```r
#First Homework Assignment
library(tidyverse)
library(R.matlab)
library(splines)
library(glmnet)
library(GGally)
library(patchwork)
library(splines2)

#set seed
set.seed(123456)

#Problem 3
#we are asked to write a computer program to sample a function f with a
#given analytical form at some given time points in [0,1].
#part a

#define a function
f1 <- function(t){
  2*sin(2*pi*t) + cos(4*pi*t)
}

n <- 10
t_points <- seq(0, 1, length.out = n)
y_values <- sapply(t_points, f1)

#function to create a design matrix for a given Fourier Basis, assuming the 3
#Fourier basis functions provided, asuming K = odd >= 3
design_matrix <- function(t, K){
  t <- as.numeric(t)
  n <- length(t)

  #number of sine/cosine pairs (subtract the intercept and divide by 2)
  #if K = 3, then there is 1, K=5, then there is two, etc
  J <- (K - 1) %/% 2

  X <- matrix(0, nrow = n, ncol = K)

  #first Fourier basis function is 1
  X[,1] <- 1

  col <- 2

  #fills sin/cosine columns of Fourier design matrix one at a time
  #J = # of pairs to include
  for (k in 1:J) {
    X[, col]   <- sqrt(2)*sin(2*pi*k*t)
    col <- col + 1
    X[, col]   <- sqrt(2)*cos(2*pi*k*t)
    col <- col + 1
  }

  return(X)
}

#penalty matrix assuming quadtratic penalty and K ?>= 3
fourier_penalty <- function(K, order = 2) {
  #number of sine/cosine pairs (subtract the intercept and divide by 2)
  #if K = 3, then there is 1, K=5, then there is two, etc
  J <- (K - 1) %/% 2

  P <- diag(0,K)

  #intercept is unpenalized
```

```
  #iterate of sin/cos pairs
  for (j in 1:J){
    #term for each "block"
    term <- (2*pi*j)^4

    #sin term
    #need the 2j-1 term or else the sin term goes in the cosine slot
    P[1+(2*j - 1), 1+(2*j - 1)] <- term

    #cosine term
    #don't need the -1 addeded here
    P[1+(2*j), 1+(2*j)] <- term
  }

  return(P)
}

#function for parts b and c
curve_fit <- function(t, K, Phi, W, lambda, R, y){

}

#part b and c
#defining basis functions, creating design matrix and penalty matrix
K <- 5
lambda.grid <- seq(0, 0.0675, length.out = 10)
Phi <- design_matrix(t_points, K)
P <- fourier_penalty(K=K)

#the dense predictions grid
t_dense <- seq(0, 1, length.out = 5*n)
Phi_dense <- design_matrix(t_dense, K)
y_true_dense <- f1(t_dense)

#precompute
XtX <- crossprod(Phi)
Xty <- crossprod(Phi, y_values)

#estimate the function for the various values of lambda
fits <- lapply(lambda.grid, function(lam) {
  A <- XtX + (lam*P)
  c_hat <- solve(A, Xty)
  y_hat <- as.vector(Phi_dense %*% c_hat)
  data.frame(t = t_dense,
             predicted = y_hat,
             lambda = lam,
             lambda_lab = format(lam, scientific = TRUE, digits = 2),
             stringsAsFactors = FALSE)
})

#bind the rows and turns the lambda into labels via scientific notation out to 2 digits
df_pred <- bind_rows(fits) %>%
  mutate(lambda_lab = factor(format(lambda, scientific = TRUE, digits = 2),
                             levels = unique(format(lambda.grid, scientific = TRUE, digits
= 2))))

df_true   <- data.frame(t = t_dense, true = y_true_dense)
df_points <- data.frame(t = t_points, y = y_values)

estimation.plot <- ggplot() +
  #draw the true function as a black line
  geom_line(data = df_true, aes(x = t, y = true), linewidth = 1.2, color = "black") +
  #add a seperate linetype, color for each lambda
  geom_line(data = df_pred,
            aes(x = t, y = predicted, color = lambda_lab, linetype = lambda_lab),
```

```
          linewidth = 0.9, alpha = 0.9) +
  #plot observed data points as big points, but remove aesthetics from the previous layers
  geom_point(data = df_points, aes(x = t, y = y), size = 2, alpha = 0.8, inherit.aes =
FALSE) +
  #Name legend
  scale_color_discrete(name = expression(lambda)) +
  scale_linetype_discrete(name = expression(lambda)) +
  #Adapt title for various Basis functions used
  labs(title = sprintf("Penalized Fourier LS: True vs Predictions (K = %d)", K),
       x = "t", y = "f(t)") +
  theme_dark(base_size = 13) +
  #make legend key wider.  Might help since we use different line types
  theme(legend.title = element_text(),
        legend.key.width = unit(1.6, "lines"))
estimation.plot

#Problem 4
#Write code to do FPCA
fpca_cust <- function(Ft, t, K = 3){
  n  <- nrow(Ft); m <- ncol(Ft)

  #get the mean function by column
  mu <- colMeans(Ft)

  #plot, want plot of mean function
  df_mu <- data.frame(t = as.vector(t), mu = mu)
  mean_plot <- ggplot(df_mu, aes(x = t, y = mu)) +
    geom_line(linewidth = 1) +
    labs(title = "Mean Function", x = "t", y = "Mean f(t)") +
    theme_minimal()
  print(mean_plot)

  #center the function
  F.centered <- scale(Ft, scale=F)

  #assuming evenly spaced time points, for later
  dt <- diff(as.vector(t))[1]

  #svd decomp
  n <- nrow(F.centered)
  sv <- svd(F.centered/sqrt(n))

  #get the principal directions
  psi <- sv$v

  #square the singular values to get eigenvalues
  #dt bc we are approximating the inner product integral with a sum
  lambda  <- (sv$d)^2 * dt

  #computing coefficients
  #get one score for each function
  scores <- F.centered %*% (psi * dt)

  #getting into the plotting mechanisms now
  #plotting raw data first
  df_mu <- data.frame(t = t, mu = mu)
  df_raw <- data.frame(
    id    = factor(rep(seq_len(n), times = m)),
    t     = rep(t, each = n),
    value = as.vector(Ft)
  )

  raw_plot <- ggplot(df_raw, aes(x = t, y = value, group = id)) +
    geom_line(alpha = 0.25) +
    labs(title = "Raw Functions", x = "t", y = "f(t)") +
```

```
    theme_minimal()
  print(raw_plot)


  #getting the K dominant directions
  psiK <- psi[, seq_len(K), drop = FALSE]
  colnames(psiK) <- paste0("PC", seq_len(K))
  df_psi <- data.frame(t = as.vector(t), psiK)

  #pivotting longer for plotting
  psi_long <- tidyr::pivot_longer(df_psi, -t, names_to = "Direction", values_to = "Value")
  eigen.vector_plot <- ggplot(psi_long, aes(x = t, y = Value, color = Direction)) +
    geom_line(linewidth = 1) +
    labs(title = paste0("Leading FPCA Directions (Top ", K, ")"),
         x = "t", y = "Eigenfunction Value") +
    theme_minimal()

  print(eigen.vector_plot)

  #scree plot
  df_lambda <- data.frame(Index = seq_along(lambda), Eigenvalue = lambda)

  scree_plot <- ggplot(df_lambda, aes(x = Index, y = Eigenvalue)) +
    geom_point(size = 2) +
    geom_line() +
    labs(title = "Eigenvalues (Scree Plot)", x = "Component", y = "Eigenvalue") +
    theme_minimal()

  print(scree_plot)

  #print the variance explained by the K PCs
  total_var <- sum(lambda)
  pve <- lambda / total_var
  cum_pve <- cumsum(pve)

  cat("\nProportion of Variance Explained (PVE):\n")
  cat(paste(sprintf("  PC%-2d: %6.2f%%   |  Cumulative: %6.2f%%",
                    1:K, 100*pve[1:K], 100*cum_pve[1:K]),
            collapse = "\n"), "\n\n")

  #doing the correlation plot
  scores_df <- as.data.frame(scores[, seq_len(K), drop = FALSE])
  colnames(scores_df) <- paste0("Coefficient:", seq_len(K))

  top_coef <- ggpairs(scores_df, title = paste("Pairwise Scatter of Top", K, "FPCA
Coefficients"))
  print(top_coef)

  #combining everything
  print(raw_plot / (mean_plot + eigen.vector_plot))
}

#read in data
#dataset 1
prob4.dat.1 <- readMat("Datasets/HW1/Problem 4/DataFile1_0_1.mat")

#implementing FPCA.  Need to center the functions
Fmatrix <- prob4.dat.1$f
ts <- prob4.dat.1$t

prob4.ques1 <- fpca_cust(F = Fmatrix, t = ts)


#now do this for the second dataset in the question
prob4.dat.2 <- readMat("Datasets/HW1/Problem 4/DataFile1_0_2.mat")
```

```
#implementing FPCA.  Need to center the functions
Fmatrix <- prob4.dat.2$f
ts <- prob4.dat.2$t

prob4.ques2 <- fpca_cust(Ft = Fmatrix, t = ts)


#problem 5
#Generate functional data as follows:  let f_i(t) = a_iPhi_1(t) + b_iPhi_(t)
#where phi_1(t) = sqrt(cos(wi*t)), phi_2(t) = sqrt(2)cos(3*pi*t)
#a_i, b_i ~ N(0,1)

prob5 <- function(n = 20, m = 50){
  #create time vector
  t <- seq(0,1,length.out=m)

  #phi functions
  phi1 <- sqrt(2)*cos(pi*t)
  phi2 <- sqrt(2)*cos(3*pi*t)

  #random coefficients
  a <- rnorm(n)
  b <- rnorm(n)

  #putting this together to create functional data
  F.dat <- matrix(NA, nrow = n, ncol = m)
  for (i in 1:n) {
    F.dat[i, ] <- a[i] * phi1 + b[i] * phi2
  }

  #could probably create a list to return more stuff but eh
  out <- list(Fmatrix = as.matrix(F.dat), t = t)
  return(out)
}

#create functional data
F.dat <- prob5()

#FPCA portion
problem5 <- fpca_cust(Ft = F.dat$Fmatrix, t = F.dat$t)




#Problem 6
prob6.dat <- readMat("Datasets/HW1/Problem 6/RegressionDataFile.mat")

#set up data
Fmat <- prob6.dat$f0
t <- as.numeric(prob6.dat$t)
y <- prob6.dat$y0

#plotting the functional data we have
df_fun <- data.frame(
  t = rep(t, each = nrow(Fmat)),
  curve = factor(rep(seq_len(nrow(Fmat)), times = ncol(Fmat))),
  value = as.vector(Fmat)
)

ggplot(df_fun, aes(x = t, y = value, group = curve)) +
  geom_line(alpha = 0.35) +
  labs(title = "Original functional predictors F_i(t)",
       x = "t", y = "F_i(t)") +
  theme_minimal(base_size = 14)
```

```
#basis (B-splines) and design matrix
K <- 5
B  <- bs(t, df = K, intercept = TRUE)
dt <- diff(t)[1]

#for trapezoid rule
wt_obs <- c(dt/2, rep(dt, length(t)-2), dt/2)
#X  <- Fmat %*% (B*dt)

#trying to keep the same notation as Silverman
Phi <- Fmat %*% (B*wt_obs)

#weight matrix
W <- diag(nrow(Phi))

#precompute
PhiT_W <- t(Phi) %*% W
PhiT_W_Phi <- PhiT_W %*% Phi

t_dense <- seq(min(t), max(t), length.out = 200)
B_dense <- bs(t_dense, df = K, intercept = TRUE)
L <- B_dense

#second difference penalty matrix
D2 <- diff(diag(K), differences = 2)
P  <- crossprod(D2)

#function to build continuous penalty
q6.pen <- function(K, t, M){
  t_pen <- seq(t[1], t[2], length.out = M)

  #getting the stepsize
  h <- (t_pen[2] - t_pen[1])

  #weights
  w <- c(h/2, rep(h, M-2), h/2)

  #get B splines and second derivatives of B splines
  B2 <- bSpline(t_pen, df = K, intercept = TRUE, derivs = 2)

  #initialize penalty matrix
  P <- matrix(0, nrow = K, ncol = K)

  #loop
  for (i in 1:K) {
    for (j in i:K) {
      val <- sum(w * B2[, i] * B2[, j])
      P[i, j] <- val
      P[j, i] <- val
    }
  }

  return(P)
}

test <- q6.pen(K=5, t = t, M = 500)

#two different penalty matrices.  One based on second differences.  The other based
#on approximating the integral
# P
# test

lambda_grid <- c(0, 0.5, 1, 5, 50, 100)
```

```
#function to fit and return CIs for one lambda
fit_one_lambda <- function(lambda) {
  A <- A <- PhiT_W_Phi + (lambda*P)
  S <- solve(A, PhiT_W)

  #coefficients and predicted values
  chat <- as.vector(S %*% y)
  yhat <- as.vector(Phi %*% chat)

  #degrees of freedom
  Hdiag  <- rowSums(Phi * t(S))
  df_eff <- sum(Hdiag)

  #estimate of sigma^2
  n <- nrow(Phi)
  sigma2_hat <- sum((y - yhat)^2) / (n-df_eff)

  #computing variance and CIS
  LS <- L %*% S
  beta_var <- sigma2_hat * rowSums(LS*LS)
  beta_hat <- as.vector(L %*% chat)
  beta_se  <- sqrt(beta_var)

  #critical Z value for 95% CI
  z <- qnorm(0.975)

  #create dataframe for outputs
  data.frame(
    t          = t_dense,
    beta       = beta_hat,
    lo         = beta_hat - (z*beta_se),
    hi         = beta_hat + (z*beta_se),
    lambda     = lambda,
    lambda_lab = paste0("Lambda=", lambda),
    df_eff     = df_eff
  )
}

df_all <- do.call(rbind, lapply(lambda_grid, fit_one_lambda))

#plot this
ggplot(df_all, aes(x = t, y = beta)) +
  geom_ribbon(aes(ymin = lo, ymax = hi), alpha = 0.18) +
  geom_line(linewidth = 1) +
  facet_wrap(~ lambda_lab, ncol = 3) +
  labs(
    title = expression(hat(beta)(t) ~ "with 95% pointwise CI for Penalized/Unpenalized 
regression"),
    subtitle = "95% Confidence Intervals Shown",
    x = "t",
    y = expression(hat(beta)(t))
  ) +
  theme_minimal(base_size = 14)
```