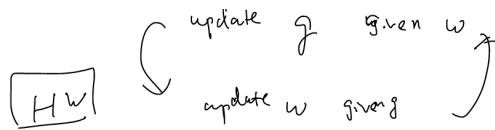


# PHS597 HW1, Spring 2022

Havell Markus

## Description

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



## Fit a PPR Model

- Assume that we have got  $\omega_m$ . We define  $v_i = \omega_m^T x_i$  we only need to find a univariate function  $g(v)$ , such that

$$\sum_{i=1}^N [y_i - g(v_i)]^2$$

- The functional form of  $g$  may be estimated by smoothing splines
- Then, given the function  $g$ , we will update values of  $\omega$ :
- We perform Taylor expansion for the ridge function:

$$g(\omega^T x_i) \approx g(\omega_{old}^T x_i) + g'(\omega_{old}^T x_i)(\omega - \omega_{old})^T x_i$$

- The loss function is reduced to :

$$\sum_{i=1}^N [y_i - g(\omega^T x_i)]^2 \approx \sum_{i=1}^N g'(\omega_{old}^T x_i)^2 \left[ \omega_{old}^T x_i + \frac{y_i - g(\omega_{old}^T x_i)}{g'(\omega_{old}^T x_i)} - \omega^T x_i \right]^2$$

- So the loss function is reduced to a quadratic function of  $\omega$

- The updated  $\omega$  can be obtained by setting the derivatives of  $\sum_{i=1}^N g'(\omega_{old}^T x_i)^2 \left[ \omega_{old}^T x_i + \frac{y_i - g(\omega_{old}^T x_i)}{g'(\omega_{old}^T x_i)} - \omega^T x_i \right]^2$  to 0, and solve for  $\omega$ .

Figure 1: Lecture notes for PPR.

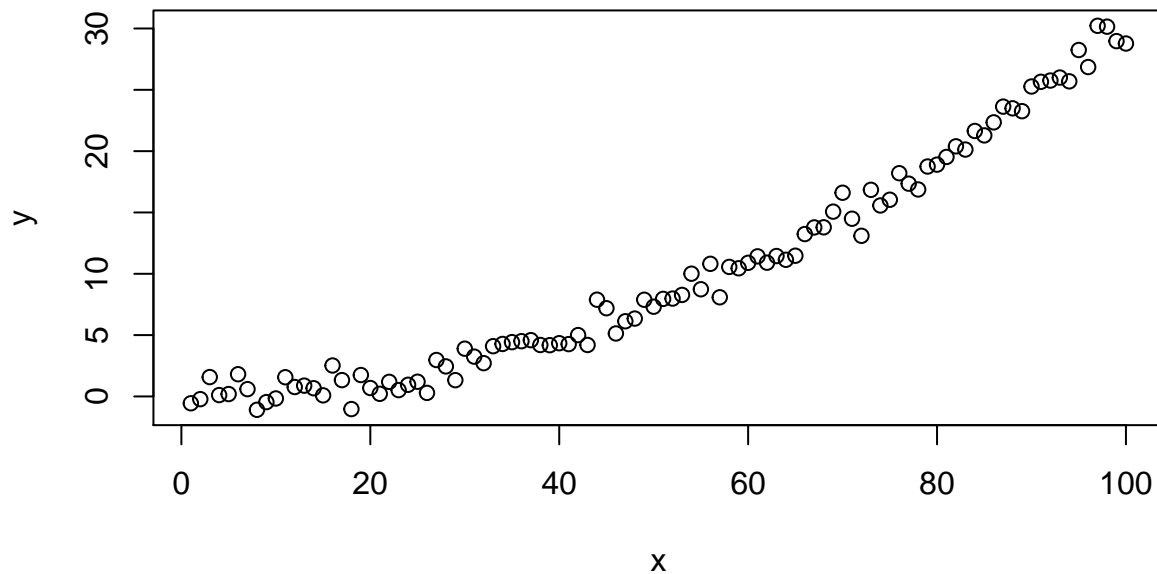
## Simulated data

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

#####

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

plot(x, y)
```

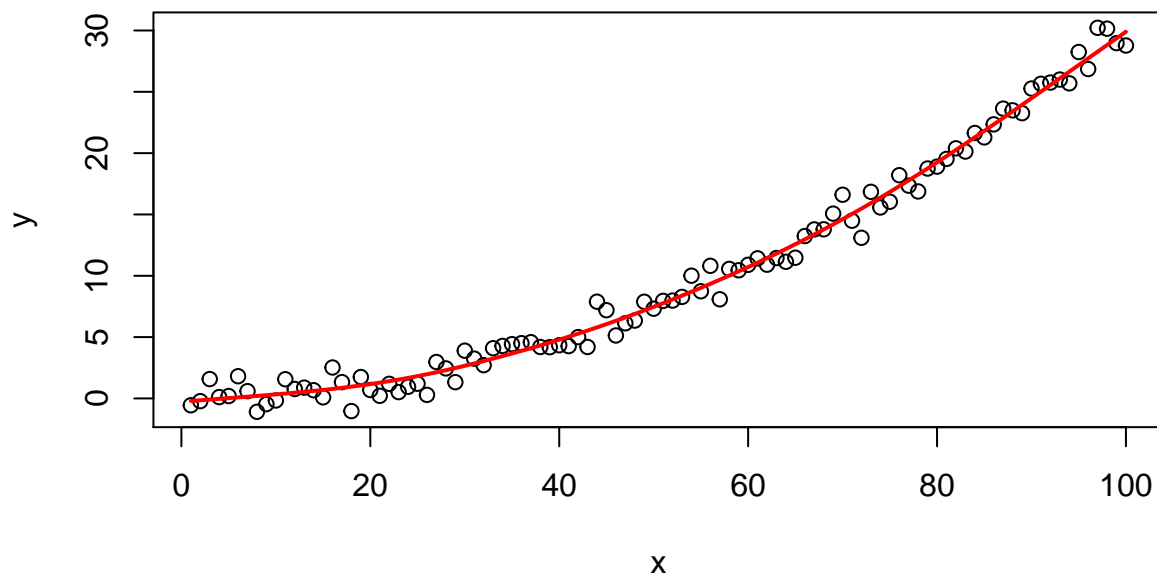


Fitting build-in PPR implementation

```
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")
```



```
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

### Natural Cubic Splines

- Put together, we express  $f(X)$  using the free parameters  $\beta_0, \beta_1$  and  $\theta_1, \dots, \theta_{K-2}$
- $$f(X) = \beta_0 + \beta_1 X + \sum_{k=1}^{K-2} \theta_k [(X - \xi_k)_+^3 - (X - \xi_K)_+^3] - \frac{\theta_k(\xi_k - \xi_K)}{\xi_{K-1} - \xi_K} [(X - \xi_{K-1})_+^3 - (X - \xi_K)_+^3]$$
- So the basis would be:
- $N_1(X) = 1, N_2(X) = X,$
- $$N_{2+k} = \frac{[(X - \xi_k)_+^3 - (X - \xi_K)_+^3]}{\xi_k - \xi_K} - \frac{[(X - \xi_{K-1})_+^3 - (X - \xi_K)_+^3]}{\xi_{K-1} - \xi_K}, k = 1, \dots, K - 2$$

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

```
#####

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.k
})

#####
## 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))

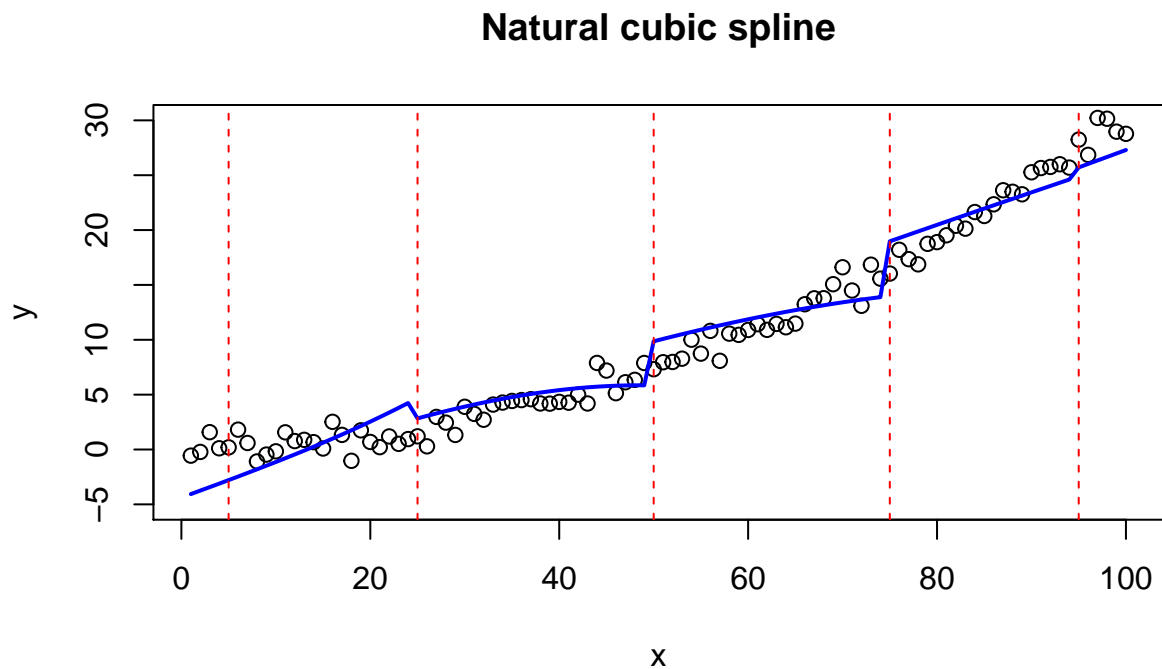
  alpha <- alpha + sum((y-g_of_v)/(x*g_prime_v))
  v <- alpha %*% t(as.matrix(x))
  v <- as.numeric(v)
  iter <- iter+1
}
```

```

}

# 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){

```

## Cubic Splines

- Expanding these constraints, we get:

$$\begin{aligned}\beta_0 + \beta_1 \xi_1 + \beta_2 \xi_1^2 + \beta_3 \xi_1^3 &= \beta_4 + \beta_5 \xi_1 + \beta_6 \xi_1^2 + \beta_7 \xi_1^3 \\ \beta_1 + 2\beta_2 \xi_1 + 3\beta_3 \xi_1^2 &= \beta_5 + 2\beta_6 \xi_1 + 3\beta_7 \xi_1^2 \\ 2\beta_2 + 6\beta_3 \xi_1 &= 2\beta_6 + 6\beta_7 \xi_1 \\ \beta_8 + \beta_9 \xi_2 + \beta_{10} \xi_2^2 + \beta_{11} \xi_2^3 &= \beta_4 + \beta_5 \xi_2 + \beta_6 \xi_2^2 + \beta_7 \xi_2^3 \\ \beta_9 + 2\beta_{10} \xi_2 + 3\beta_{11} \xi_2^2 &= \beta_5 + 2\beta_6 \xi_2 + 3\beta_7 \xi_2^2 \\ 2\beta_{10} + 6\beta_{11} \xi_2 &= 2\beta_6 + 6\beta_7 \xi_2\end{aligned}$$

- 6 constraints remove 6 independent parameters, and only 6 parameters are left as free parameters.
- It is straightforward to show that  $f(X)$  is spanned by  $1, X, X^2, X^3, (X - \xi_1)_+^3$  and  $(X - \xi_2)_+^3$

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

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

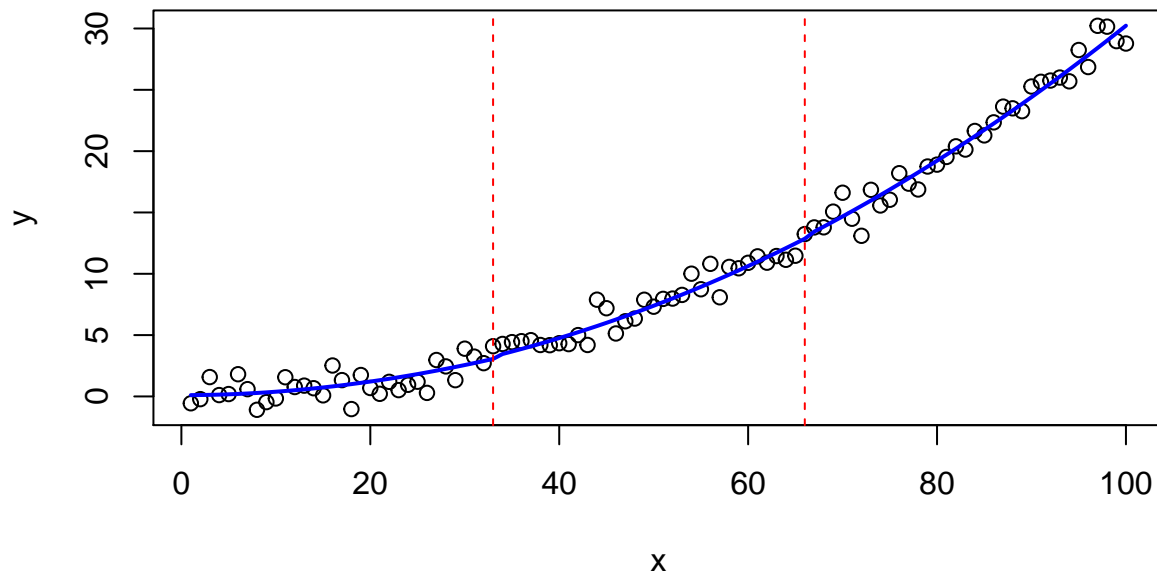
lm.fit <- lm(y~.+0, data.frame(y = y, n1 = 1, n2=v, n3=v^2, n4=v^3, n5=basis.fxn(v, e1), n6=basis.fxn(v, e2)))

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")
```

## Cubic spline



```
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){

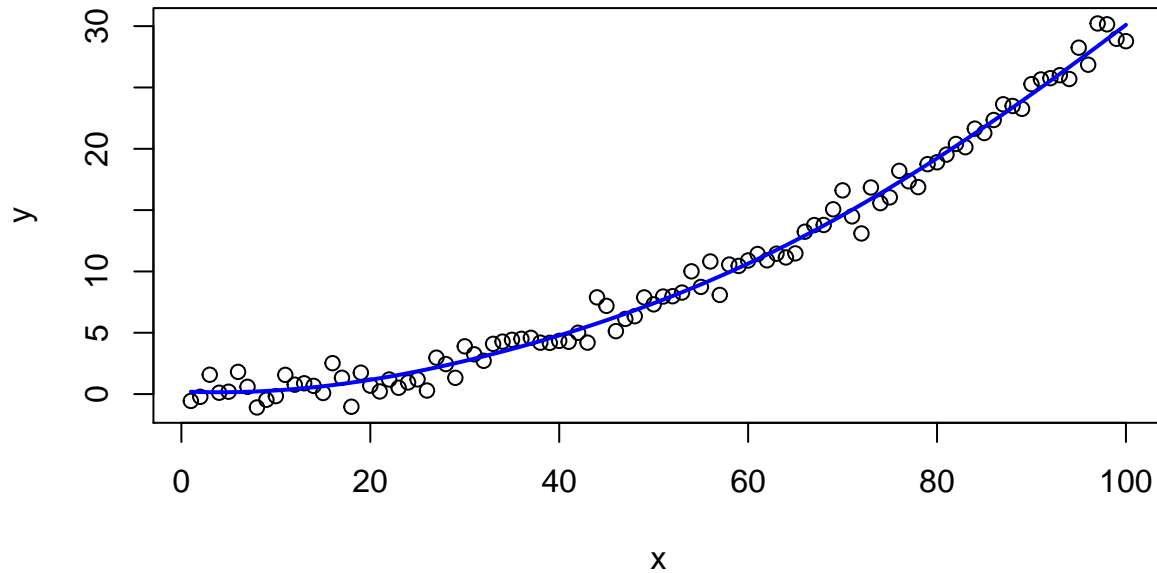
  v.smooth.spline <- smooth.spline(v, y, nknots=4)

  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
}
```

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

## R smooth.spline function



```
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