STATS 790 - Statistical Learning
Homework 3

Tommy Flynn

03 March, 2023

Contents

Question 1:

Below is the original ESL Figure 5.1 and my version followed by the R code used to create it.

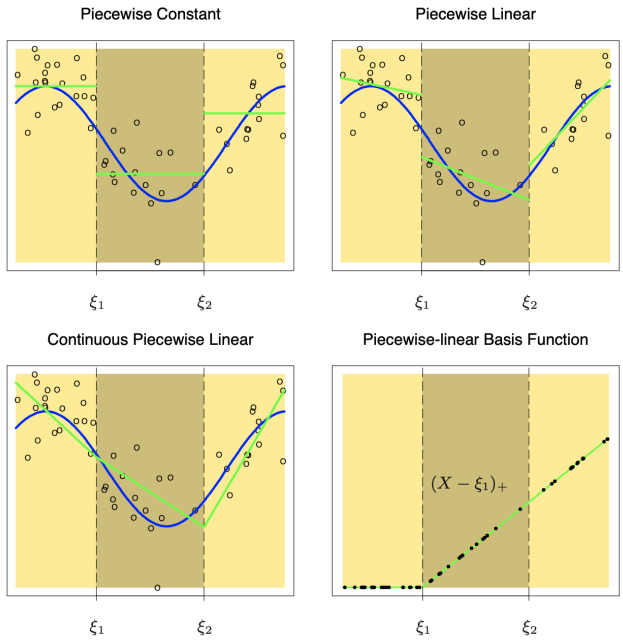

Figure 1: Original Figure 5.1 from ESL.

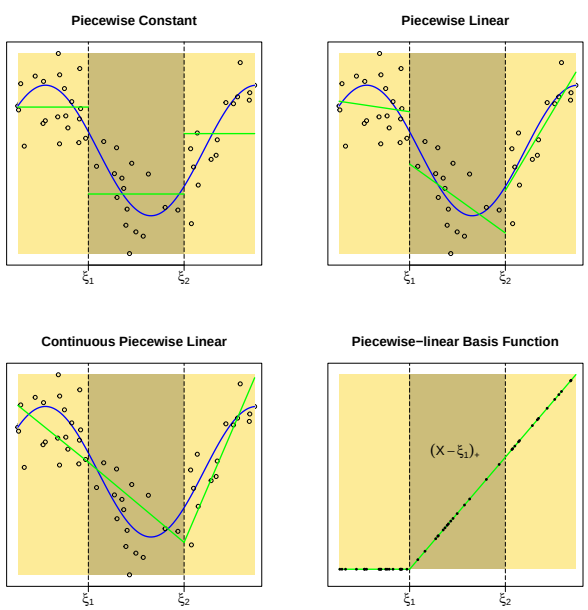

Figure 2: Replica of Figure 5.1.

```r
# seed
set.seed(13)

# data for true curve - let's say sin(3x)/2
x.true <- seq(0.25, 2.6, length.out=100)
y.true <- sin(3*x.true)/2

# knots and noise
xi.1 <- 0.95
xi.2 <- 1.9
s <- 0.2

# generate as many points as the book did for each section - slightly pedantic
n1 <- 20
n2 <- 16
n3 <- 14

x1 <- runif(n1, min=0.25, max=xi.1)
y1 <- sin(3*x1)/2 + rnorm(n1, mean=0, sd=s)

x2 <- runif(n2, min=xi.1, max=xi.2)
y2 <- sin(3*x2)/2 + rnorm(n2, mean=0, sd=s)

x3 <- runif(n3, min=xi.2, max=2.6)
y3 <- sin(3*x3)/2 + rnorm(n3, mean=0, sd=s)

ys <- c(y1, y2, y3)
xs <- c(x1, x2, x3)


# plots the sample points and the true curve for the first three subplots
plot.base <- function(t) {
```

```r
  # figure set-up
  plot(x.true, y.true, xaxt='n', xlab='', ylab='', yaxt='n', main=t,
       cex.main=1.5, ylim=c(min(ys) - 0.025, max(ys) + 0.025),
       xlim=c(min(xs) - 0.025, max(xs) + 0.025))
  rect(0.25, min(ys), 2.6, max(ys), col='#FDEB98', border=NA)
  rect(xi.1, min(ys), xi.2, max(ys), col='#CCBD7A', border=NA)

  # true curve and sample points
  lines(x.true, y.true, col='blue', lwd=2)
  points(xs, ys, lwd=1.25)

  # dashed lines at knots and axis
  abline(v=xi.1, col='black', lty=5)
  abline(v=xi.2, col='black', lty=5)
  axis(1, at=c(xi.1, xi.2), labels=c(expression(xi[1]), expression(xi[2])),
       cex.axis=1.5)
}

# basis functions according to ESL
piecewise.constant <- function(x) {
  cbind(sapply(x, function(x) ifelse(x < xi.1, 1, 0)),
        sapply(x, function(x) ifelse(x >= xi.1 & x < xi.2 , 1, 0)),
        sapply(x, function(x) ifelse(x >= xi.2, 1, 0)))
}

piecewise.linear <- function(x) {
  cbind(sapply(x, function(x) ifelse(x < xi.1, 1, 0)),
        sapply(x, function(x) ifelse(x < xi.1, 1, 0)*x),
        sapply(x, function(x) ifelse(x >= xi.1 & x < xi.2 , 1, 0)),
        sapply(x, function(x) ifelse(x >= xi.1 & x < xi.2 , 1, 0)*x),
        sapply(x, function(x) ifelse(x >= xi.2, 1, 0)),
        sapply(x, function(x) ifelse(x >= xi.2, 1, 0)*x))
```

```r
}

continuous.piecewise.linear <- function(x) {
  cbind(sapply(x, function(x) 1),
        sapply(x, function(x) x),
        sapply(x, function(x) pmax(x-xi.1, 0)),
        sapply(x, function(x) pmax(x-xi.2, 0)))
}

spline.model <- function(B) {
  model <- lm(y~.-1, data=data.frame(x=B, y=ys))
  return(model$coefficients)
}


# 2 by 2 figure
par(mfrow=c(2,2))

# plot (1, 1)
plot.base('Piecewise Constant')

# model
beta.pc <- spline.model(piecewise.constant(xs))

# lines
segments(x0=0.25, x1=xi.1, y0=beta.pc[1], y1=beta.pc[1], col='green', lwd=2)
segments(x0=xi.1, x1=xi.2, y0=beta.pc[2], y1=beta.pc[2], col='green', lwd=2)
segments(x0=xi.2, x1=2.6, y0=beta.pc[3], y1=beta.pc[3], col='green', lwd=2)


# plot (1, 2)
```

```r
plot.base('Piecewise Linear')

# model
beta.pl <- spline.model(piecewise.linear(xs))

# lines
segments(x0=0.25, x1=xi.1,
         y0=beta.pl[1] + 0.25*beta.pl[2],
         y1=beta.pl[1] + xi.1*beta.pl[2],
         col='green', lwd=2)

segments(x0=xi.1, x1=xi.2,
         y0=beta.pl[3] + xi.1*beta.pl[4],
         y1=beta.pl[3] + xi.2*beta.pl[4],
         col='green', lwd=2)

segments(x0=xi.2, x1=2.6,
         y0=beta.pl[5] + xi.2*beta.pl[6],
         y1=beta.pl[5] + 2.6*beta.pl[6],
         col='green', lwd=2)


# plot (2, 1)
plot.base('Continuous Piecewise Linear')

# model
beta.cpl <- spline.model(continuous.piecewise.linear(xs))

# lines
f1 <- beta.cpl[1] + beta.cpl[2]*0.25
f2 <- beta.cpl[1] + beta.cpl[2]*xi.1
```

```r
f3 <- beta.cpl[1] + beta.cpl[2]*xi.2 + beta.cpl[3]*(xi.2 - xi.1)
f4 <- beta.cpl[1] + beta.cpl[2]*2.6 + beta.cpl[3]*(2.6 - xi.1) +
  beta.cpl[4]*(2.6 - xi.2)

segments(x0=0.25, x1=xi.1, y0=f1, y1=f2, col='green', lwd=2)
segments(x0=xi.1, x1=xi.2, y0=f2, y1=f3, col='green', lwd=2)
segments(x0=xi.2, x1=2.6, y0=f3, y1=f4, col='green', lwd=2)



# plot (2, 2)

# basis function
ys.b <- pmax(xs - xi.1, 0)

# plot base
plot(xs, ys.b, xaxt='n', xlab='', ylab='', yaxt='n',
     main='Piecewise-linear Basis Function', col='white', cex.main=1.5,
     ylim=c(min(ys.b) - 0.075, max(ys.b) + 0.075),
     xlim=c(min(xs) - 0.025, max(xs) + 0.025))
rect(0.25, 0, 2.6,  1.65, col='#FDEB98', border=NA)
rect(xi.1, 0, xi.2,  1.65, col='#CCBD7A', border=NA)
abline(v=xi.1, col='black', lty=5)
abline(v=xi.2, col='black', lty=5)
text(1.4, 1, labels=expression((X- xi[1])['+']), cex=1.5)
axis(1, at=c(xi.1, xi.2), labels=c(expression(xi[1]), expression(xi[2])),
     cex.axis=1.5)

# lines and points
segments(x0=0.25, x1=xi.1, y0=0, y1=0, col='green', lwd=2)
segments(x0=xi.1, x1=2.6, y0=0, y1=pmax(2.6 - xi.1, 0), col='green', lwd=2)
points(xs, ys.b, cex=0.7, pch=16)
```

Question 2:

Below is the predicted values and SE bands for tobacco from the South Africa coronary heart disease data using B-spline, natural spline, and truncated polynomial spline bases in a logistic regression model with an intercept. The figure is followed by the code used to produce the plot. Note: it was assumed that 5 knots meant 3 interior knots and 2 boundary knots - hopefully this is adequate. I also tried to mimic the style presented in ESL Figure 5.4.

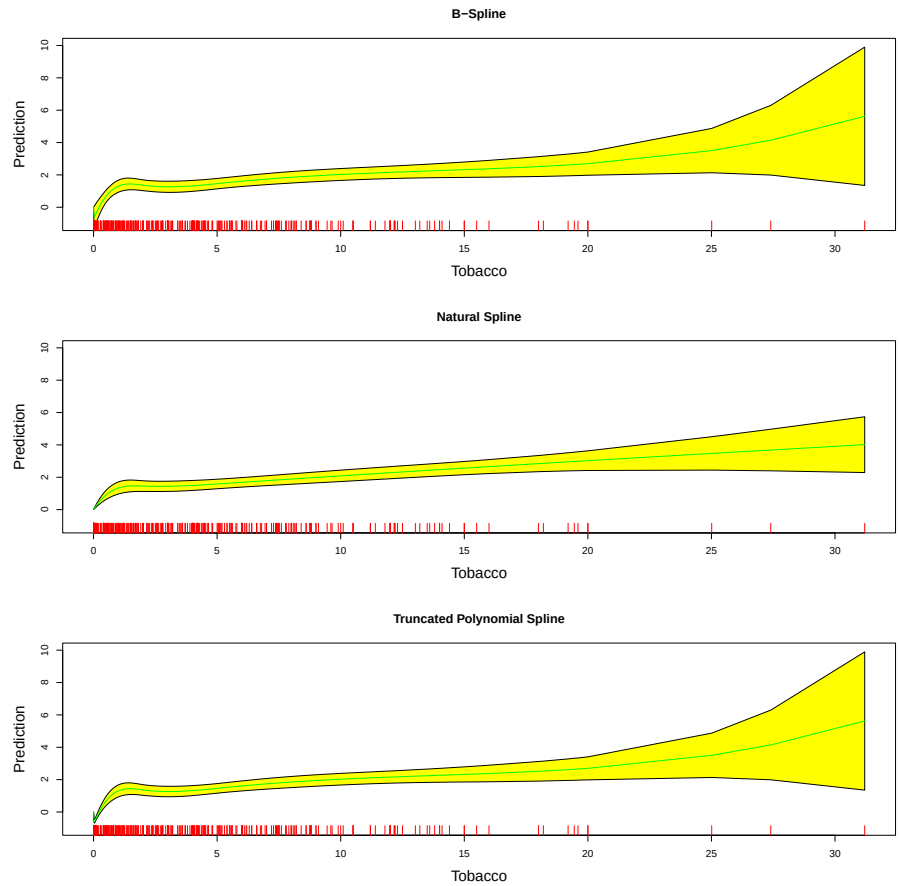

Figure 3: B-spline, natural spline, and truncated polynomial spline predicted values with ±1 SE band.

```
# set-up
library(splines)
set.seed(13)
heart <- read.table('SAheart.data.txt', sep=',', head=TRUE, row.names=1)
```

```r
# sort data so it plots line left to right
ind <-  sort(heart$tobacco, index.return=TRUE)$ix
heart$tobacco <- heart$tobacco[ind]
heart$chd <- heart$chd[ind]
x <- heart$tobacco

# bases
b.basis <- bs(x, df=6)
nat.basis <- ns(x, df=4)
trunc.basis.f <- function(x) {
  cbind(sapply(x, function(x) x),
        sapply(x, function(x) x^2),
        sapply(x, function(x) x^3),
        sapply(x, function(x) pmax((x - 0.0525)^3, 0)),
        sapply(x, function(x) pmax((x - 2)^3, 0)),
        sapply(x, function(x) pmax((x - 5.5)^3, 0)))
}
trunc.basis <- trunc.basis.f(x)

# models
b.model <- glm(chd ~ b.basis, data=heart, family=binomial)
nat.model <- glm(chd ~ nat.basis, data=heart, family=binomial)
trunc.model <- glm(chd ~ trunc.basis, data=heart, family=binomial)

# fitted values
b.fit <- b.basis %*% coef(b.model)[2:7]
nat.fit <- nat.basis %*% coef(nat.model)[2:5]
trunc.fit <- trunc.basis %*% coef(trunc.model)[2:7]

# variances
b.H <- cbind(rep(1, 462), b.basis)
b.W <- diag(b.model$weight)
```

```r
b.var <- solve(t(b.H) %*% b.W %*% b.H)[2:7, 2:7]
b.band <- sqrt(diag(b.basis %*% b.var %*% t(b.basis)))

nat.H <- cbind(rep(1, 462), nat.basis)
nat.W <- diag(nat.model$weight)
nat.var <- solve(t(nat.H) %*% nat.W %*% nat.H)[2:5, 2:5]
nat.band <- sqrt(diag(nat.basis %*% nat.var %*% t(nat.basis)))

trunc.H <- cbind(rep(1, 462), trunc.basis)
trunc.W <- diag(trunc.model$weight)
trunc.var <- solve((t(trunc.H) %*% trunc.W %*% trunc.H)+1e-6*diag(7))[2:7, 2:7]
trunc.band <- sqrt(diag(trunc.basis %*% trunc.var %*% t(trunc.basis)))

# plots
par(mfrow=c(3,1))

# B-Spline
plot(x, b.fit, col='green', type='l', main='B-Spline', xlab='Tobacco',
     ylab='Prediction', ylim=c(-1, 10), cex.lab=1.5)
polygon(c(x, rev(x)), c(b.fit - b.band, rev(b.fit + b.band)), col='yellow')
lines(x, b.fit, col='green', type='l')
axis(side=1, tck=0.05, at=x, labels=FALSE, tick=TRUE, col.ticks='red')

# Natural Spline
plot(x, nat.fit, col='green', type='l', main='Natural Spline', xlab='Tobacco',
     ylab='Prediction', ylim=c(-1, 10), cex.lab=1.5)
polygon(c(x, rev(x)), c(nat.fit-nat.band, rev(nat.fit+nat.band)), col='yellow')
lines(x, nat.fit, col='green', type='l')
axis(side=1, tck=0.05, at=x, labels=FALSE, tick=TRUE, col.ticks='red')

# Truncated Polynomial Spline
```

```r
plot(x, trunc.fit, col='green', type='l', main='Truncated Polynomial Spline',
     xlab='Tobacco', ylab='Prediction', ylim=c(-1, 10), cex.lab=1.5)
polygon(c(x, rev(x)), c(trunc.fit - trunc.band, rev(trunc.fit + trunc.band)),
        col='yellow')
lines(x, trunc.fit, col='green', type='l')
axis(side=1, tck=0.05, at=x, labels=FALSE, tick=TRUE, col.ticks='red')
```

Question 3:

Below is the side-by-side plot of the truncated polynomial basis and the natural spline basis using the matplot function in R. Following this is the function used to create the bases which takes data, degrees of freedom, and a bool to determine whether to use natural or not.

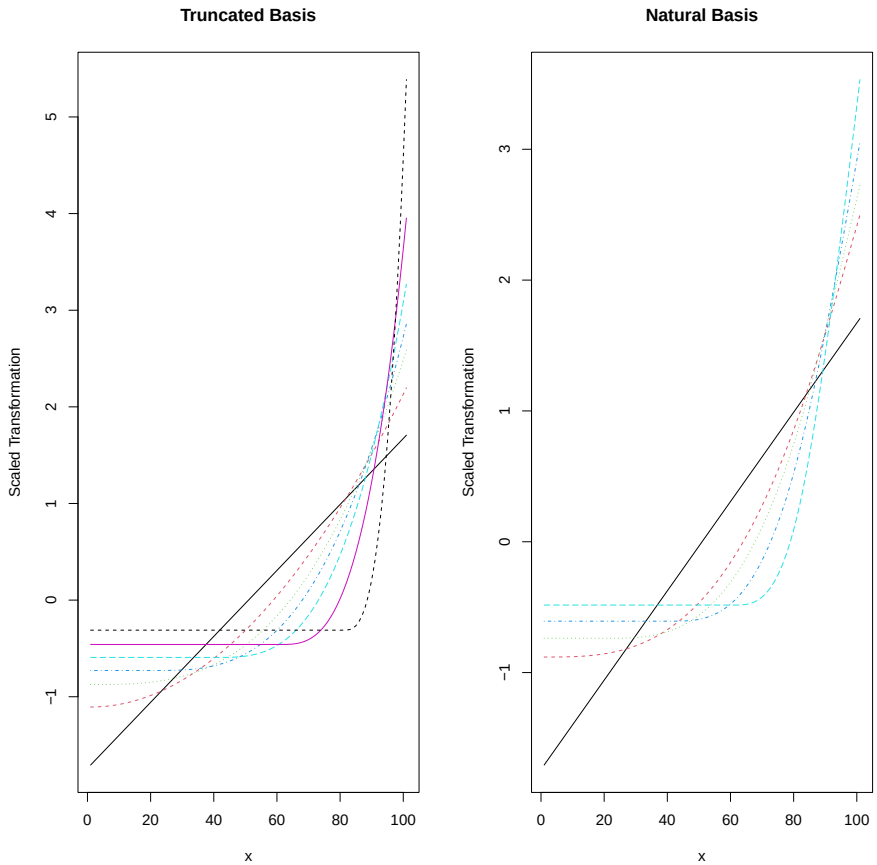

Figure 4: Truncated polynomial spline and natural spline bases comparison.

```
# set-up
library(splines)
library(Matrix)
set.seed(13)

# main function
truncpolyspline <- function(x, df, natural=FALSE) {
```

```r
  # truncated case
  if (natural == FALSE) {                              # altered from the notes
    knots <- quantile(x, seq(0, 1, length = df - 1))
    trunc_fun <- function(k) {(x>=k)*(x-k)^3}
    S <- sapply(knots[1:(df-2)], trunc_fun)
    S <- as(S, "CsparseMatrix")
    S <- cbind(x, x^2, S)
  }

  # natural case
  else{
    knots <- quantile(x, seq(0, 1, length = df - 1))
    K <- knots[df-1]
    K1 <- knots[df-2]
    d.K1 <- ((x>=K1)*(x-K1)^3 - (x>=K)*(x-K)^3)/(K-K1) # ESL Equation 5.5
    trunc_fun <- function(k) {
      ((x>=k)*(x-k)^3 - (x>=K)*(x-K)^3)/(K-k) - d.K1   # ESL Equation 5.4
      }
    S <- sapply(knots[1:(df-3)], trunc_fun)
    S <- as(S, "CsparseMatrix")
    S <- cbind(x, S)
  }

  return(S)
}

# sample data
xvec <- seq(0, 1, length = 101)
trunc <- truncpolyspline(sort(xvec), df=7, natural=FALSE)
nat <- truncpolyspline(sort(xvec), df=7, natural=TRUE)

# plot
```

```r
par(mfrow=c(1,2))
matplot(scale(trunc), type='l', main='Truncated Basis',
        xlab='x', ylab='Scaled Transformation')
matplot(scale(nat), type='l',  main='Natural Basis',
        xlab='x', ylab='Scaled Transformation')
```

Question 4:

a) I chose to simulate data from the smooth 2D surface of a spherical cap. This surface can be parameterized as follows:

$$\sigma(\theta, \phi) = (\cos(\theta)\sin(\phi),\ \sin(\theta)\sin(\phi),\ \cos(\phi)),\ \ \theta \in [0, 2\pi], \phi \in [0, \pi/3]$$

Since the radius is 1 it lies on the unit sphere. It is simply the top of the unit sphere down to an angle of $\pi/3$. The function is displayed in the r code below.

b) Using the simulated data from the surface, we fit 2D splines for 250 iterations via gam with generalized cross-validation and restricted maximum likelihood. At each iteration we note the bias, variance, mse of predictions as well as the time via microbenchmark. The code is displayed below along with a table of average results.

	Bias	**Variance**	**MSE**	**Time (ms)**
GCV	$-3.89 \times 10^{-16} \approx 0$	0.00226	0.0127	22.43208
REML	$3.61 \times 10^{-16} \approx 0$	0.00165	0.0145	29.58928

```r
# set-up
library(mgcv)
library(microbenchmark)
set.seed(13)

# generates data from a spherical cap on the unit sphere - from top to pi/3
surface.data <- function(n) {
  # 2D parameters
  theta <- runif(n, 0, 2*pi)
  phi <- runif(n, 0, pi/3)

  # noisy data
  s <- 0.1
  x <- cos(theta) * sin(phi) + rnorm(n, 0, s)
  y <- sin(theta) * sin(phi) + rnorm(n, 0, s)
  z <- cos(phi) + rnorm(n, 0, s)
```

```r
  return(list(x=x, y=y, z=z))
}

info <- function(D, method) {
  # model
  m <- gam(z ~ te(x, y, bs = 'gp'), method=method, data=D)

  # details
  pred <- predict(m, type='response', se.fit=TRUE)
  bias <- mean(pred$fit - D$z)
  variance <- mean(pred$se.fit^2)
  mse <- mean((D$z - pred$fit)^2)
  return(c(bias, variance, mse))
}

D <- surface.data(100)
GCV.Cp <- c(0, 0, 0)
REML <- c(0, 0, 0)

# computational complexity, bias, variance, and mse
m <- microbenchmark(GCV.Cp <- GCV.Cp + info(D, method='GCV.Cp'),
                    REML <-REML + info(D, method='REML'),
                    times=250,
                    unit='ms')

# results
m
GCV.Cp/250
REML/250
```

Question 5: (ESL 5.4)

Let us begin with the truncated power basis with K knots and define

$$f(X) = \sum_{j=0}^{3} \beta_j X^j + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3$$

Natural cubic splines constrain the function to be linear before and after the boundary knots ξ_1 and ξ_K respectively. Before ξ_1 we have:

$$f(X) = \sum_{j=0}^{3} \beta_j X^j + \cancelto{0}{\sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3} = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$$

For linearity to hold we require $\beta_2 = \beta_3 = 0$ so that $f(X) = \beta_0 + \beta_1 X$. Continuing from the previous constraint, we have the following after ξ_K:

$$f(X) = \beta_0 + \beta_1 X + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3 = \beta_0 + \beta_1 X + \sum_{k=1}^{K} (\theta_k X^3 - 3\xi_k \theta_k X^2 + 3\xi_k^2 \theta_k X - \xi_k^3 \theta_k)$$

For linearity to hold we require $\sum_{k=1}^{K} \theta_k = \sum_{k=1}^{K} \xi_k \theta_k = 0$ so that $f(X) = \beta_0 + \beta_1 X + \sum_{k=1}^{K} (3\xi_k^2 \theta_k X - \xi_k^3 \theta_k)$. With these constraints on the truncated power basis we can derive the natural cubic spline basis as follows:

$f(X) = \beta_0 + \beta_1 X + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3$

evaluate last two terms of series

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2} \theta_k (X - \xi_k)_+^3 + \theta_K (X - \xi_K)_+^3 + \theta_{K-1} (X - \xi_{K-1})_+^3$

rewrite θ_K using constraint $\sum_{k=1}^{K} \theta_k = 0 \implies \theta_K = -\sum_{k=1}^{K-2} \theta_k - \theta_{K-1}$

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2} \theta_k (X - \xi_k)_+^3 + [-\sum_{k=1}^{K-2} \theta_k (X - \xi_K)_+^3 - \theta_{K-1} (X - \xi_K)_+^3] + \theta_{K-1} (X - \xi_{K-1})$

factor

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2} \theta_k [(X - \xi_k)_+^3 - (X - \xi_K)_+^3] + \theta_{K-1} [(X - \xi_{K-1})_+^3 - (X - \xi_K)_+^3]$

rewrite in terms of d

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2} \theta_k (\xi_K - \xi_k) d_k(X) + \theta_{K-1} (\xi_K - \xi_{K-1}) d_{K-1}(X)$

rewrite $-\theta_{K-1}\xi_{K-1}$ using constraint $\sum_{k=1}^{K}\theta_k\xi_k = 0 \implies -\theta_{K-1}\xi_{K-1} = \sum_{k=1}^{K-2}\theta_k\xi_k + \theta_K\xi_K$

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2}\theta_k(\xi_K - \xi_k)d_k(X) + [\theta_{K-1}\xi_K + (\sum_{k=1}^{K-2}\theta_k\xi_i + \theta_K\xi_K)]d_{K-1}(X)$

factor

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2}\theta_k(\xi_K - \xi_k)d_k(X) + [(\theta_{K-1} + \theta_K)\xi_K + \sum_{k=1}^{K-2}\theta_k\xi_k]d_{K-1}(X)$

rewrite $\theta_{K-1} + \theta_K$ using constraint $\sum_{k=1}^{K}\theta_k = 0 \implies \theta_{K-1} + \theta_K = -\sum_{k=1}^{K-2}\theta_k$

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2}\theta_k(\xi_K - \xi_k)d_k(X) + [-\sum_{k=1}^{K-2}\theta_k\xi_K + \sum_{k=1}^{K-2}\theta_k\xi_k]d_{K-1}(X)$

factor

$= \beta_0 + \beta_1 X + \sum_{k=1}^{K-2}\theta_k(\xi_K - \xi_k)(d_k(X) - d_{K-1}(X))$

$= \beta_0 N_1(X) + \beta_1 N_2(X) + \sum_{k=1}^{K-2}\theta_k(\xi_K - \xi_k)(N_{k+2}(X))$

Therefore, the natural spline basis can be written using a truncated power spline basis under natural boundary constraints by choosing coefficients β_0 for $N_1(X)$, β_1 for $N_2(X)$, and $\theta_k(\xi_K - \xi_k)$ for $N_{k+2}(X)$, $k = 1, \ldots, K-2$.

Question 6: (ESL 5.13)

Suppose we have fitted a smoothing spline to (x_i, y_i), $i = 1, \ldots, N$ obtaining $\hat{f}_\lambda$ and augment the data with $(x_0, \hat{f}_\lambda(x_0))$. Notice that $\hat{f}_\lambda(x_0) = \hat{f}_\lambda^{(-0)}(x_0)$ since x_0 was not in the original data. This means we can write the augmented data vector as $\vec{Y}^* = \vec{Y} + (\hat{f}_\lambda^{(-0)}(x_0) - y_0, 0, \ldots, 0)'$. Proceeding we have:

$\hat{f}_{\text{Aug}} = \hat{f}_\lambda^{(-0)} = \mathbf{S}_\lambda \vec{Y}^*$

$= \mathbf{S}_\lambda(\vec{Y} + (\hat{f}_\lambda^{(-0)}(x_0) - y_0, 0, \ldots, 0)'$

$= \mathbf{S}_\lambda \vec{Y} + \mathbf{S}_\lambda(\hat{f}_\lambda^{(-0)}(x_0) - y_0, 0, \ldots, 0)'$

$= \hat{f}_\lambda + \vec{s}_1(\hat{f}_\lambda^{(-0)}(x_0) - y_0)$

$\iff \hat{f}_\lambda^{(-0)}(x_0) = \hat{f}_\lambda(x_0) + s_{11}(\hat{f}_\lambda^{(-0)}(x_0) - y_0)$

$\iff \hat{f}_\lambda^{(-0)}(x_0) - y_0 = \hat{f}_\lambda(x_0) + s_{11}(\hat{f}_\lambda^{(-0)}(x_0) - y_0) - y_0$

$\iff (\hat{f}_\lambda^{(-0)}(x_0) - y_0)(1 - s_{11}) = \hat{f}_\lambda(x_0) - y_0$

$\iff \hat{f}_\lambda^{(-0)}(x_0) - y_0 = \frac{\hat{f}_\lambda(x_0) - y_0}{1 - s_{11}}$

Since x_0 is general we can apply this to the cross validation formula as follows:

$$\text{CV}(\hat{f}_\lambda) = \sum_{i=1}^{N} (y_i - \hat{f}_\lambda^{(-i)}(x_i))^2 = \sum_{i=1}^{N} \left(\frac{\hat{f}_\lambda(x_i) - y_i}{1 - s_{ii}} \right)^2$$