HW 3

Wenjuan Bian

07/17/2023

Provide your answers and relevant outputs below each question in this markdown file and submit the knitted `html` or `pdf` file to Canvas. Put your answers/comments/conclusions in **bold**.

Problem 1

Consider the `gasoline` dataset (providing information for $N = 60$ gasoline samples) from the `refund` package. The goal is to build a regression model to predict `octane` number of a gasoline sample using its near-infrared reflectance (NIR) spectral curve.

Part (a)

Plot the 60 spectral curves using the `fda::matplot()` function (within it, set `type='l'`). The curve values are stored in a 60 by 401 matrix and you would first need to transpose it so that the columns correspond to observations and the rows to the curve values at different points $t_j, j = 1, 2, \ldots, 401$.

```
library(refund)
library(fda)
```

```
## Warning: package 'fda' was built under R version 4.2.3
```

```
## Loading required package: splines
```

```
## Loading required package: fds
```

```
## Warning: package 'fds' was built under R version 4.2.3
```

```
## Loading required package: rainbow
```

```
## Warning: package 'rainbow' was built under R version 4.2.3
```

```
## Loading required package: MASS
```

```
## Loading required package: pcaPP
```

```
## Warning: package 'pcaPP' was built under R version 4.2.3
```

```
## Loading required package: RCurl
```

```
## Warning: package 'RCurl' was built under R version 4.2.3
```

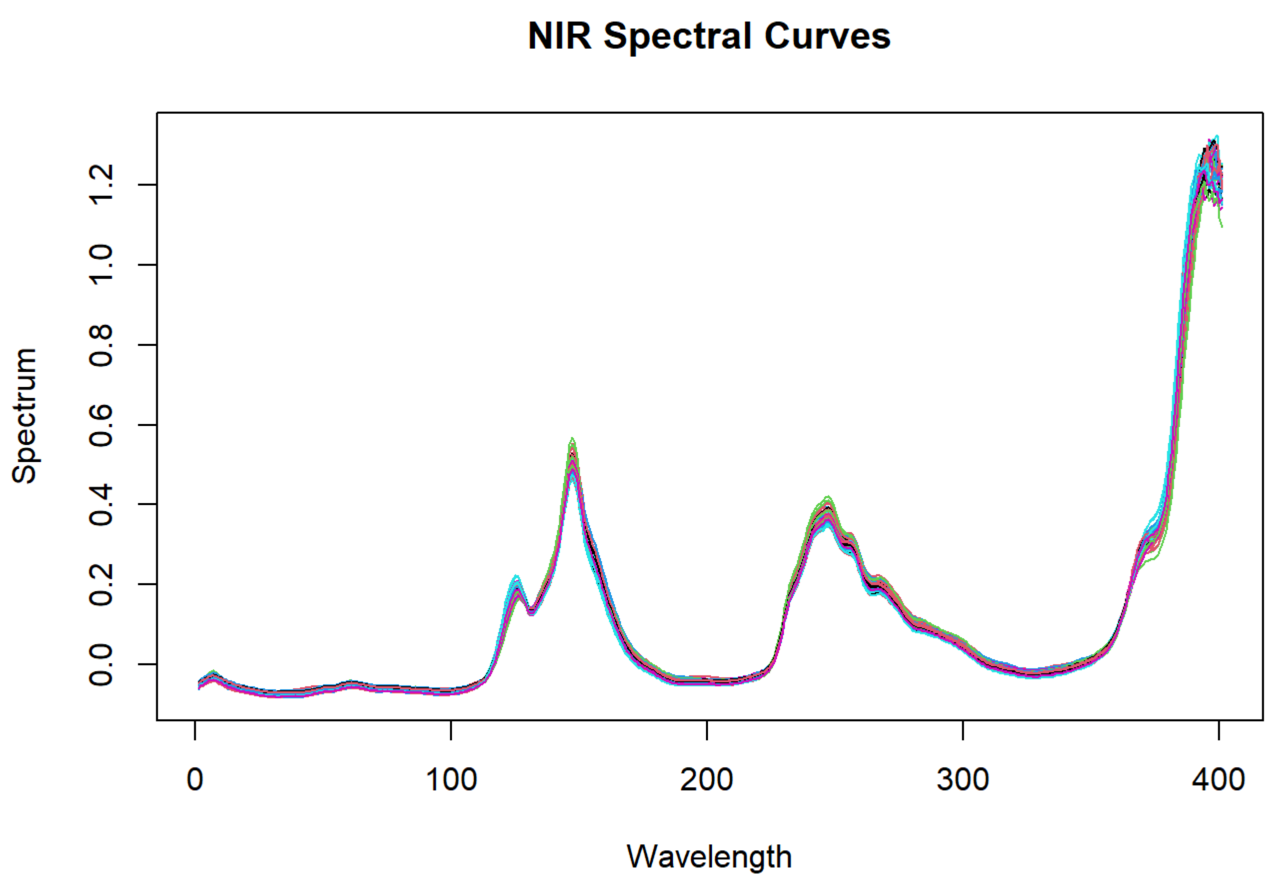
```
## Loading required package: deSolve
```

```
## Warning: package 'deSolve' was built under R version 4.2.3
```

```
##
## Attaching package: 'fda'
```

```
## The following object is masked from 'package:graphics':
##
##     matplot
```

```
matplot(t(gasoline$NIR), type='l', lty=1,
        main="NIR Spectral Curves",
        xlab="Wavelength",
        ylab="Spectrum")
```



Part (b)

Take the first 45 observations for training and the remaining for testing. Let `Ytrain` and `Ytest` be the vectors of octane numbers of the training and test sets. Let `Xtrain` and `Xtest` be the matrices (of dimensions 45 by 401 and 15 by 401) containing the 401 NIP spectrum measurements for each sample in the training and test sets respectively.

```
Ytrain <- gasoline$octane[1:45]
Xtrain <- gasoline$NIR[1:45, ]

Ytest <- gasoline$octane[46:60]
Xtest <- gasoline$NIR[46:60, ]
```

Part (c) (Basis)

Fit a model (without a penalty term) to the training set where the coefficient function $\beta(t)$ is represented as a linear combination of some basis functions. Experiment with different types of basis functions (using the `bs` argument of the `lf()` function) and their number (using the `k` argument of the `lf()` function). After your model is fit, find its predictions for the observations in the test set (use the `predict()` function for that) and compute the test mean absolute error (MAE) ($\frac{1}{15}\sum_{i=1}^{15}|y_i - \hat{y}_i|$). Also, plot the graph of the estimated coefficient function $\hat{\beta}(t)$.

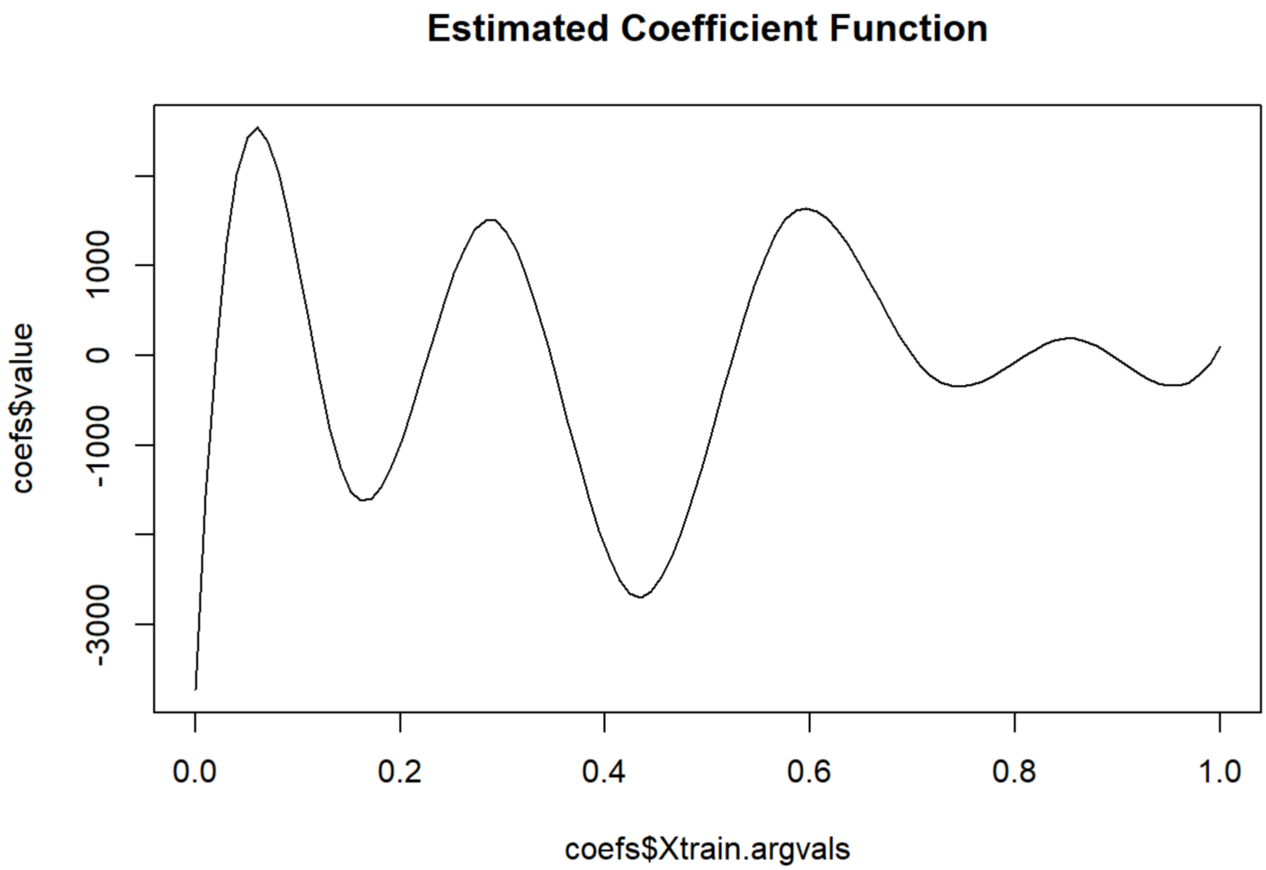
```
fit.lin = pfr(Ytrain ~ lf(Xtrain, bs = "ps", k = 10,fx = TRUE))


preds = predict(fit.lin, newdata = list(Xtrain = Xtest))


mae = mean(abs(Ytest - preds))


print(mae)
```

```
## [1] 0.230225
```

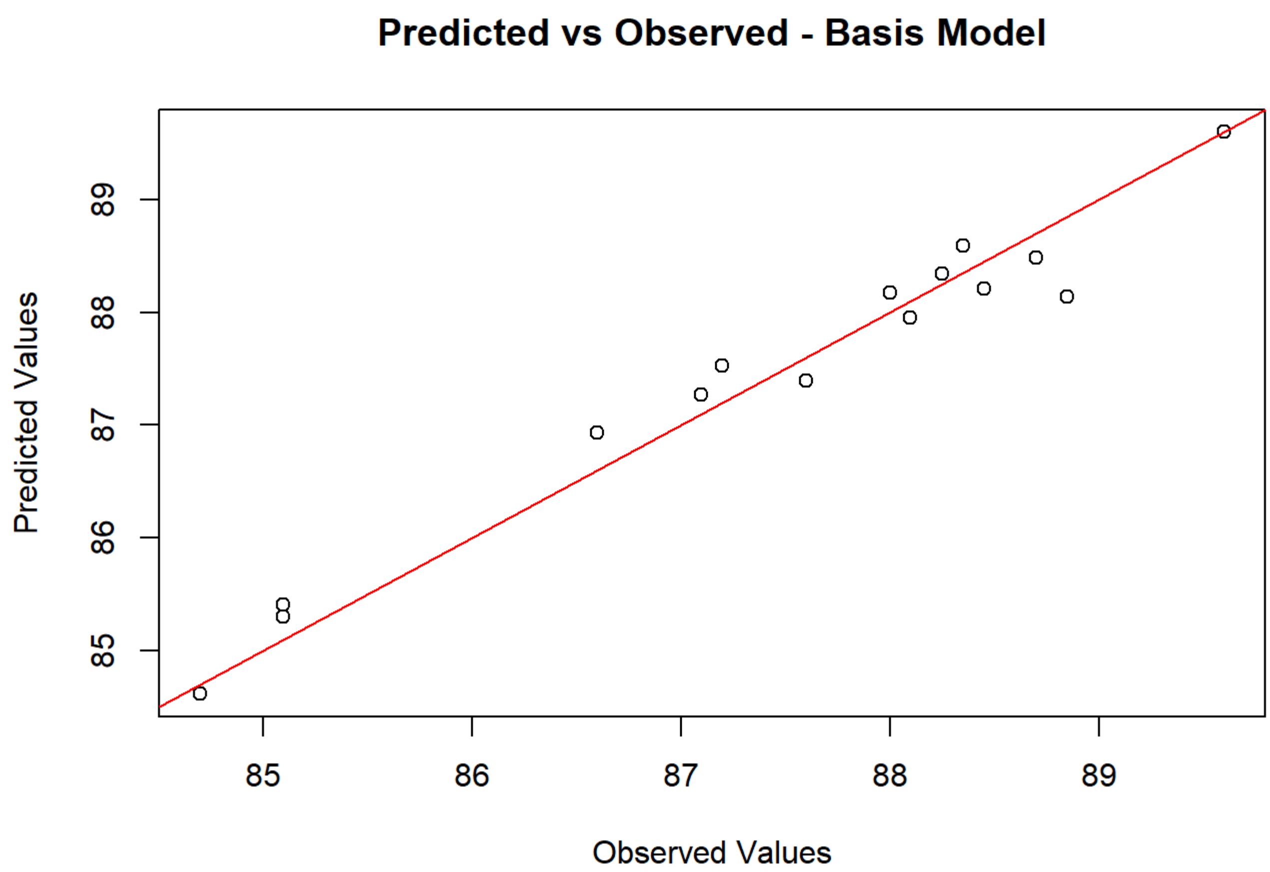
```
coefs <- coef(fit.lin)
print(str(coefs))
```

```
## 'data.frame':    100 obs. of  3 variables:
##  $ Xtrain.argvals: num  0 0.0101 0.0202 0.0303 0.0404 ...
##  $ value         : num [1:100, 1] -3724.8 -1565.8 76.5 1254.9 2022.5 ...
##  $ se            : num  3324 2190 1741 1827 2050 ...
## NULL
```

```
plot(coefs$Xtrain.argvals, coefs$value, type = "l", main = "Estimated Coefficient Function")
```




```
plot(Ytest, preds, main = "Predicted vs Observed - Basis Model", xlab = "Observed Values", ylab
= "Predicted Values")
abline(0, 1, col = "red")
```



Part (d) (Penalized)

Repeat part (c) by imposing a roughness penalty. In this case, set the number of basis functions, k, to a large enough number (it is often recommended to use k equal to the number of points t_j at which curves X_i are observed). Use either `method='REML'` or `method = 'GCV.Cp'` inside the `pfr()` function as your smoothing parameter estimation method.

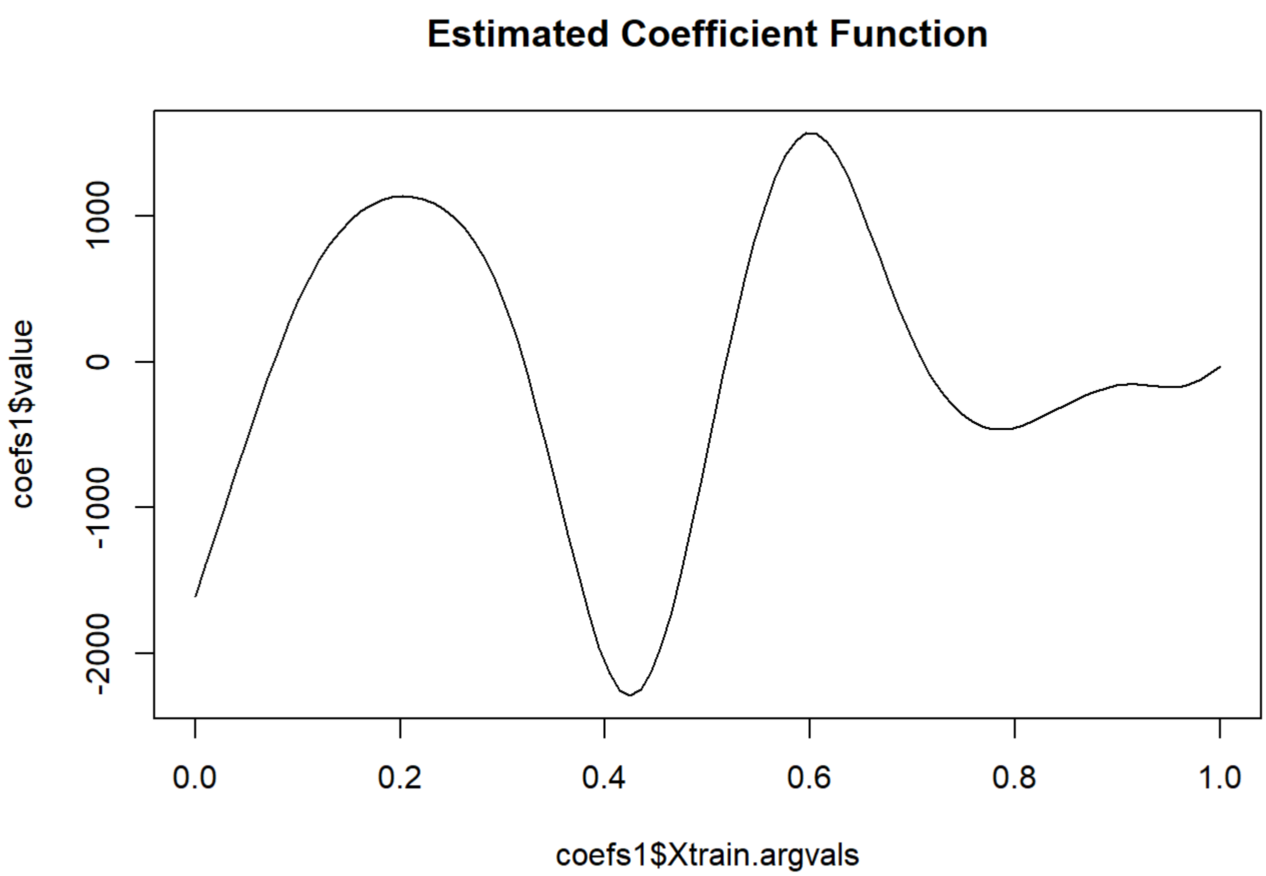
```
fit.pfr = pfr(Ytrain ~ lf(Xtrain, bs = "ps", k = 44), method = "REML")

preds1 = predict(fit.pfr, newdata = list(Xtrain = Xtest))

mae1 = mean(abs(Ytest - preds1))

print(mae1)
```

```
## [1] 0.2103946
```

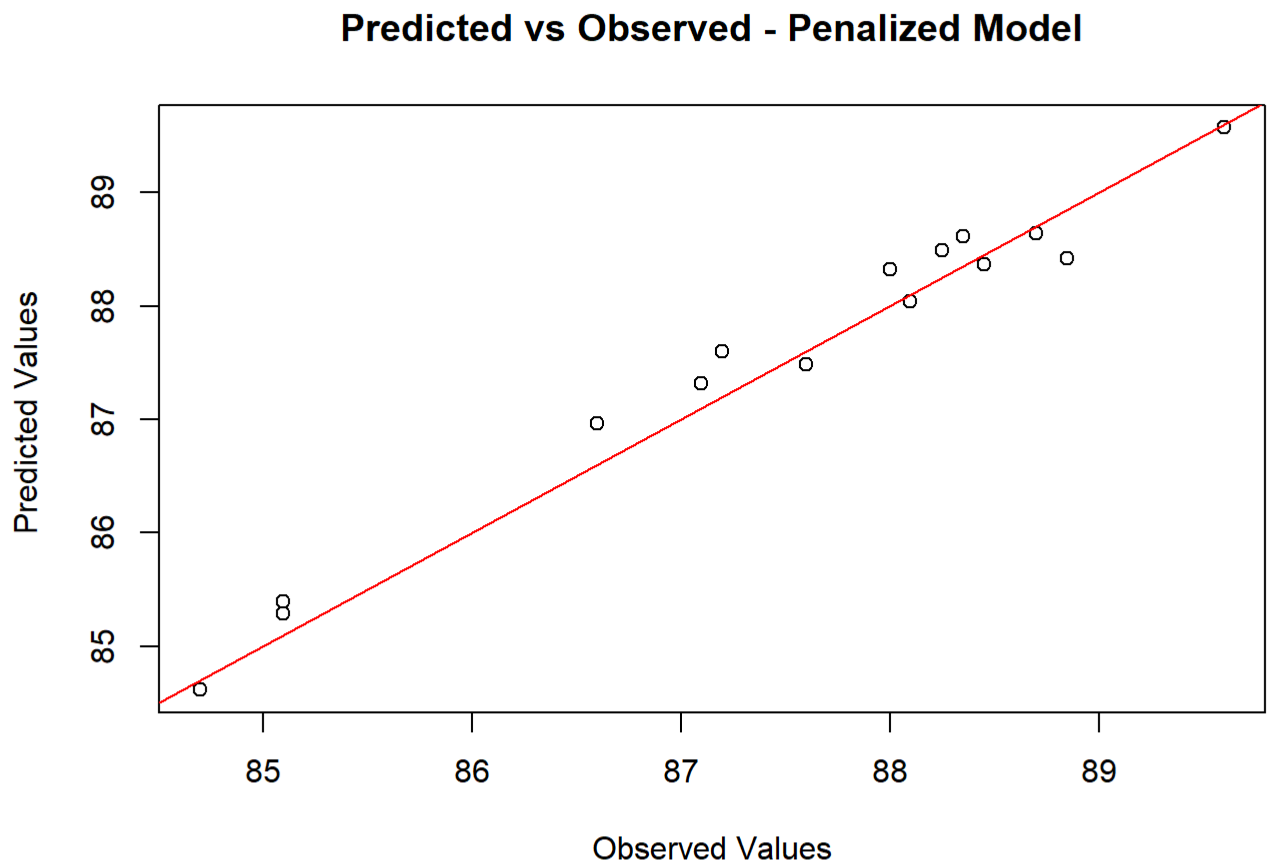
```
coefs1 <- coef(fit.pfr)
print(str(coefs1))
```

```
## 'data.frame':    100 obs. of  3 variables:
##  $ Xtrain.argvals: num  0 0.0101 0.0202 0.0303 0.0404 ...
##  $ value         : num [1:100, 1] -1608 -1389 -1171 -955 -740 ...
##  $ se            : num  1910 1648 1413 1207 1031 ...
## NULL
```

```
plot(coefs1$Xtrain.argvals, coefs1$value, type = "l", main = "Estimated Coefficient Function")
```




```
plot(Ytest, preds1, main = "Predicted vs Observed - Penalized Model", xlab = "Observed Values",
ylab = "Predicted Values ")
abline(0, 1, col = "red")
```



Part (e) (FPC)

This part is about building a regression model using functional principal components (FPC). Although we can fit such a model using the `fpc()` function inside `pfr()`, in class we had trouble with the `predict()` function to compute predictions for new test observations. Here, you'll fit a FPC model manually following the steps below:

- Using the `Data2fd()` function convert the raw data stored in `Xtrain` and `Xtest` into functional objects. Use the first **ten** B-splines basis functions. Set `grid <- seq(0,1,length.out=401)` (grid of 401 t_j points). Save the outputs as `Xfdtrain` and `Xfdtest`.

```
library(fda)

grid <- seq(0, 1, length.out = 401)

mybasis <- create.bspline.basis(rangeval = range(grid), nbasis = 10)

Xfdtrain <- Data2fd(argvals = grid, y = t(Xtrain), mybasis)

Xfdtest <- Data2fd(argvals = grid, y = t(Xtest), mybasis)
```

- Using the `pca.fd()` function compute the first **four** functional principal components (FPC) for `Xfdtrain`. Save the output as `FPCtrain`.

```
FPCtrain <- pca.fd(Xfdtrain, nharm = 4)
```

- `FPCtrain$scores` is the 45 by 4 matrix of scores ($\hat{\xi}_{ij}^{(train)}$) of the sample covariance operator for the training set. Save it as `Sctrain`.

```
Sctrain <- FPCtrain$score
```

- Using the `lm()` function fit a linear model with formula `Ytrain ~ Sctrain`. Print the summary of the model.

```
fit.lm <- lm(Ytrain ~ Sctrain)
summary(fit.lm)
```

```
##
## Call:
## lm(formula = Ytrain ~ Sctrain)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.36805 -0.13006 -0.02595  0.14915  0.37123
##
## Coefficients:
##              Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   87.08778    0.02954 2948.497  < 2e-16 ***
## Sctrain1      57.60973    3.63053   15.868  < 2e-16 ***
## Sctrain2      24.29535   10.27618    2.364    0.023 *
## Sctrain3    -816.96725   16.97348  -48.132  < 2e-16 ***
## Sctrain4    -212.55354   23.55002   -9.026 3.41e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1981 on 40 degrees of freedom
## Multiple R-squared:  0.9852, Adjusted R-squared:  0.9837
## F-statistic: 663.9 on 4 and 40 DF,  p-value: < 2.2e-16
```

- (Bonus) Compute the 15 by 4 matrix of scores ($\hat{\xi}_{ij}^{(test)}$) for the test set using the first four FPCs computed off of the training set. (Hint: $\hat{\xi}_{ij}^{(test)} = \langle X_i^{(test)} - \hat{\mu}(t)^{(test)}, \hat{v}_j^{(train)} \rangle$. In R, the inner product can be computed using the `inprod()` function of the `fda` package). Save the outcome as `Sctest`.

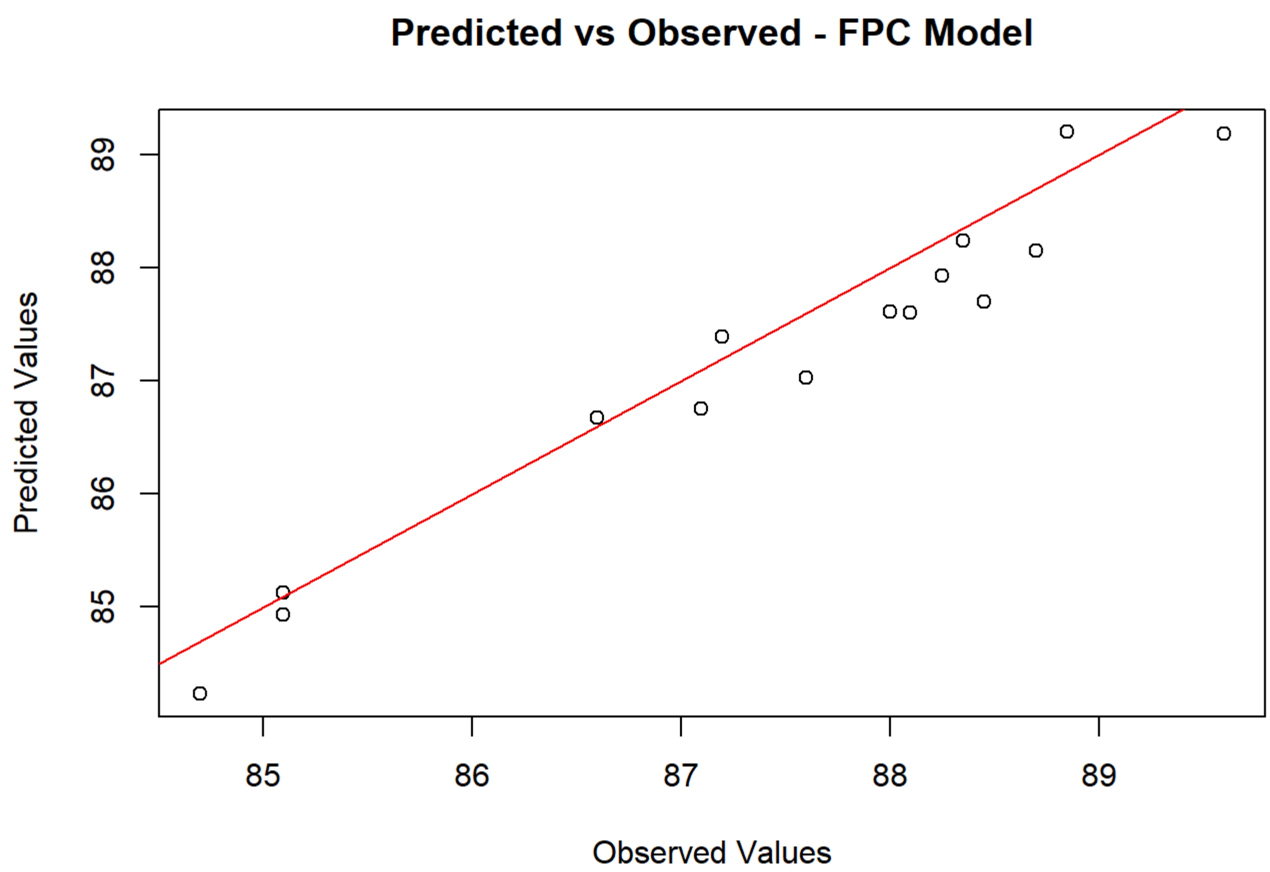
```
cumsum(FPCtrain$varprop)
```

```
## [1] 0.8324419 0.9363457 0.9744307 0.9942146
```

```
first4 <- FPCtrain$harmonics[,1:4]

mu.test <- rowMeans(Xfdtest$coefs)

Sctest <- matrix(NA, nrow = ncol(Xfdtest$coefs), ncol = 4)

for (i in 1:ncol(Xfdtest$coefs)) {
  for (j in 1:4) {
    test.c <- Xfdtest$coefs[,i] - mu.test

    test.c <- matrix(test.c, nrow = length(test.c), ncol = 1)

    test.c.fd <- fd(test.c, Xfdtest$basis)

    jcoefs <- first4$coefs[,j]

    jcoefs <- matrix(jcoefs, nrow = length(jcoefs), ncol = 1)

    eigen.j <- fd(jcoefs, Xfdtrain$basis)

    Sctest[i, j] <- inprod(test.c.fd, eigen.j)
  }
}
```

- (Bonus) Using the `predict()` function find the predictions of the fitted linear model passing `Sctest` to the `newdata` argument. Also compute corresponding test MAE.

```
data6 <- as.data.frame(Sctest)
names(data6) <- c("PC1", "PC2", "PC3", "PC4")

fit.lm2 <- lm(Ytrain ~ PC1 + PC2 + PC3 + PC4, data = as.data.frame(cbind(PC1 = FPCtrain$scores[,
1], PC2 = FPCtrain$scores[,2], PC3 = FPCtrain$scores[,3], PC4 = FPCtrain$scores[,4], Ytrain = Yt
rain)))


Ypred <- predict(fit.lm2, newdata = data6)

plot(Ytest, Ypred, xlab = "Observed Values", ylab = "Predicted Values", main = "Predicted vs Obs
erved - FPC Model")
abline(a = 0, b = 1, col = "red")
```




```
MAE3 <- mean(abs(Ytest - Ypred))

print(MAE3)
```

```
## [1] 0.3499533
```

Part (f)

Based on your findings which model would you recommend and why?

```
# Create the data frame
df <- data.frame(
  Model = c("Basis", "Penalized", "FPC"),
  MAE = c(mae, mae1, MAE3)
)

# Print the data frame
print(df)
```

```
##       Model       MAE
## 1     Basis 0.2302250
## 2 Penalized 0.2103946
## 3       FPC 0.3499533
```

Based on the Mean Absolute Error (MAE) values shown above, the ‘Penalized’ model has the lowest MAE, which suggests that, on average, its predictions deviate less from the observed values compared to the ‘Basis’ and ‘FPC’ models. Lower MAE is generally better as it indicates less error in the model’s predictions.

Furthermore, the plot shows a closer fit between the predicted and actual values for the ‘Penalized’ model compared to the other two models, this would provide additional evidence supporting the ‘Penalized’ model’s superior performance.

Therefore, I would recommend the ‘Penalized’ model. Its lower MAE suggests it has better predictive accuracy, and the consistent evidence from the plots further supports this recommendation.