MA 750 HW 5

Benjamin Draves

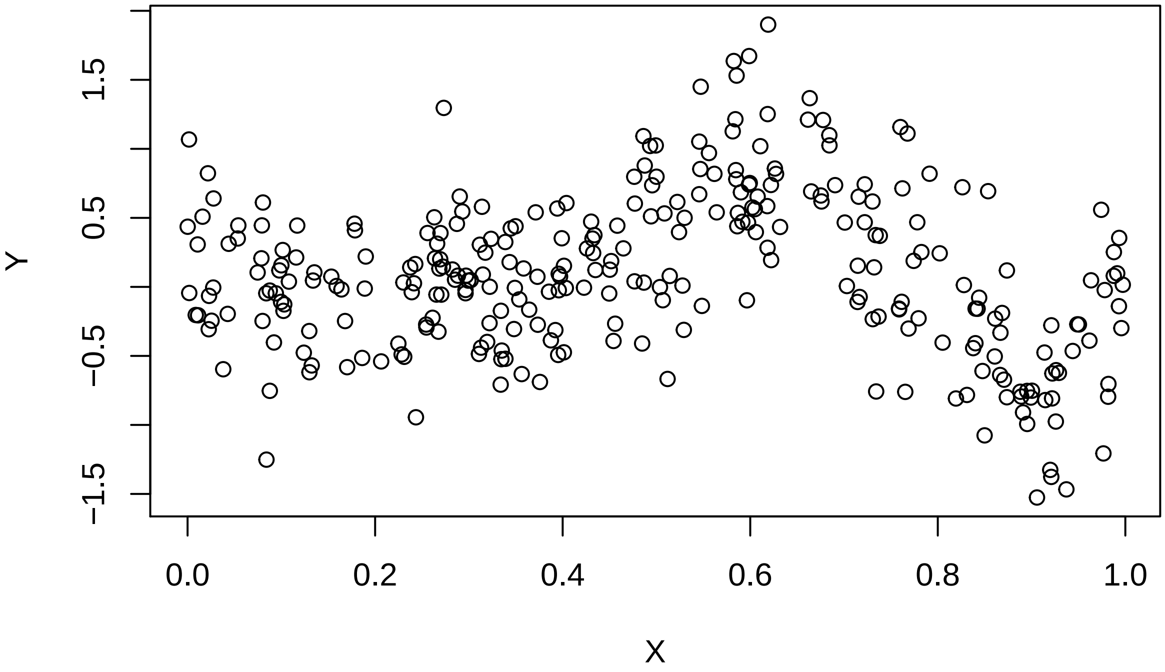
November 14

Exercise 5.1

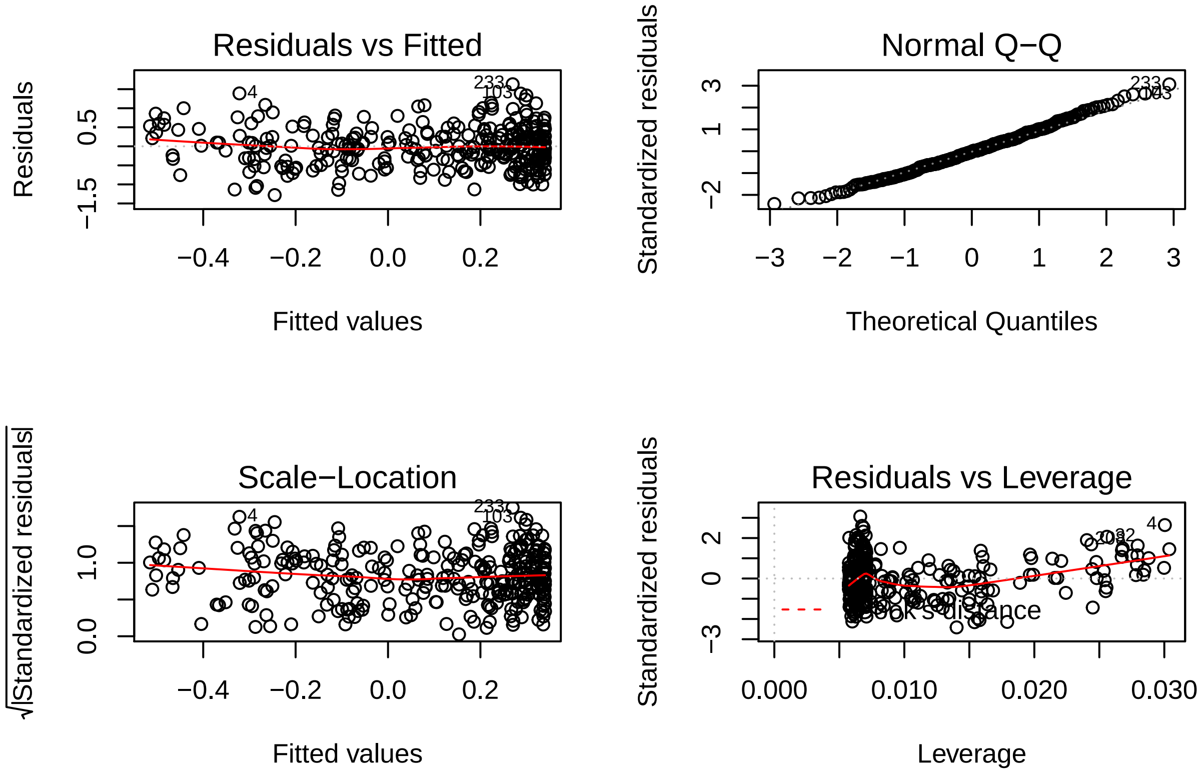
```r
#generate data
X = runif(300)
e = rnorm(300, mean = 0, sd = sqrt(0.2))
Y = sin(2*pi*X^3)^3 + e
```

(a)

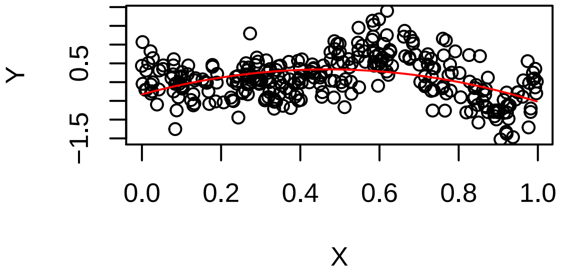
```r
#build parametric model
plot(X, Y)
```



```r
m = lm(Y~ poly(X, 2))
par(mfrow = c(2,2))
plot(m)
```



```
#build parametric model
plot(X, Y)
lines(sort(X), fitted(m)[order(X)], col='red', type='l')
```



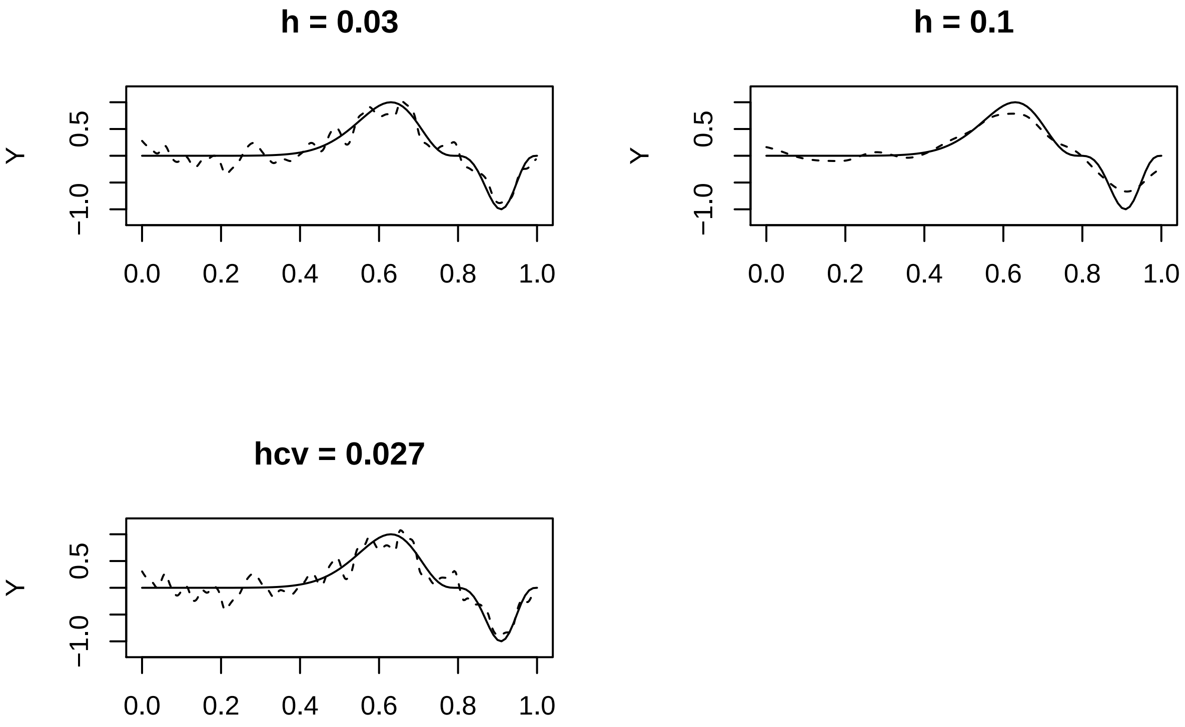
(b)

```
library(sm)
```

```
## Package 'sm', version 2.2-5.4: type help(sm) for summary information
```

```
#build kerenel regression
km.03 = ksmooth(X,Y, kernel = "normal", bandwidth = .03)
km.1 = ksmooth(X,Y, kernel = "normal", bandwidth = .1)
km.hcv = ksmooth(X,Y, kernel = "normal", bandwidth = hcv(X,Y))

#plot regression functions
par(mfrow = c(2,2))
```

```r
curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "h = 0.03", ylim = c(-1
points(km.03$x, km.03$y, type = "l", lty = 2, ylab = "Y", xlab = "X")

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "h = 0.1",ylim = c(-1.2
points(km.1$x, km.1$y, type = "l", lty = 2, ylab = "Y", xlab = "X")

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "hcv = 0.027",ylim = c(
points(km.hcv$x, km.hcv$y, type = "l", lty = 2, ylab = "Y", xlab = "X")
```



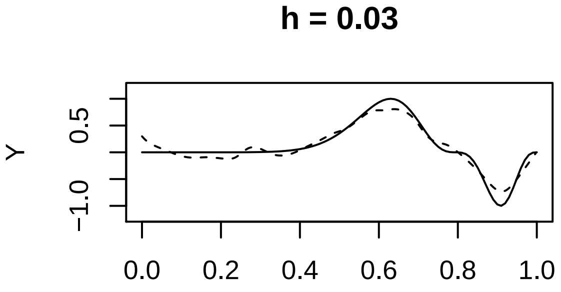
(c)

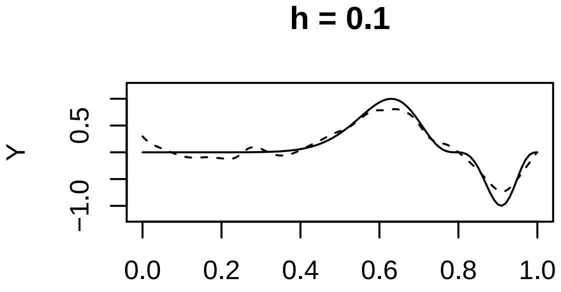
```r
library(KernSmooth)
```

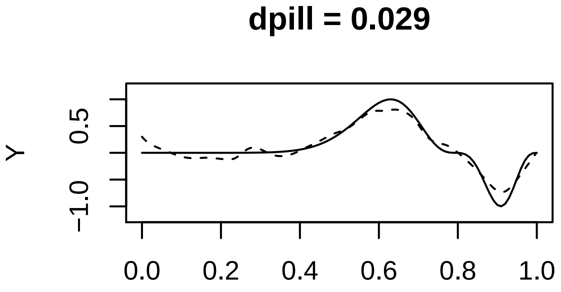
```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

```r
#build local linear
llm.03 = locpoly(X,Y, degree = 1, kernel = "normal", bandwidth = 0.03)
llm.1 = locpoly(X,Y, degree = 1, kernel = "normal", bandwidth = 0.03)
llm.dpill = locpoly(X,Y, degree = 1, kernel = "normal", bandwidth = dpill(X,Y))


#plot regression functions
par(mfrow = c(2,2))

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "h = 0.03", ylim = c(-1
points(llm.03$x, llm.03$y, type = "l", lty = 2, ylab = "Y", xlab = "X")
```

```r
curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "h = 0.1",ylim = c(-1.2
points(llm.1$x, llm.1$y, type = "l", lty = 2, ylab = "Y", xlab = "X")

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "dpill = 0.029",ylim =
points(llm.dpill$x, llm.dpill$y, type = "l", lty = 2, ylab = "Y", xlab = "X")
```







Exercise 5.2

(a)

By defining a variable bandwidth kernel via k-NN, we allow our kernel regression estimate to be more adaptive in that as we see more data in certain areas of the density, we can more accurately estimate the regression function in that region. Therefore, at the modes of the X distribution, we only consider Y points that are very close to evaluation point $f(x)$ which should improve our estimation. Compared to Kernel regression and local linear models that have fixed bandwidth sizes, we can more confidently estimate the response in these high density areas.

At the tails, or lower density areas of the X distribution, our estimation will have higher variance than the local linear or kernel regression estimates. Since we require k neighbors regardless of where the point is located, the regression function at lower density areas of the X distribution could be highly variable. In essence, we will be averaging over Y points that are not close to the evaluation point $f(Y)$. For this reason, the k-NN estimator may help near the modes of the X distribution but may suffer in lower density areas of the X distribution.

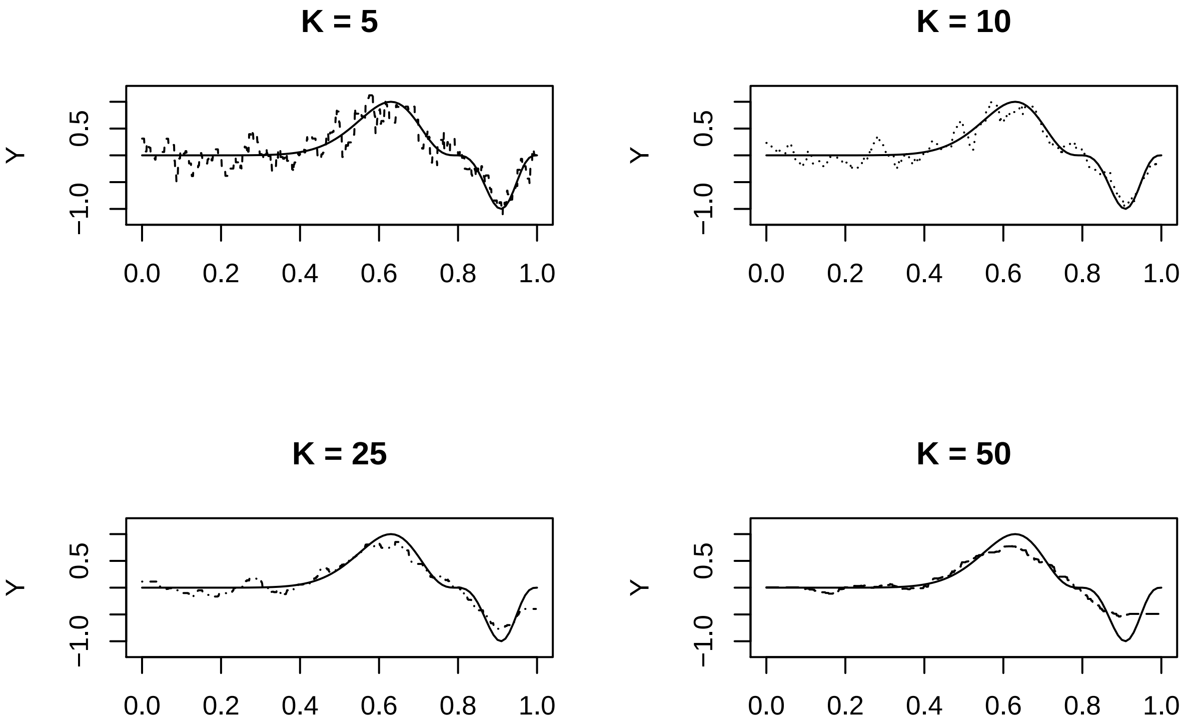
Lastly, as the weights are defined here, we are taking a simple average for the k nearest neighbors. This will produce a bumpy, non smooth function. While in some cases this is desirable, in most cases, we like to think of $f(\cdot)$ as a smooth, continuous function. This is a disadvantage as compared to local linear and kernel regression functions.

(b)

```
#function for knn for single point
knny = function(x, y, k){
  return(order(abs(x -y))[1:k])
}

#knn regression
knn_reg = function(X, Y, k){
  #set grid
  grid = unique(sort(c(seq(from = min(X), to = max(X), by = .001), X)))
  #initialize data structs
  y = numeric(length(grid))
  n = length(X)
  #find Y values for grid
  for(i in 1:length(grid)){
    #find neaest neighbors
    nn = knny(X, grid[i], k)

    #get y values
    y[i] = (1/n)*sum((n/k)*Y[nn])

  }
  mat = cbind(grid,y)
  return(mat)
}

#Symmetric function

knn_sym = function(X,Y,k){
  #get known values
  m = knn_reg(X,Y,k)

  #lay out grid as in knn_reg
  grid = unique(sort(c(X, seq(from=min(X), to = max(X), by = .001))))
  n = length(grid)

  #get i indices
  start = floor(k/2)+1
  end = n - floor(k/2)-1

  #reorder Y based on X ordering
  y = m[order(m[,1]),][,2]

  #build data structure for values
  newY = numeric(length(start:end))

  #initialize first and last values with know values
  newY[1:start] = m[1:start,2]
  newY[(end+2):length(grid)] = m[((end+2):length(grid)),2]

  #define recursive sequence
  for(i in start:end){
    newY[i+1] = newY[i] + 1/k*(y[i + floor(k/2)+1] - y[i -floor(k/2)])
  }

  return(cbind(sort(grid), y))

}
```

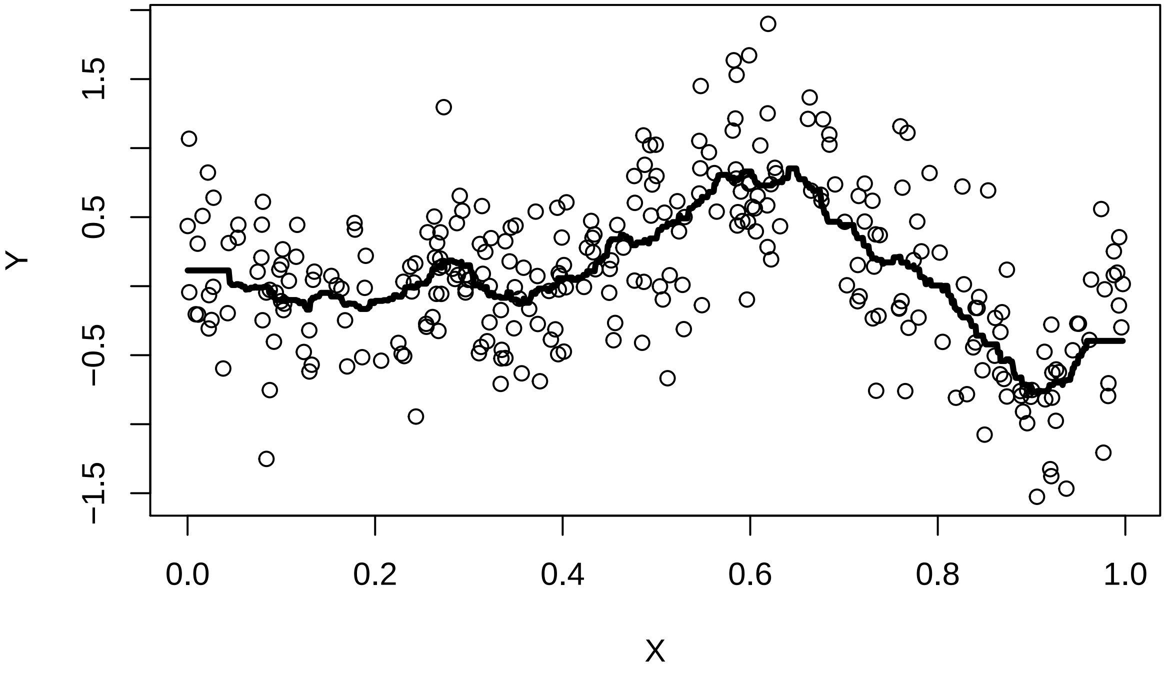
Now we will actually implement the KNN smoother.

```
#Implement KNN based on k = 5, 10, 25, 50

knn5 = knn_sym(X, Y, 5)
knn10 = knn_sym(X, Y, 10)
knn25 = knn_sym(X, Y, 25)
knn50 = knn_sym(X, Y, 50)



#plot
par(mfrow = c(2,2))
curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "K = 5", ylim = c(-1.2,
points(knn5[,1], knn5[,2], lty = 2, type = "l")

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "K = 10", ylim = c(-1.2
points(knn10[,1], knn10[,2], lty = 3, type = "l")

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "K = 25", ylim = c(-1.2
points(knn25[,1], knn25[,2], lty = 4, type = "l")

curve(sin(2*pi*x^3)^3, from = 0, to = 1, ylab = "Y", xlab = "", main = "K = 50", ylim = c(-1.2
points(knn50[,1], knn50[,2], lty = 5, type = "l")
```



It appears that $K = 25$ provides the best smoothing parameter in this case. Looking at this smoother against the data, we have the following.

```r
plot(X,Y)
points(knn25[,1], knn25[,2], type = "l", lwd = 3)
```


Y
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
X