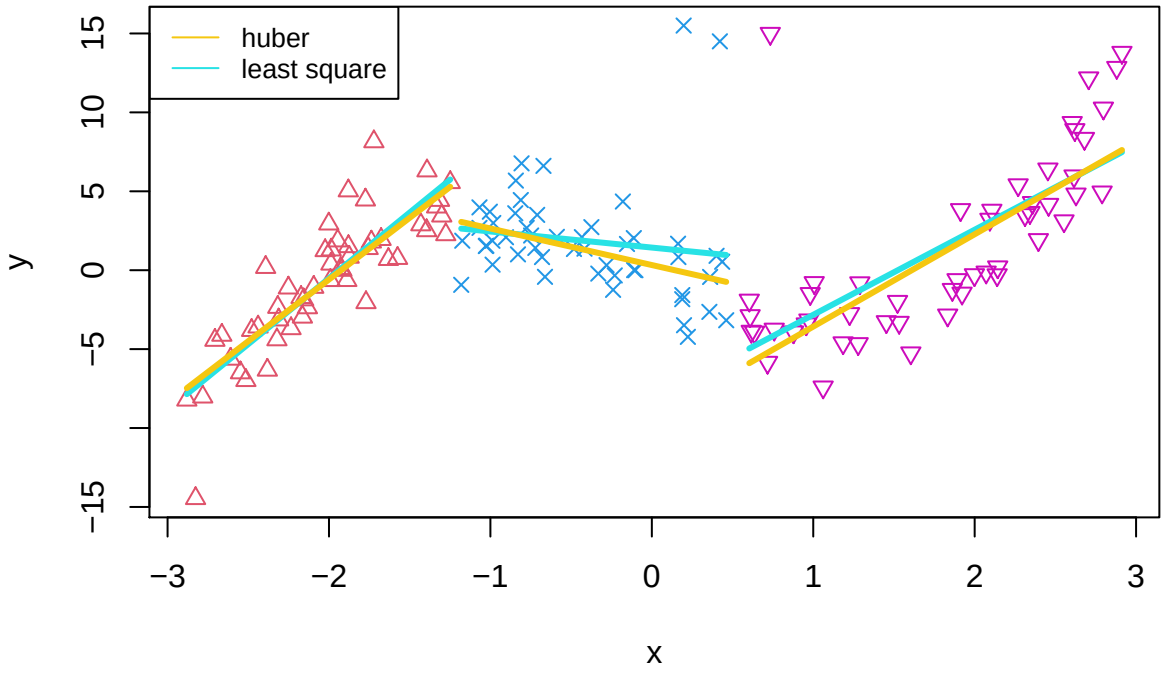

y
15
10
5
0
−5
−15
−3
−2
−1
0
1
2
3
x

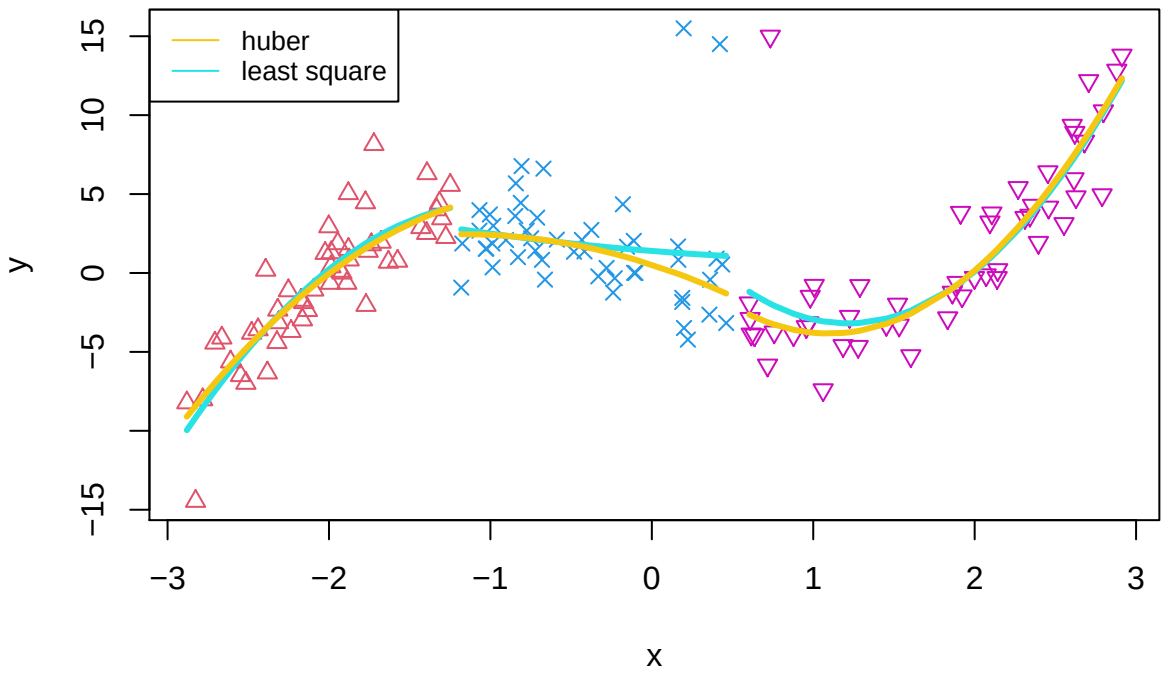


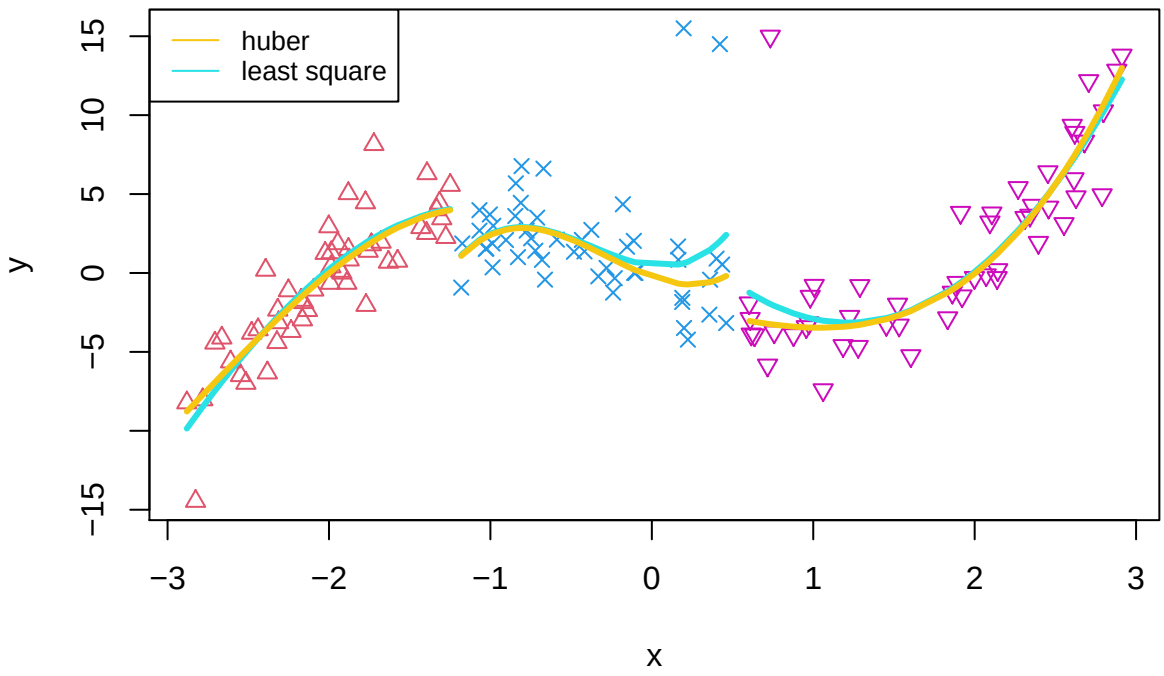
4(d)

```
point_type = c(rep(2,50), rep(4,50), rep(6,50))
col_type = c(rep(2,50), rep(4,50), rep(6,50))


plot(x,y,pch=point_type, col=col_type)
zone1.x=x[1:50]
zone1.y=y[1:50]
zone2.x=x[51:100]
zone2.y=y[51:100]
zone3.x=x[101:150]
zone3.y=y[101:150]
for (d in 1:3){
  plot(x,y,pch=point_type, col=col_type)
  model = predict(glm(zone1.y~poly(zone1.x,degree = d)))
  lines(zone1.x, model,col=5,lwd=3)
  model = predict(glm(zone2.y~poly(zone2.x,degree = d)))
  lines(zone2.x, model,col=5,lwd=3)
  model = predict(glm(zone3.y~poly(zone3.x,degree = d)))
  lines(zone3.x, model,col=5,lwd=3)



  model = predict(rlm(zone1.y~poly(zone1.x,d), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k
  lines(zone1.x, model,col=7,lwd=3)
  model = predict(rlm(zone2.y~poly(zone2.x,d), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k
```

```
  lines(zone2.x, model,col=7,lwd=3)
  model = predict(rlm(zone3.y~poly(zone3.x,d), data=data,scale.est = c("MAD", "Huber", "proposal 2"), k
  lines(zone3.x, model,col=7,lwd=3)
  legend("topleft",legend=c("huber", "least square"),col=c(7, 5), lty=c(1,1), cex=0.8)

}
```




huber
least square
y
x
15
10
5
0
−5
−15
−3
−2
−1
0
1
2
3



we observe least square would try to overfit the extreme value, which is undesired. huber fit is more stable, causing more bias less variance.