PHS597 HW1, Spring 2022

Havell Markus

Description

- This homework contains the implementation for Projection Pursuit Regression (PPR).

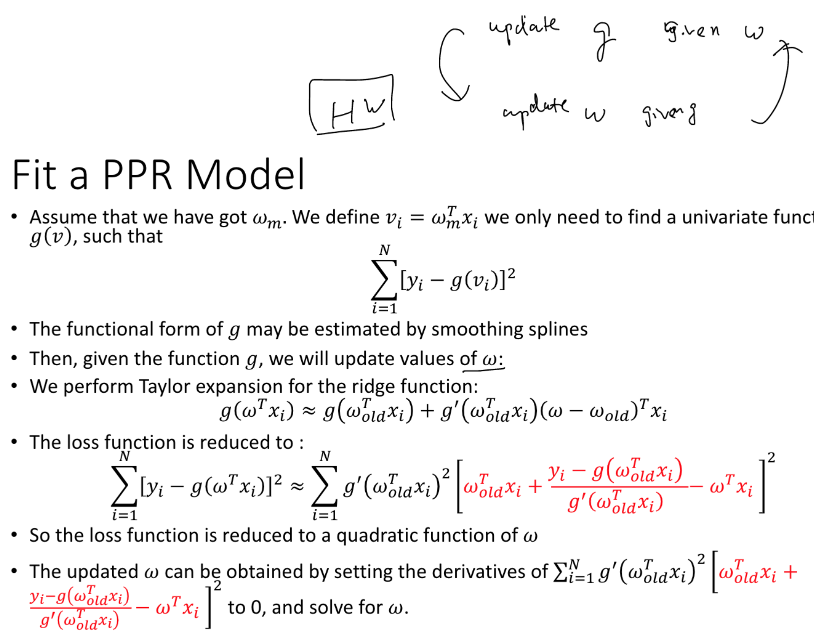
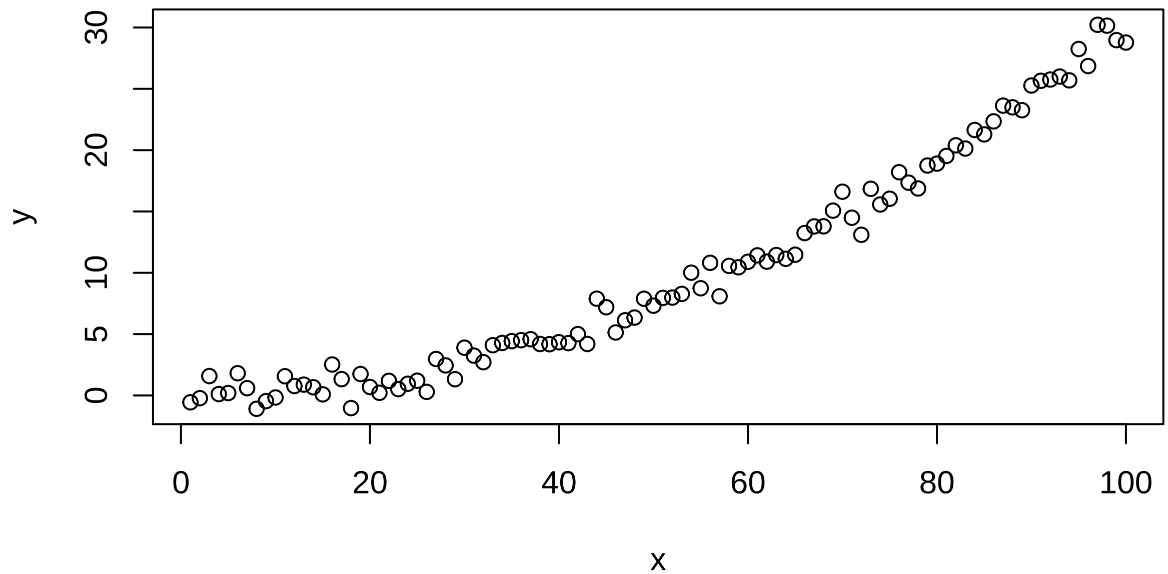


Figure 1: Lecture notes for PPR.

Simulated data

```r
library("dplyr")
library("foreach")

############################################

set.seed(123)
x <- seq(1:100)
y <- (3*(x^2) - 2*x)/1000 + rnorm(100)

plot(x, y)
```

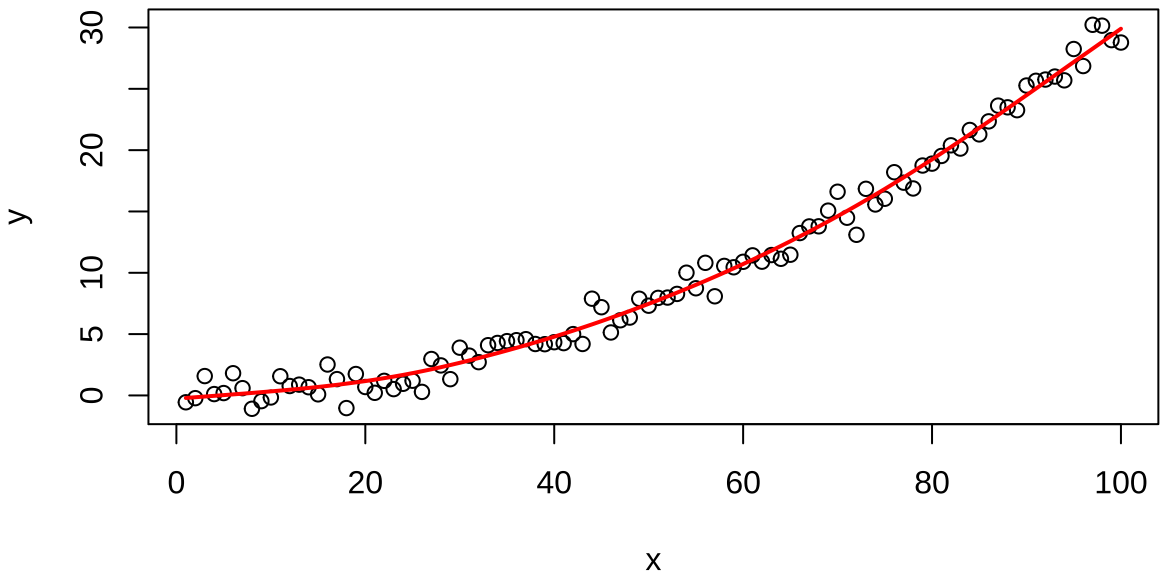


Fitting build-in PPR implementation

```
x_y.ppr <- ppr(y ~ x, data = data.frame(x=x,y=y), nterms = 1, max.terms = 1, sm.method = "spline")
summary(x_y.ppr)
```

```
## Call:
## ppr(formula = y ~ x, data = data.frame(x = x, y = y), nterms = 1,
##     max.terms = 1, sm.method = "spline")
##
## Goodness of fit:
##  1 terms
## 80.56198
##
## Projection direction vectors ('alpha'):
## [1] 1
##
## Coefficients of ridge terms ('beta'):
##   term 1
## 9.065847
##
## Equivalent df for ridge terms:
## term 1
##   5.04
```

```
plot(x, y)
points(x, x_y.ppr$fitted.values, type = "l", lwd = 2, col ="red")
```




```r
sum((x_y.ppr$fitted.values-y)^2)
```

```
## [1] 80.56198
```

Natural cubic spline implementation

- To estimate $g(v_i)$ I will first fit a natural cubic spline.
- I will use 5 knots, thus resulting in five basis functions.
- I will initiate $\omega = 0.1$, and update it 5 times

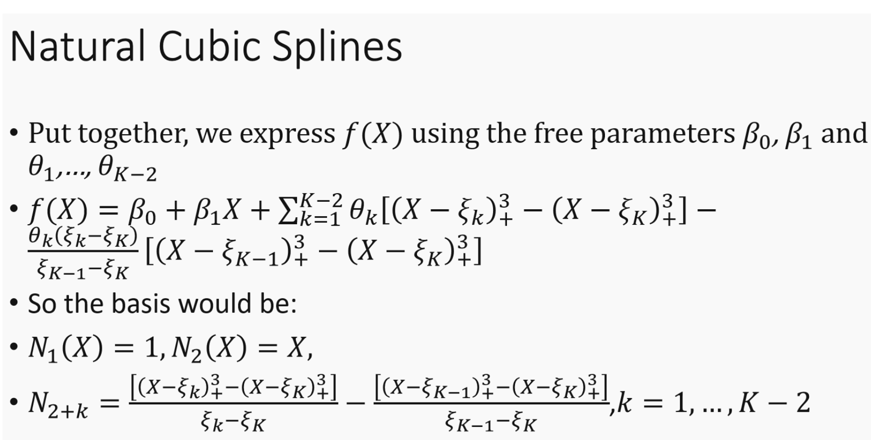
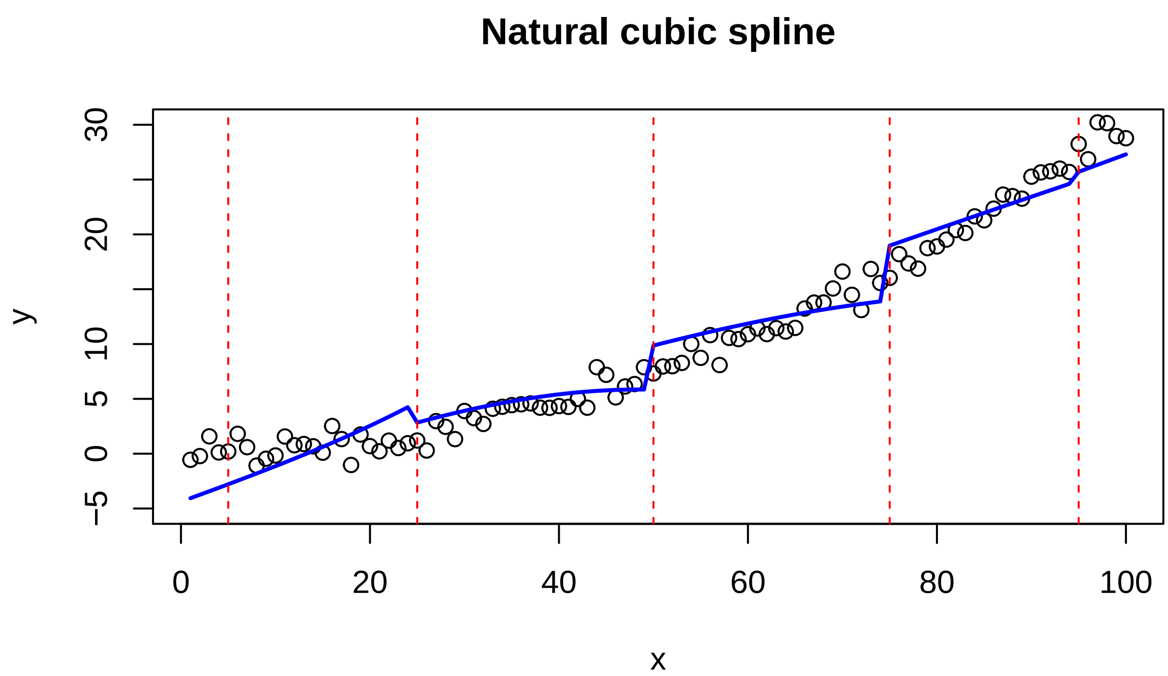


Figure 2: Lecture notes describe the basis function of natural cubic spline

```r
###########################################

basis.fxn <- function(value, thres){
  ifelse(value < thres, 0, value^3)
}

basis.fxn.sq <- function(value, thres){
  ifelse(value < thres, 0, value^2)
}

basis.prime.fxn <- function(v.v, e.knot1, e.knot2){
  (((3*basis.fxn.sq(value = v.v, thres = e.knot1))-(3*basis.fxn.sq(value = v.v, thres = e.knot2)))/(e.kn
}

###########################################
## natural cubic spline

alpha <- 0.1 %>% as.matrix()
v <- alpha %*% t(as.matrix(x))
v <- as.numeric(v)
iter <- 0

while(iter <= 5){
  e1 <- v[5] %>% as.numeric()
  e2 <- v[25] %>% as.numeric()
  e3 <- v[50] %>% as.numeric()
  e4 <- v[75] %>% as.numeric()
  e5 <- v[95] %>% as.numeric()

  n_1 <- 1
  n_2 <- v

  n_3 <- ((basis.fxn(value = v, thres = e1) - basis.fxn(value = v, thres = e5))/(e5-e1))
  n_3 <- n_3 - ((basis.fxn(value = v, thres = e4) - basis.fxn(value = v, thres = e5))/(e5-e4))

  n_4 <- ((basis.fxn(value = v, thres = e2) - basis.fxn(value = v, thres = e5))/(e5-e2))
  n_4 <- n_4 - ((basis.fxn(value = v, thres = e4) - basis.fxn(value = v, thres = e5))/(e5-e4))

  n_5 <- ((basis.fxn(value = v, thres = e3) - basis.fxn(value = v, thres = e5))/(e5-e3))
  n_5 <- n_5 - ((basis.fxn(value = v, thres = e4) - basis.fxn(value = v, thres = e5))/(e5-e4))

  lm.fit <- lm(y ~ .+0, data.frame(y=y, n_1=n_1, n_2=n_2, n_3 = n_3, n_4 = n_4, n_5=n_5))

  g_of_v <- lm.fit$fitted.values
  g_prime_v <- lm.fit$coefficients[2]
  g_prime_v <- g_prime_v + lm.fit$coefficients[3]*(basis.prime.fxn(v, e1, e5)-basis.prime.fxn(v, e4, e5
  g_prime_v <- g_prime_v + lm.fit$coefficients[4]*(basis.prime.fxn(v, e2, e5)-basis.prime.fxn(v, e4, e5
  g_prime_v <- g_prime_v + lm.fit$coefficients[5]*(basis.prime.fxn(v, e3, e5)-basis.prime.fxn(v, e4, e5

  alpha <- alpha + sum((y-g_of_v)/(x*g_prime_v))
  v <- alpha %*% t(as.matrix(x))
  v <- as.numeric(v)
  iter <- iter+1
```

```
}

# plot data and f(x)
plot(x, y, main="Natural cubic spline", ylim = c(-5,30))                # data
lines(x, lm.fit$fitted.values, col = "blue", lwd = 2)
abline(v=x[c(5,25,50,75,95)], lty = 2, col = "red")
```




```
sum((lm.fit$fitted.values-y)^2)
```

```
## [1] 356.8867
```

Cubic spline implementation

- To estimate $g(v_i)$ I will now use cubic spline.
- I will use 2 knots, thus resulting in 4 basis functions.
- I will initiate $\omega = 0.1$, and update it 5 times

```
############################################
## cubic spline

alpha <- 0.1 %>% as.matrix()
v <- alpha %*% t(as.matrix(x))
v <- as.numeric(v)
iter <- 0

while(iter <= 5){
```

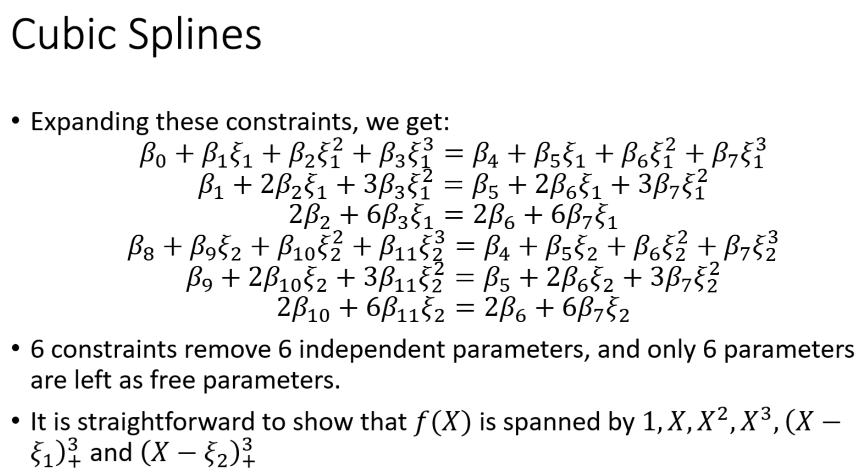
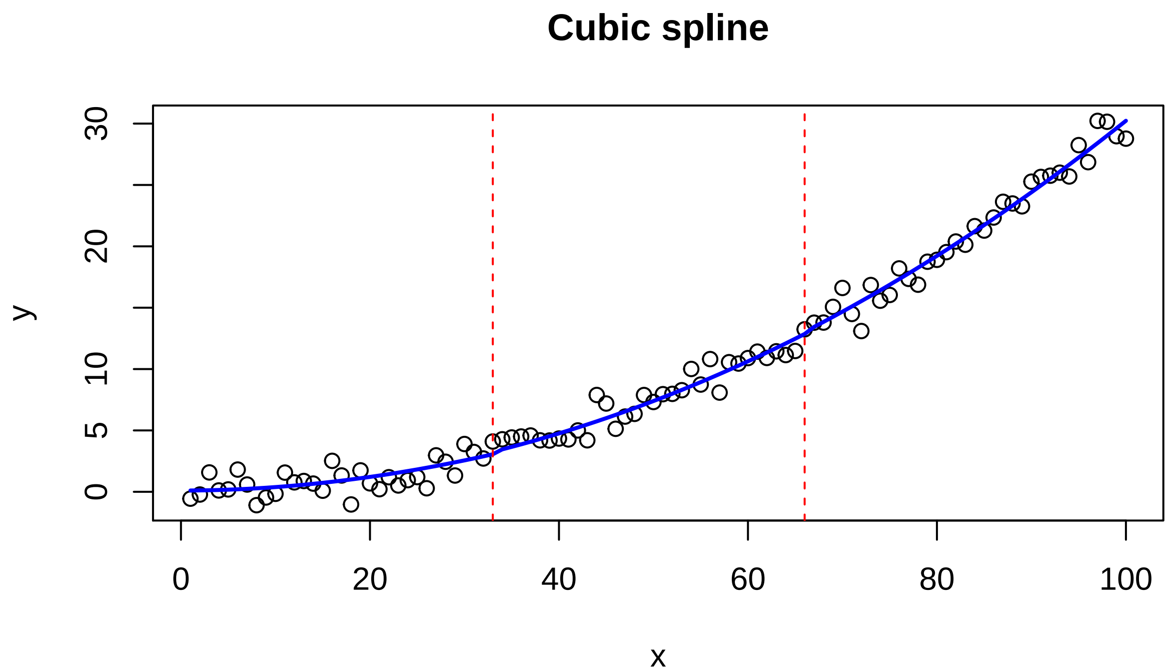


Figure 3: Lecture notes describe the basis function of cubic spline

```
  e1 <- v[33] %>% as.numeric()
  e2 <- v[66] %>% as.numeric()

  lm.fit <- lm(y~.+0, data.frame(y = y, n1 = 1, n2=v, n3=v^2, n4=v^3, n5=basis.fxn(v, e1), n6=basis.fxn

  g_of_v <- lm.fit$fitted.values
  g_prime_v <- lm.fit$coefficients[2]+(2*lm.fit$coefficients[3]*v)+(3*lm.fit$coefficients[4]*v^2)+
    (3*lm.fit$coefficients[5]*basis.fxn.sq(v,e1))+(3*lm.fit$coefficients[6]*basis.fxn.sq(v,e2))

  alpha <- alpha + sum((y-g_of_v)/(x*g_prime_v))
  v <- alpha %*% t(as.matrix(x))
  v <- as.numeric(v)
  iter <- iter+1
}

plot(x, y, main="Cubic spline")
lines(x, lm.fit$fitted.values, col = "blue", lwd = 2)
abline(v=x[c(33,66)], lty = 2, col = "red")
```




```
sum((lm.fit$fitted.values-y)^2)
```

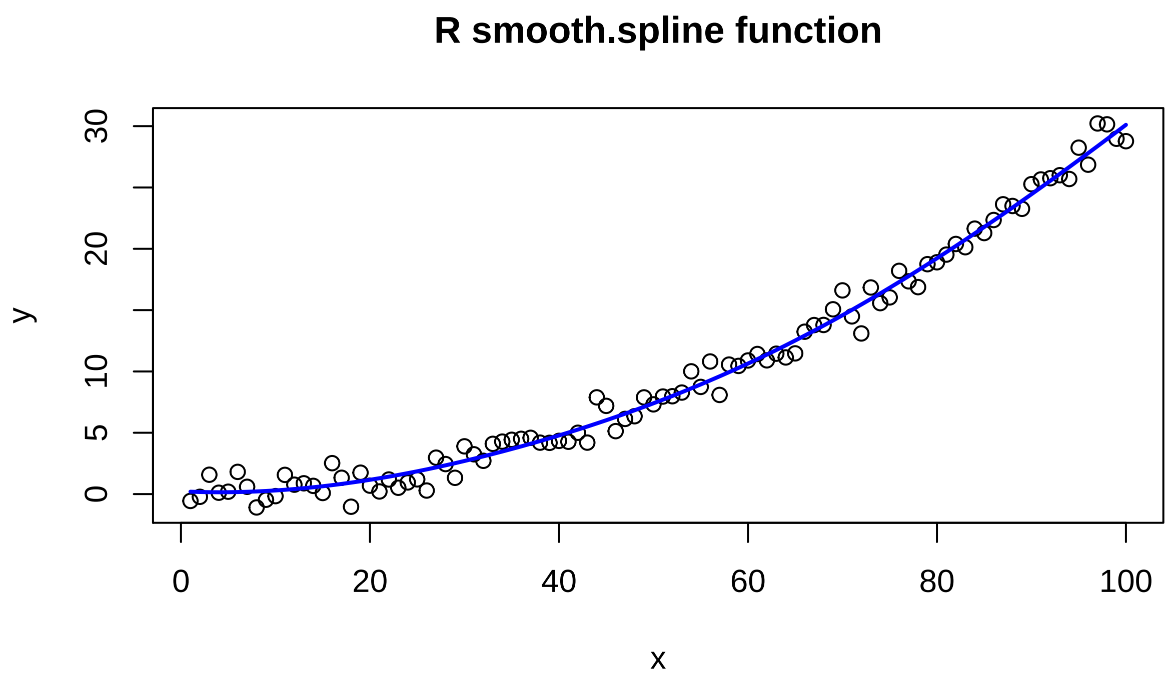
```
## [1] 80.86659
```

smooth.spline R implementation

- To estimate $g(v_i)$ I will now use the build in `smooth.spline` function
- I will initiate $\omega = 0.1$, and update it 5 times

```
alpha <- 0.1 %>% as.matrix()
v <- alpha %*% t(as.matrix(x))
v <- as.numeric(v)
iter <- 0

while(iter <= 5){

  v.smooth.spline <- smooth.spline(v, y, nknots=4)

  g_of_v <- v.smooth.spline$y
  g_prime_v <- predict(v.smooth.spline, v, deriv = 1)$y

  alpha <- alpha + sum((y-g_of_v)/(x*g_prime_v))
  v <- alpha %*% t(as.matrix(x))
  v <- as.numeric(v)
  iter <- iter+1
}
```

```r
plot(x, y, main="R smooth.spline function")
lines(x, v.smooth.spline$y, col = "blue", lwd = 2)
```




```r
sum((v.smooth.spline$y-y)^2)
```

```
## [1] 80.52857
```

Conclusions

- The `ppr` method from R had a sum-of-squared error of 80.56198
- The **natural cubic spline** method from R had a sum-of-squared error of 356.8867
- The **cubic spline** method from R had a sum-of-squared error of 80.86659
- The `smooth.spline` method from R had a sum-of-squared error of 80.52857