STATS 790 Assignment 3

Yiran Zhang
400119421

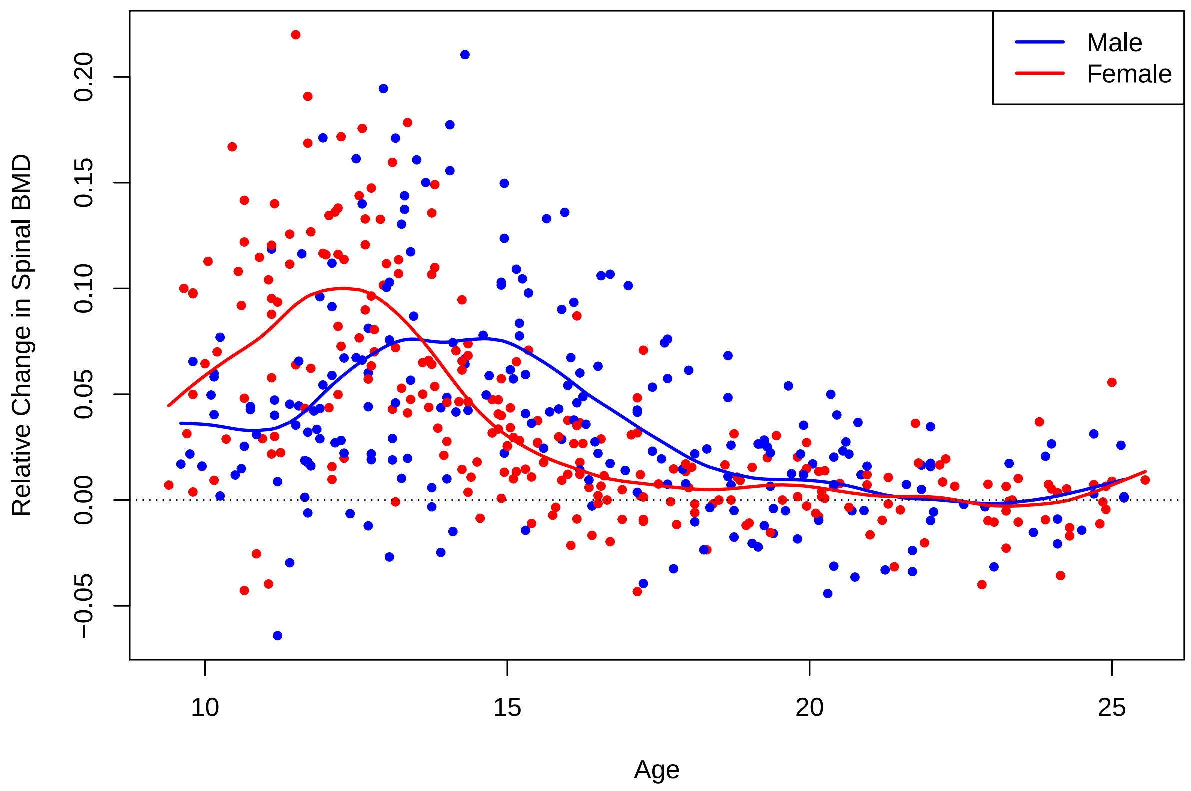
01 March, 2023

Question 1

ESL Chapter 5 Figure 5.6 is replicated, the code is shown as below.

```r
# Question 1
# Replicate ESL Chapter 5 Figure 5.6

# Import dataset
bone <- read.table('https://hastie.su.domains/ElemStatLearn/datasets/bone.data',
                   header = TRUE)

# According to ESL Figure 5.6 description, spnbmd is the target variable
# and age is the predictor.

# Plot the age against relative change in spinal BMD, color separated by gender.
plot(x = bone$age, y = bone$spnbmd,
     col = ifelse(bone$gender == 'female','red','blue'),
     pch = 20, xlab = 'Age', ylab = 'Relative Change in Spinal BMD')

# Split male and female.
male_bone <- bone[bone$gender == 'male', ]
female_bone <- bone[bone$gender == 'female', ]

# Spline, using degree of freedom = 12 (given by the textbook)
male_spline <- smooth.spline(x = male_bone$age, y = male_bone$spnbmd, df = 12)
female_spline <- smooth.spline(x = female_bone$age, y = female_bone$spnbmd, df = 12)
```

```r
# Add splines to the graph with corresponding color for each gender.
lines(male_spline, col = 'blue', lwd = 2)
lines(female_spline, col = 'red', lwd = 2)

# Add a horizontal dash line at y = 0.
abline(h=0, lty=3)

# Add a legend at the top right corner.
legend(x='topright', legend=c('Male', 'Female'),
       col=c('blue', 'red'), lwd=2)
```

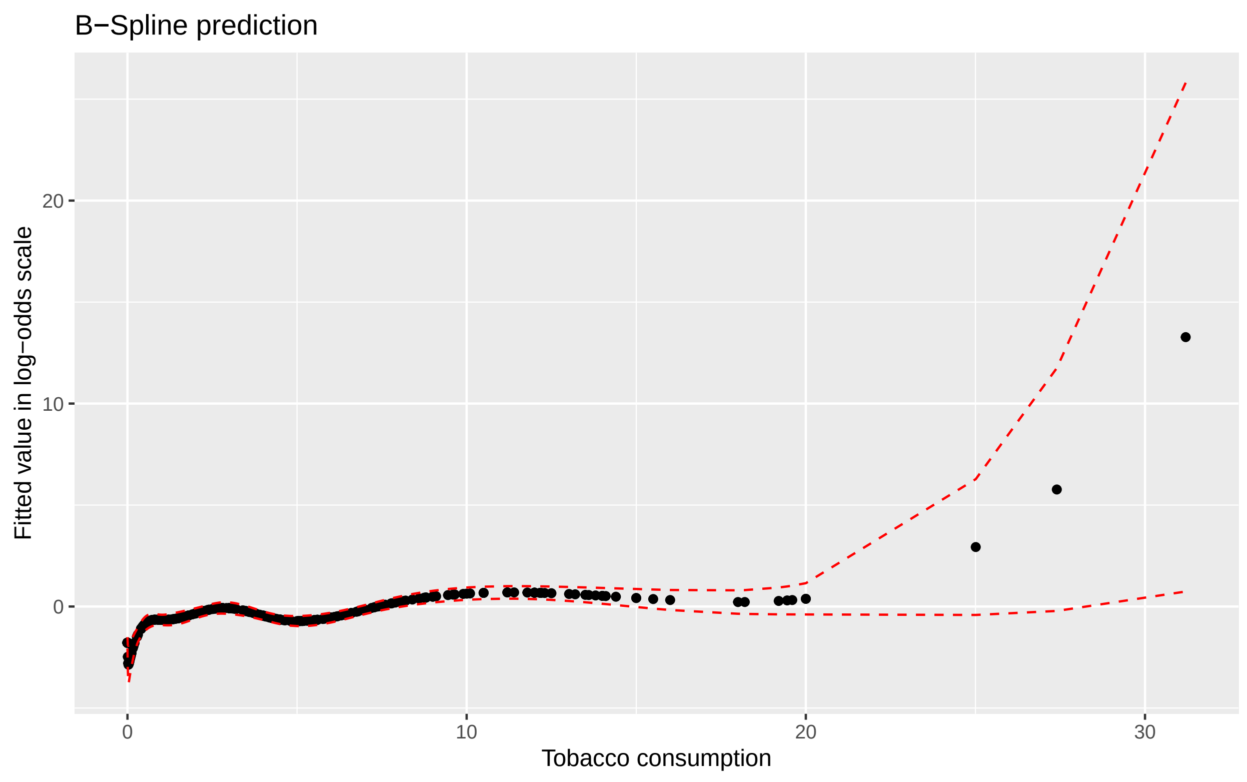


Question 2 (South Africa coronary heart disease data)

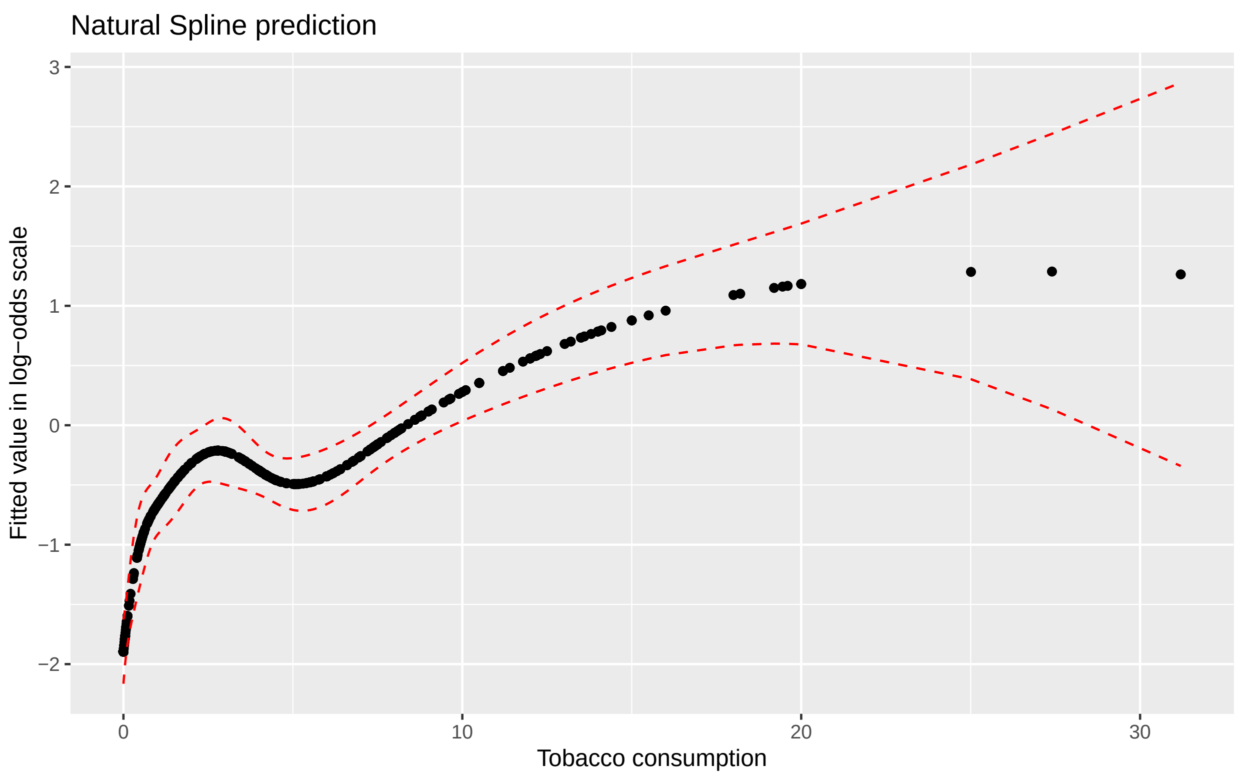
```r
# Question 2
# Import libraries
```

```r
library(splines)
library(ggplot2)

# Import South Africa coronary heart disease data.
url <- "http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data"
heart <- read.csv(url, row.names = 1)

# Create a list of 5 knots location for the bases
# Knots that can equally divide the tobacco range are chosen
# There are 107 out of 462 tobacco values are 0, 107/462 = 0.2316
# So the first knot need to be greater than 0.2316 to avoid NA values in glm
knots <- quantile(heart$tobacco, probs = c(0.24, 0.40, 0.55, 0.70, 0.85))

# B-spline base
b_spline <- bs(heart$tobacco, knots = knots)

# Natural spline base
n_spline <- ns(heart$tobacco, knots = knots)

# Truncated polynomial base
# Below is the function that creates truncated polynomial spline base
# it is modified based on Dr. Bolker's code
truncpolyspline <- function(x, knots) {
  trunc_fun <- function(k) (x > k)*(x-k)^3
  S <- sapply(knots, trunc_fun)
  S <- cbind(x, x^2, x^3, S)
  return(S)
}

poly_spline <- truncpolyspline(heart$tobacco, knots) # create basis matrix

# Fit logistic regression
# b-spline
logistic_b <- glm(chd ~ b_spline, data = heart, family = 'binomial')
# natural spline
logistic_n <- glm(chd ~ n_spline, data = heart, family = 'binomial')
```

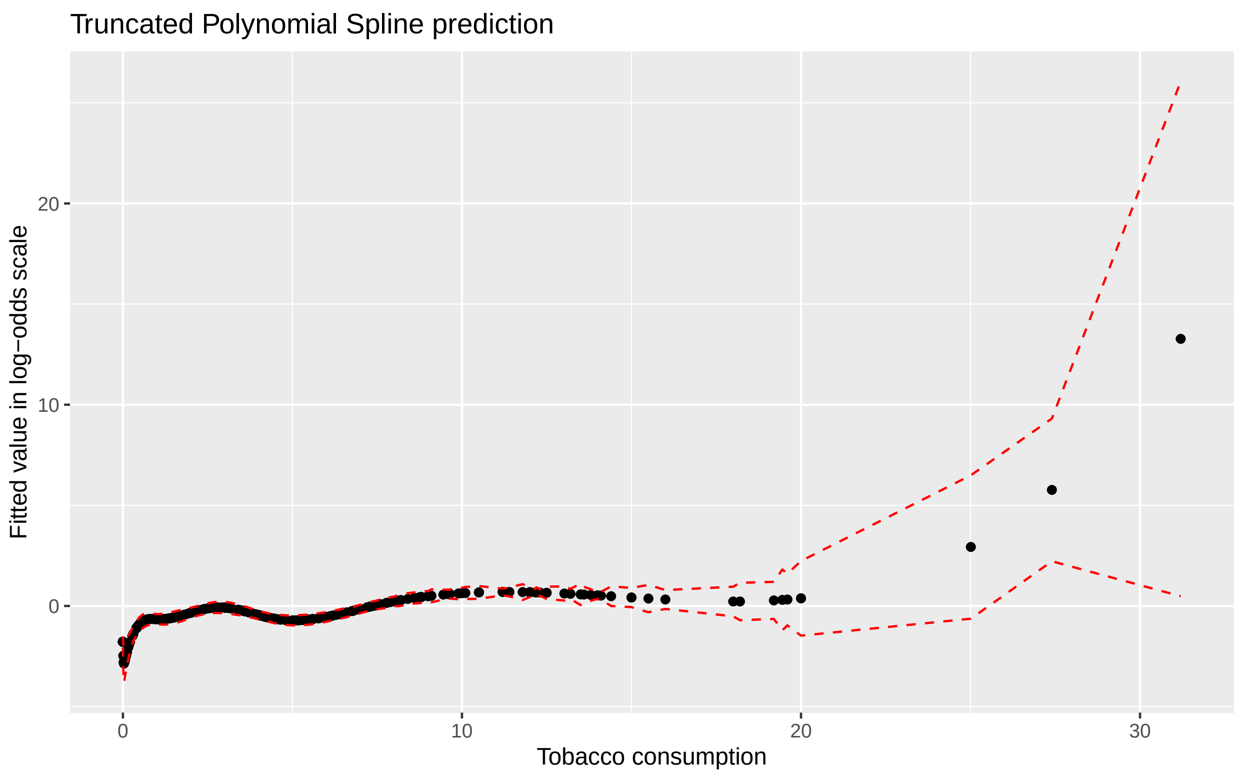
```r
# truncated polynomial spline
logistic_poly <- glm(chd ~ poly_spline, data = heart, family = 'binomial')

# Predict without using predict() function on log-odds scale
# b-spline
X_b <- model.matrix(logistic_b) # design matrix for b-spline
coef_b <- as.vector(logistic_b$coefficients) # coefficients vector for b-spline
Y_b <- X_b %*% coef_b # predicted value
var_Y_b <- diag(X_b %*% vcov(logistic_b) %*% t(X_b)) # predicted variance
se_Y_b <- as.matrix(sqrt(var_Y_b)) # predicted se
upper_b <- Y_b + se_Y_b # upper bound of the CI
lower_b <- Y_b - se_Y_b # lower bound of the CI

# Create a dataframe for b-spline x, predicted y, upper and lower bound of y.
df_b <- as.data.frame(cbind(heart$tobacco, Y_b, upper_b, lower_b))
colnames(df_b) <- c('tobacco', 'pred', 'upper', 'lower')

# Plot for b-spline
ggplot(df_b, aes(x=tobacco)) +  geom_point(aes(y = pred)) + # scatter plot for Y_b
  geom_line(aes(y = upper), linetype = 'dashed', color = 'red') + # add upper bound
  geom_line(aes(y = lower), linetype = 'dashed', color = 'red') +  # add lower bound
  labs(x = "Tobacco consumption", y = "Fitted value in log-odds scale",
       title = "B-Spline prediction") # add axis labels
```



```
# natural spline
X_n <- model.matrix(logistic_n) # design matrix for b-spline
coef_n <- as.vector(logistic_n$coefficients) # coefficients vector for b-spline
Y_n <- X_n %*% coef_n # predicted value
var_Y_n <- diag(X_n %*% vcov(logistic_n) %*% t(X_n)) # predicted variance
se_Y_n <- as.matrix(sqrt(var_Y_n)) # predicted se
upper_n <- Y_n + se_Y_n # upper bound of the CI
lower_n <- Y_n - se_Y_n # lower bound of the CI

# Create a dataframe for natural spline x, predicted y, upper and lower bound of y.
df_n <- as.data.frame(cbind(heart$tobacco, Y_n, upper_n, lower_n))
colnames(df_n) <- c('tobacco', 'pred', 'upper', 'lower')

# Plot for natural spline
ggplot(df_n, aes(x=tobacco)) +  geom_point(aes(y = pred)) + # scatter plot for Y_b
  geom_line(aes(y = upper), linetype = 'dashed', color = 'red') + # add upper bound
  geom_line(aes(y = lower), linetype = 'dashed', color = 'red') +  # add lower bound
  labs(x = "Tobacco consumption", y = "Fitted value in log-odds scale",
       title = "Natural Spline prediction") # add axis labels
```



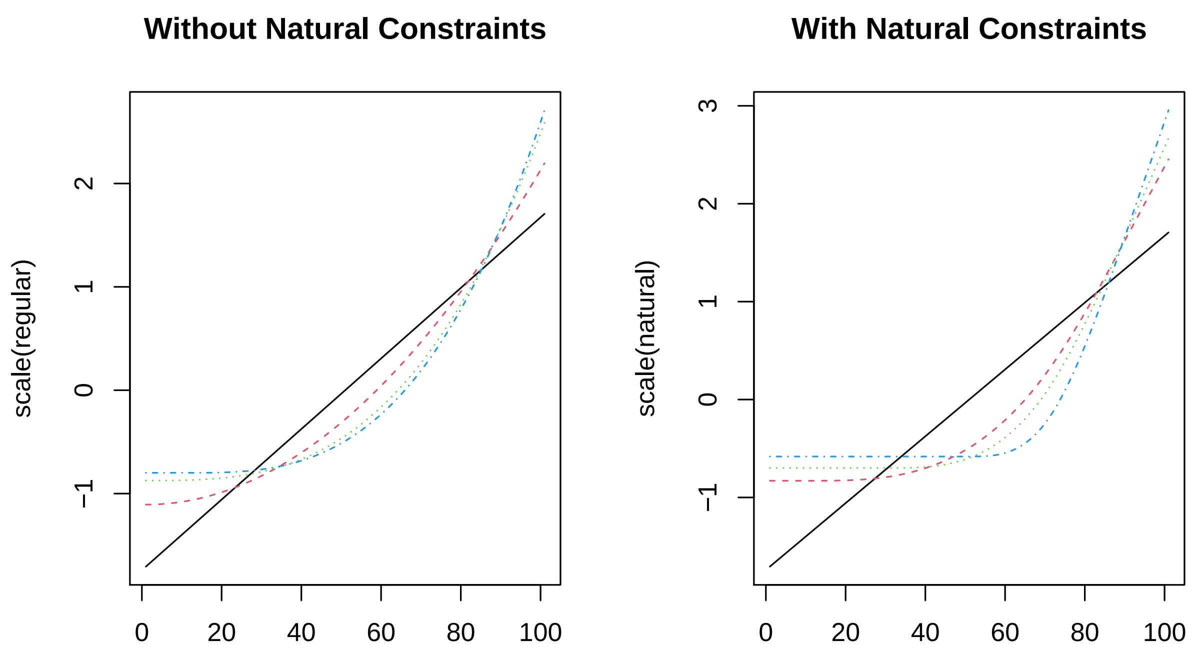
```r
# truncated polynomial spline
X_poly <- model.matrix(logistic_poly) # design matrix for b-spline
coef_poly <- as.vector(logistic_poly$coefficients) # coefficients vector for b-spline
Y_poly <- X_poly %*% coef_poly # predicted value
var_Y_poly <- diag(X_poly %*% vcov(logistic_poly) %*% t(X_poly)) # predicted variance
se_Y_poly <- as.matrix(sqrt(abs(var_Y_poly))) # predicted se
upper_poly <- Y_poly + se_Y_poly # upper bound of the CI
lower_poly <- Y_poly - se_Y_poly # lower bound of the CI

# Create a dataframe for truncated polynomial spline x, predicted y,
# upper and lower bound of y.
df_poly <- as.data.frame(cbind(heart$tobacco, Y_poly, upper_poly, lower_poly))
colnames(df_poly) <- c('tobacco', 'pred', 'upper', 'lower')

# Plot for truncated polynomial spline
ggplot(df_poly, aes(x=tobacco)) +  geom_point(aes(y = pred)) + # scatter plot for Y_b
  geom_line(aes(y = upper), linetype = 'dashed', color = 'red') + # add upper bound
  geom_line(aes(y = lower), linetype = 'dashed', color = 'red') +  # add lower bound
  labs(x = "Tobacco consumption", y = "Fitted value in log-odds scale",
    title = "Truncated Polynomial Spline prediction") # add axis labels
```



Question 3

```
# Write a function that adds natural spline constraint on truncated polynomial.
truncPoly <- function(x, df, natural = TRUE) {
  # x is the predictor
  # df is degree of freedom
  # natural is a boolean argument determining if natural constraint is needed
  if (natural){
    K <- df # df = K for natural, K is the number of knots
    knots <- quantile(x, seq(0.1, 0.9, length = K)) # obtain K knots
    # d_k(X) is a matrix with n rows and K-1 columns
    d_k <- matrix(NA, nrow = length(x), ncol = K-1)
    for (i in 1:(K-1)){
      numerator <- (x > knots[i])*(x-knots[i])^3 - (x > knots[K])*(x-knots[K])^3
      denominator <- knots[K]-knots[i]
      d_k[,i] <- numerator/denominator # ESL(5.5)
```

```r
    }
    # N_{k+2}(X) is a matrix with n rows and K-2 columns
    N_kplus2 <- matrix(NA, nrow = length(x), ncol = K-2)
    for (i in 1:(K-2)){
      N_kplus2[,i] <- d_k[,i] - d_k[,K-1] # ESL(5.4)
    }
    # Final matrix of basis, N_2(X) = X, N_{k+2}(X) defined as above
    N <- cbind(x, N_kplus2)
    return(N)
  }
  else {
    K <- df - 4 # df = K+4 for regular
    knots <- quantile(x, seq(0.1, 0.9, length = K)) # obtain K knots
    trunc_fun <- function(k) (x > k)*(x-k)^3
    S <- sapply(knots[1:(df-2)], trunc_fun)
    S <- cbind(x, x^2, x^3, S) # regular truncated polynomial spline
    return(S)
  }
}

# Plot an example of regular and natural bases
x <- seq(0, 10, length = 101) # example from lecture
regular <- truncPoly(x, df = 5, natural = FALSE) # regular bases
natural <- truncPoly(x, df = 5, natural = TRUE) # natural bases
# Plots
par(mfrow = c(1, 2))
matplot(scale(regular), type = "l", main = "Without Natural Constraints")
matplot(scale(natural), type = "l", main = "With Natural Constraints")
```



Question 4

```
# Part(a)
# Simulate data from a surface on the unit square with Gaussian noise
my_simulation <- function(n, noise_mean = 0, noise_sd = 1){
  # n is the number of observations required
  # noise_mean is the mean of the noise term, default is 0
  # noise_sd if the sd of the noise term, default is 1

  # draw numbers from unit square x-axis
  x <- runif(n)
  # draw numbers from unit square y-axis
  y <- runif(n)
  # noise term from Gaussian distribution
  noise <- rnorm(n, noise_mean, noise_sd)
  # create a smooth surface using a high-order bivariate polynomial
  z <- 5*x^7*y^6 + 2*x^6*y^5 + x^5 + 7*x^3*y^2 + 2*x^2*y + x*y^2 + y^3 + 3*y^4*x + y^8
  # smooth surface with gaussian noise
  z_noise <- z + noise
  # return a data frame with three columns: x, y, and z
  data.frame(x = x, y = y, z = z_noise)
}

# Example of the simulation
# Expect a list of 5 numbers that are generated using the polynomial above
# and with noises from a Gaussian distribution with mean 0.5 and sd 2.5.
my_simulation(5, 0.5, 2.5)$z
```

```
## [1]  5.2798148 -0.2362215  5.6060050  1.3019650 -2.0691686
```

```
# Part (b)
# Import libraries
library(mgcv)

# Function below returns the true value of z, used to compare with the prediction
true_z <- function(x, y){
  5*x^7*y^6 + 2*x^6*y^5 + x^5 + 7*x^3*y^2 + 2*x^2*y + x*y^2 + y^3 + 3*y^4*x + y^8
}

# 250 simulations
simulation_num <- 250

# (i) method = "GCV.Cp".
# Create an empty matrices to store the computed value for each simulation
comp_time1 <- matrix(NA, nrow=simulation_num, ncol=1)
bias1 <- matrix(NA, nrow=simulation_num, ncol=1)
variance1 <- matrix(NA, nrow=simulation_num, ncol=1)
mse1 <- matrix(NA, nrow=simulation_num, ncol=1)

for (i in 1:simulation_num) {
  # Simulate 500 random samples for each simulation with standard gaussian noise
  my_data <- my_simulation(500, 0, 1)

  # Fit a GAM model using z ~ te(x,y) formula and method = "GCV.Cp"
  start_time <- Sys.time() # time before running model
  model1 <- gam(z ~ te(x, y), data = my_data, method = "GCV.Cp")
  end_time <- Sys.time() # time after running model
  comp_time1[i] <- as.numeric(end_time - start_time) # computation time

  # Create test set using runif()
  x_test <- runif(50)
  y_test <- runif(50)
  test_set <- as.data.frame(cbind(x = x_test, y = y_test))

  # Prediction
  pred <- predict(model1, newdata = test_set) # predicted z value
  true <- true_z(x_test, y_test) # true value of z for comparison

  # Compute bias, variance, and MSE of the predictions
  bias1[i] <- mean(pred - true)
  variance1[i] <- mean((pred - mean(pred))^2)
  mse1[i] <- mean((pred - true)^2)
}

mean(comp_time1)
```

```
## [1] 0.01752807
```

```
mean(bias1)
```

```
## [1] -0.004085706
```

```
mean(variance1)
```

```
## [1] 8.676667
```

```
mean(mse1)
```

```
## [1] 0.05931893
```

```r
# (ii) method = "REML".
# Create an empty matrices to store the computed value for each simulation
comp_time2 <- matrix(NA, nrow=simulation_num, ncol=1)
bias2 <- matrix(NA, nrow=simulation_num, ncol=1)
variance2 <- matrix(NA, nrow=simulation_num, ncol=1)
mse2 <- matrix(NA, nrow=simulation_num, ncol=1)

for (i in 1:simulation_num) {
  # Simulate 500 random samples for each simulation with standard gaussian noise
  my_data <- my_simulation(500, 0, 1)

  # Fit a GAM model using z ~ te(x,y) formula and method = "REML"
  start_time <- Sys.time()
  model2 <- gam(z ~ te(x, y), data = my_data, method = "REML")
  end_time <- Sys.time()
  comp_time2[i] <- as.numeric(end_time - start_time) # computation time

  # Create test set using runif()
  x_test <- runif(50)
  y_test <- runif(50)
  test_set <- as.data.frame(cbind(x = x_test, y = y_test))

  # Prediction
  pred <- predict(model2, newdata = test_set) # predicted z value
  true <- true_z(x_test, y_test) # true value of z for comparison

  # Compute bias, variance, and MSE of the predictions
  bias2[i] <- mean(pred - true)
  variance2[i] <- mean((pred - mean(pred))^2)
  mse2[i] <- mean((pred - true)^2)
}

mean(comp_time2)
```

```
## [1] 0.04040532
```

```
mean(bias2)
```

```
## [1] -0.008017124
```

```
mean(variance2)
```

```
## [1] 8.024845
```

```
mean(mse2)
```

```
## [1] 0.05705356
```

```
# Create a comparison table for two methods
compare <- matrix(c(mean(comp_time1), mean(bias1), mean(variance1), mean(mse1),
                    mean(comp_time2), mean(bias2), mean(variance2), mean(mse2)),
                  ncol = 2, nrow = 4)
colnames(compare) <- c("GCV.Cp", "REML")
rownames(compare) <- c("computation time", "bias", "variance", "mse")
compare
```

```
##                        GCV.Cp         REML
## computation time  0.017528066  0.040405324
## bias             -0.004085706 -0.008017124
## variance          8.676666636  8.024844551
## mse               0.059318931  0.057053562
```

```
# Generally, both methods have similar computation time, bias, variance and mse.
# REML has slightly greater values in all outputs.
```

Question 5

ESL 5.4 The natural boundary conditions for natural cubic splines is that "the function is linear beyond the boundary knots". When X $< \xi_1$, f(x) = $\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$, to make it linear, we need β_2 and β_3 equal to 0. Alternatively, we can prove by taking the second derivative of f(x) and set it to 0. $f''(x) = 2\beta_2 + 6\beta_3 x = 0$, hence $\beta_2 = \beta_3 = 0$.

Similarly, the function f(x) should be linear when X $> \xi_k$ where f(x) $= \beta_0 + \beta_1 x + \sum_{k=1}^{K} \theta_k (X - \xi_k)^3$, we take second derivative and set it to 0. $f''(x) = 6\sum_{k=1}^{K} \theta_k (X - \xi_k) = 6(\sum_{k=1}^{K} \theta_k X - \sum_{k=1}^{K} \theta_k \xi_k) = 0$, then we have $\sum_{k=1}^{K} \theta_k X - \sum_{k=1}^{K} \theta_k \xi_k = 0$, which implies that $\sum_{k=1}^{K} \theta_k = 0$ and $\sum_{k=1}^{K} \theta_k \xi_k = 0$, as required.

Now we derive (5.4) and (5.5). The function we have is $f(x) = \beta_0 + \beta_1 x + \sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3 = 0$, by observing the function, we get that $N_1(X) = 1, N_2(X) = X$, we now want to prove that $\sum_{k=1}^{K} \theta_k (X - \xi_k)_+^3$ can be written is the form of $N_{k+2}(X) = d_k(X) - d_{k-1}(X)$.

Given that $\sum_{k=1}^{K} \theta_k = 0$ and $\sum_{k=1}^{K} \theta_k \xi_k = 0$, we know that $\sum_{k=1}^{K-2} \theta_k = -\theta_K - \theta_{K-1}$ and $\sum_{k=1}^{K-2} \theta_k \xi_k = -\theta_K \xi_K - \theta_{K-1}\xi_{K-1}$.

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

$$\begin{aligned}
\theta_{K-1}(X - \xi_{K-1})_+^3 &= \theta_{K-1}(X - \xi_{K-1})_+^3 \frac{\xi_{K-1} - \xi_K}{\xi_{K-1} - \xi_K} \\
&= \frac{(X - \xi_{K-1})_+^3}{\xi_{K-1} - \xi_K} (\theta_{K-1}\xi_{K-1} - \theta_{K-1}\xi_K) \\
&= \frac{(X - \xi_{K-1})_+^3}{\xi_{K-1} - \xi_K} (\theta_{K-1}\xi_{K-1} - \theta_{K-1}\xi_K + \theta_K \xi_K - \theta_K \xi_K) \\
&= \frac{(X - \xi_{K-1})_+^3}{\xi_{K-1} - \xi_K} (\theta_{K-1}\xi_{K-1} + \theta_K \xi_K - \theta_{K-1}\xi_K - \theta_K \xi_K) \\
&= \frac{(X - \xi_{K-1})_+^3}{\xi_{K-1} - \xi_K} (-\sum_{k=1}^{K-2} \theta_k \xi_k + \xi_K \sum_{k=1}^{K-1} \theta_k) \\
&= \frac{(X - \xi_{K-1})_+^3}{\xi_{K-1} - \xi_K} \sum_{k=1}^{K-2} \theta_k (\xi_K - \xi_k) \\
&= \sum_{k=1}^{K-2} \theta_k (\xi_K - \xi_k) \frac{(X - \xi_{K-1})_+^3}{\xi_{K-1} - \xi_K}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\theta_K (X - \xi_K)_+^3 &= \frac{(X - \xi_K)_+^3}{\xi_{K-1} - \xi_K} \theta_K (\xi_{K-1} - \xi_K) \\
&= \frac{(X - \xi_K)_+^3}{\xi_{K-1} - \xi_K} (\theta_K \xi_{K-1} - \theta_K \xi_K + \theta_{K-1}\xi_{K-1} - \theta_{K-1}\xi_{K-1}) \\
&= \frac{(X - \xi_K)_+^3}{\xi_{K-1} - \xi_K} (-\xi_{K-1} \sum_{k=1}^{K-2} \theta_K + \sum_{k=1}^{K-2} \theta_k \xi_k) \\
&= \sum_{k=1}^{K-2} \theta_k (\xi_k - \xi_{K-1}) \frac{(X - \xi_K)_+^3}{\xi_{K-1} - \xi_K}
\end{aligned}$$

Put them back together and get:

$$\begin{aligned}
\sum_{k=1}^{K}\theta_k(X-\xi_k)_+^3 &= \sum_{k=1}^{K-2}\theta_k(X-\xi_k)_+^3+\theta_{K-1}(X-\xi_{K-1})_+^3+\theta_K(X-\xi_K)_+^3 \\
&= \sum_{k=1}^{K-2}\theta_k(X-\xi_k)_+^3+\sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)\frac{(X-\xi_{K-1})_+^3}{\xi_{K-1}-\xi_K}+\sum_{k=1}^{K-2}\theta_k(\xi_k-\xi_{K-1})\frac{(X-\xi_K)_+^3}{\xi_{K-1}-\xi_K} \\
&= \sum_{k=1}^{K-2}\theta_k[(X-\xi_k)_+^3+(\xi_K-\xi_k)\frac{(X-\xi_{K-1})_+^3}{\xi_{K-1}-\xi_K}+(\xi_k-\xi_{K-1})\frac{(X-\xi_K)_+^3}{\xi_{K-1}-\xi_K}] \\
&= \sum_{k=1}^{K-2}\theta_k[(X-\xi_k)_+^3\frac{\xi_K-\xi_k}{\xi_K-\xi_k}+(\xi_K-\xi_k)\frac{(X-\xi_{K-1})_+^3}{\xi_{K-1}-\xi_K}+(\xi_k-\xi_{K-1})\frac{(X-\xi_K)_+^3}{\xi_{K-1}-\xi_K}\frac{\xi_K-\xi_k}{\xi_K-\xi_k}] \\
&= \sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)[\frac{(X-\xi_k)_+^3}{\xi_K-\xi_k}+\frac{(X-\xi_{K-1})_+^3}{\xi_{K-1}-\xi_K}+(\xi_k-\xi_{K-1})\frac{(X-\xi_K)_+^3}{(\xi_{K-1}-\xi_K)(\xi_K-\xi_k)}] \\
&= \sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)[\frac{(X-\xi_k)_+^3}{\xi_K-\xi_k}+\frac{(X-\xi_{K-1})_+^3}{\xi_{K-1}-\xi_K}+(X-\xi_K)_+^3(\frac{-1}{\xi_{K-1}-\xi_K}-\frac{1}{\xi_K-\xi_k})] \\
&= \sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)[\frac{(X-\xi_k)_+^3-(X-\xi_K)_+^3}{\xi_K-\xi_k}+\frac{(X-\xi_{K-1})_+^3-(X-\xi_K)_+^3}{\xi_{K-1}-\xi_K}] \\
&= \sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)[\frac{(X-\xi_k)_+^3-(X-\xi_K)_+^3}{\xi_K-\xi_k}-\frac{(X-\xi_{K-1})_+^3-(X-\xi_K)_+^3}{\xi_K-\xi_{K-1}}] \\
&= \sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)(d_k(X)-d_{K-1}(X)) \\
&= \sum_{k=1}^{K-2}\theta_k(\xi_K-\xi_k)N_{k+2}(X)
\end{aligned}$$

Therefore, to conclude, we get $N_1(X)=1, N_2(X)=X, N_{k+2}(X)=d_k(X)-d_{K-1}(X)$ as desired in (5.4) and (5.5).

ESL 5.13

References