R code file for latent factor model for multivariate functional data

Ruonan Li[*] Luo Xiao[†]

March 19, 2023

Introduction

The functional latent factor model (FLFM) for multivariate functional data is fully implemented and evaluated in this document. The evaluation includes an example with simulated data in Section 1 and an example with EEG data in Section 2. It takes about three minutes to compile this RMD file.

In Section 1, we compare FLFM to the multivariate functional mixed model (MFMM) in Li et al. (2022) and MFPCA in Happ and Greven (2018) using a dataset of $n = 100$ subjects, $J = 20$ functional outcomes, and $L = 2$. Section 1.1 provides examples of how to use the `FLFM` function to fit FLFM and sparse FLFM, and the `predict.FLFM` function to predict data for test sets. While the selection of tuning parameters can be time-consuming (about 20 minutes), we commented this part by pound symbol. If you are interest in cross-validation, please remove pound symbols in this part and run the code. Section 1.2 displays visualizations of estimated orthonormal loading coefficient vectors and estimated covariance functions of shared latent factors. In Section 1.3, we measure the estimation accuracy of covariance functions using the relative integrated squared error (RISE), and we draw an additional 100 subjects as a test dataset to evaluate prediction accuracy.

Section 2 demonstrates an example of using FLFM with EEG data. Section 2.1 outlines the application of the `FLFM` function on EEG data. While the selection of L can also be time-consuming (about 20 minutes), we commented this part by pound symbol. Section 2.2 displays visualizations of estimated orthonormal loading coefficient vectors and estimated covariance functions of shared latent factors.

```r
require(Rcpp)
require(RSpectra)
require(refund)
require(face)
require(splines)
require(MASS)
require(MFPCA)
require(mgcv)
require(Matrix)
require(fields)
require(rARPACK)
require(quadprog)
require(glmnet)
require(elasticnet)
require(pracma)
require(ggplot2)
require(gridExtra)
require(reshape2)

# Load functions
```

[*]Department of Statistics, North Carolina State University
[†]Department of Statistics, North Carolina State University

```
source("./R code/GenerateData_L2.R")
source("./R code/GenerateData_L5.R")
source("./R code/FLFM.R")
source("./R code/Validation_L.R")
source("./R code/predict.FLFM.R")
source("./R code/predict.MFPCA.R")
source("./R code/fbps.cov.R") # for smooth covariance matrices
source("./R code/Renew_beta.R") # for renew Beta
source("./R code/beta_init_func.R")
source("./R code/Renew_C.R") # for renew shared covariance fucntions
source("./R code/Renew_G_sigma.R") # for renew outcome-specific terms
source("./R code/Renew_G_gamma_sigma.R") # for renew outcome-specific terms

source("./R code/Validation_rho.R") # for sparse beta
source("./R code/spca_seqADMM.R") # for sparse beta
sourceCpp("./R code/eigendecomp.cpp") # for sparse beta
sourceCpp("./R code/MatrixMtp.cpp") # for sparse beta
```

1. A simulation example

Date are generated from the model $Y_{ij}(t) = \mu_j(t) + \sum_{\ell=1}^{3} \beta_{j\ell} U_{i\ell}(t) + V_{ij}(t) + \epsilon_{ij}(t)$. $L = 2$ and the functional intra-class correlation (FICC) is 2/3 here. The detail of the data from the `GenerateData` function is shown in the paper.

```
Nsubj <- 200
Ntrain <- 100
Ntest <- 100
t <- 101
J <- 20
L_true <- 2 #L for generating data
FICC <- 2/3

set.seed(1)
data <- GenerateData_L2(Nsubj=Nsubj, argval=t, J=J, L=L_true, FICC=FICC, SNR=5)
Y <- data$Y
Y_train <- lapply(1:J, function(x){(Y[[x]])[1:Ntrain,]})
Y_test <- lapply(1:J, function(x){(Y[[x]])[-(1:Ntrain),]})
# True values
beta_true <- as.matrix(data$beta)
C_true <- data$C
para <- data$lambda
G_true <- lapply(1:J, function(x){para[x] * data$G})
sigma2_true <- data$sigma2
argvals <- seq(0, 1, length=t)
```

1.1. Get estimations

Below are examples demonstrating how to use the function `FLFM`. Note that FLFM requires a parameter L, which is selected using 5-fold cross-validation. Due to time limit, we have omitted the cross-validation process in these examples. Once the model is fitted, the function `predict.FLFMA` can be used to make predictions for test data sets.

1.1.1. Get estimations by FLFM

```
# Set parameters
pve <- 0.95 #the proportion of variance explained
knots <- 30 #the number of equidistant knots for all dimensions
p <- 3 #degrees of B-splines
m <- 2 #order of differencing penalty
tol <- 1e-4 #tolerance for iteration
Nmax <- 200  #number of maximal iteration

#######################################################
# Due to time limit, we put pound symbols on this part
# Select the best value L by cross validation.
# seqL = c(1, 2, 3, 4)
# L_result <- Validation_L(Y=Y_train, Nsubj=Ntrain, argvals=argvals, seqL=seqL, kfold=5,
#                          knots=knots, p=p, m=m, tol=10e-4, Nmax=100)
# L <- L_result$L_min
# L_se <- L_result$L_min_se
# cv_diff <- L_result$cv_diff
#######################################################
L_se <- L_true

# Estimate parameters
time1 <- Sys.time()
FLFM_fit <- FLFM(Y=Y_train, Nsubj=Ntrain, argvals=argvals, L=L_se, pve1=pve, pve2=pve,
                 knots=knots, p=p, m=m, tol=tol, Nmax=Nmax, fit=T)
time2 <- Sys.time()
C_hat <-  FLFM_fit$C #estimated Cl
beta_hat <- FLFM_fit$beta #estimated beta
G_hat <- FLFM_fit$G #estimated Gj
sigma2_hat <- FLFM_fit$sigma2 # estimated sigma2

# Predict data in testing set
Pred <- predict.FLFM(object=FLFM_fit, newdata=Y_test, argvals=argvals)
Ypred_FLFM <- Pred$Ypred

time2 - time1
```

```
## Time difference of 7.597572 secs
```

1.1.2. Get estimations by sparse FLFM

This section provides an example of how to fit a sparse FLFM model. The initial parameter values for sparse FLFM are derived from FLFM. It is strongly recommended to set the initial values as the estimates obtained when "sparse=FALSE". Due to time constraints, we have omitted the cross-validation process for tuning the ρ parameter. However, we note that as the true coefficient vectors are not particularly sparse, the ρ value obtained through cross-validation is likely to be small.

```
initial = list(C=FLFM_fit$C, beta=FLFM_fit$beta,
               G=FLFM_fit$G, sigma2=FLFM_fit$sigma2)

##########################################################
# Due to time limit, we put pound symbols on this part
# Select the best value rho by cross validation
# seqrho <- seq(0, 0.5, 0.05) # FICC=2/3
```

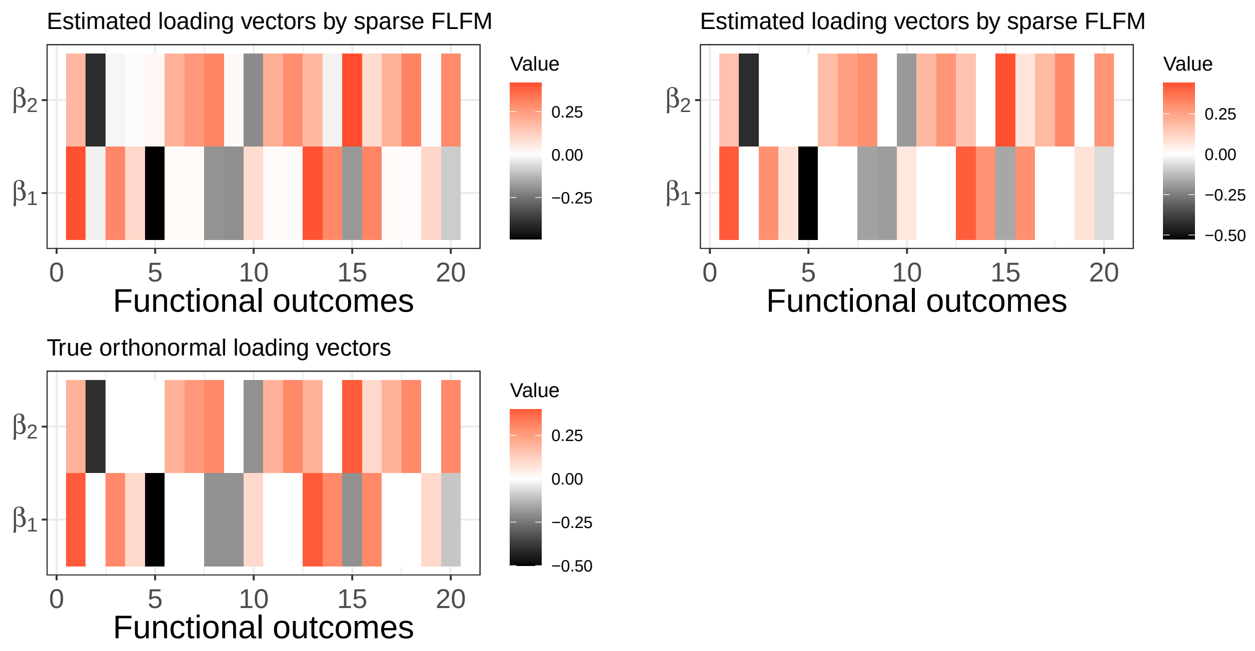
```r
# time2 <- Sys.time()
# rho_result <- Validation_rho(Y=Y_train, Nsubj=Ntrain, argvals=argvals, L=L_se,
#                              seqrho=seqrho, kfold=5, initial=initial, knots=knots,
#                              p=p, m=m, tol=10e-4, Nmax=100)
# rho <- rho_result$rho_min
########################################################
rho <- 0.2

# Estimate parameters
time1 <- Sys.time()
FLFM_sparse <- FLFM(Y=Y_train, Nsubj=Ntrain, argvals=argvals, L=L_se, sparse=TRUE,
                    rho=rho, pve1=pve, pve2=pve, initial=initial, knots=knots,
                    p=p, m=m, tol=tol, Nmax=Nmax, fit=TRUE)
time2 <- Sys.time()
sC_hat <-  FLFM_sparse$C #estimated Cl
sbeta_hat <- FLFM_sparse$beta #estimated beta
sG_hat <- FLFM_sparse$G #estimated Gj
ssigma2_hat <- FLFM_sparse$sigma2 # estimated sigma2

# Predict data in testing set
Pred_sparse <- predict.FLFM(FLFM_sparse, newdata=Y_test, argvals=argvals)
Ypred_sFLFM <- Pred_sparse$Ypred

time2 - time1
```

```
## Time difference of 10.08442 secs
```

1.2. Visualize results

1.2.1. Visualize the estimated orthonormal loading coefficient vectors

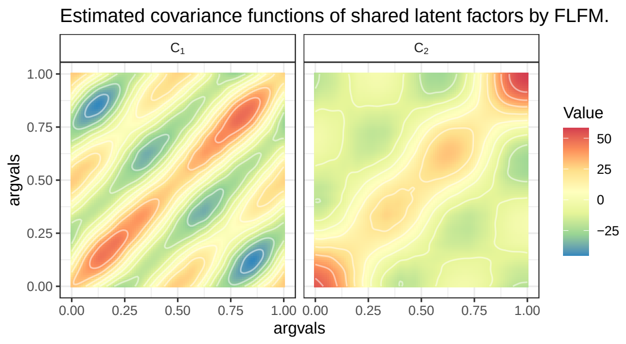
```r
colnames(beta_hat) <- 1:L_true
rownames(beta_hat) <- 1:J
longData <- melt(beta_hat)
longData_sparse <- melt(sbeta_hat)
longData_true <- melt(beta_true)

my.labs=c(expression(beta[1]),expression(beta[2]),expression(beta[3]))


p1 <- ggplot(longData, aes(x=rep(1:J,L_true), y=rep(1:L_true,each=J))) +
  geom_raster(aes(fill=value)) +
  scale_fill_gradient2(low="black", mid="white", high="red", midpoint=0) +
  scale_y_discrete(limits=c("beta1","beta2"), labels=my.labs) +
  labs(x="Functional outcomes", y=" ", fill="Value",
       title="Estimated loading vectors by sparse FLFM") +
  theme_bw() +
  theme(axis.text.x=element_text(size=15, angle=0, vjust=0.3),
        axis.text.y=element_text(size=15),
        axis.title.x = element_text(size = 18),
        axis.title.y = element_text(size = 22),
        axis.title=element_text(size=20))

p2 <- ggplot(longData_sparse, aes(x=rep(1:J,L_true), y=rep(1:L_true,each=J))) +
  geom_raster(aes(fill=value)) +
  scale_fill_gradient2(low="black", mid="white", high="red", midpoint=0) +
  scale_y_discrete(limits=c("beta1","beta2"), labels=my.labs) +
  labs(x="Functional outcomes", y=" ", fill="Value",
       title="Estimated loading vectors by sparse FLFM") +
  theme_bw() +
  theme(axis.text.x=element_text(size=15, angle=0, vjust=0.3),
        axis.text.y=element_text(size=15),
        axis.title.x = element_text(size = 18),
        axis.title.y = element_text(size = 22),
        axis.title=element_text(size=20))

p3 <- ggplot(longData_true, aes(x=rep(1:J,L_true), y=rep(1:L_true,each=J))) +
  geom_raster(aes(fill=value)) +
  scale_fill_gradient2(low="black",mid="white",high="red",midpoint=0) +
  scale_y_discrete(limits=c("beta1","beta2"),labels=my.labs) +
  labs(x="Functional outcomes", y=" ", fill="Value",
       title="True orthonormal loading vectors") +
  theme_bw() +
  theme(strip.background = element_rect(fill = "white"),
        axis.text.x=element_text(size=15, angle=0, vjust=0.3),
        axis.text.y=element_text(size=15),
        axis.title.x = element_text(size = 18),
        axis.title.y = element_text(size = 22),
        axis.title=element_text(size=20))

figure <- grid.arrange(p1, p2, p3, ncol=2, nrow=2)
```

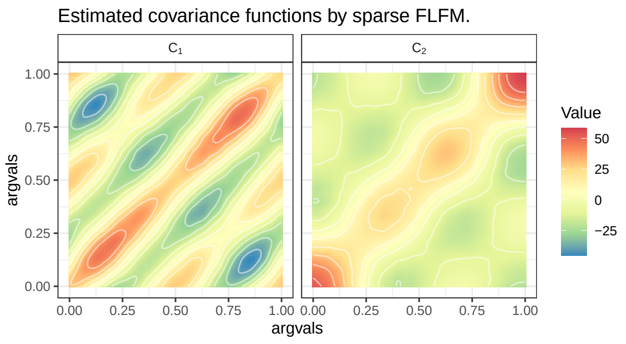


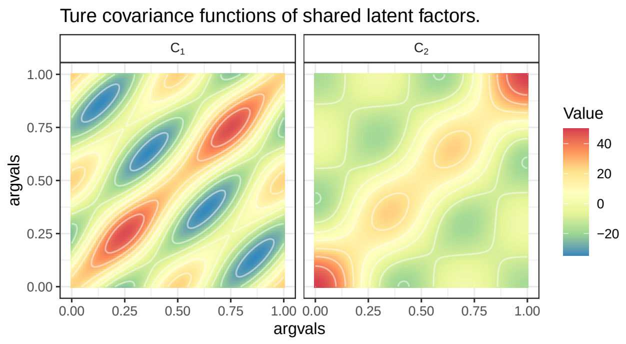
1.2.2. Visualize estimated covariance functions of shared latent factors

```
x = rep(argvals, t)
y = rep(argvals, each=t)
```

```r
df1 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C_hat[[1]])), label=rep("C[1]",t^2))
df2 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C_hat[[2]])), label=rep("C[2]",t^2))
df_est <- rbind(df1,df2)

df4 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(sC_hat[[1]])), label=rep("C[1]",t^2))
df5 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(sC_hat[[2]])), label=rep("C[2]",t^2))
dfs_est <- rbind(df4,df5)

df7 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C_true[[1]])), label=rep("C[1]",t^2))
df8 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C_true[[2]])), label=rep("C[2]",t^2))
df_true <- rbind(df7,df8)

p1 <- ggplot(df_est, aes(x=x, y=y, z=Value, fill=Value)) +
  geom_tile() + coord_equal() +
  geom_contour(color = "white", alpha = 0.5) +
  facet_grid(.~label, labeller = label_parsed) +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  labs(x="argvals", y="argvals",
       title="Estimated covariance functions of shared latent factors by FLFM.") +
  theme_bw() +
  theme(strip.background = element_rect(fill = "white"))

p2 <- ggplot(dfs_est, aes(x=x, y=y, z=Value, fill=Value)) +
  geom_tile() + coord_equal() +
  geom_contour(color = "white", alpha = 0.5) +
  facet_grid(.~label, labeller = label_parsed) +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  labs(x="argvals", y="argvals",
       title="Estimated covariance functions by sparse FLFM.") +
  theme_bw() +
  theme(strip.background = element_rect(fill = "white"))

p3 <- ggplot(df_true, aes(x=x, y=y, z=Value, fill=Value)) +
  geom_tile() + coord_equal() + geom_contour(color = "white", alpha = 0.5) +
  facet_grid(.~label, labeller = label_parsed) +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  labs(x="argvals", y="argvals",
       title="Ture covariance functions of shared latent factors.") +
  theme_bw() +
  theme(strip.background = element_rect(fill = "white"))

figure <- grid.arrange(p1, p2, p3, nrow=2)
```







1.3. Compare FLFM with MFMM and MFPCA

We compare FLFM with MFMM and MFPCA on the same dataset. The relative integrated squared error (RISE) is used to measure the estimation accuracy of covariance functions. For prediction accuracy, we draw 100 additional subjects as the test dataset. Details of RISE and prediction errors are shown in the paper.

1.3.1. Get estimated results by MFMM and MFPCA

Fit the data by MFMM and MFPCA. For MFPCA, we also use a PVE of 0.95. The number of multivariate functional principal components (denoted by M) is fixed at 80.

```
# 1. Get estimated results by MFMM
MFMM_fit <- FLFM(Y=Y_train, Nsubj=Ntrain, argvals=argvals,
                 L=1, pve1=pve, pve2=pve, common_G=T,
                 knots=knots, p=p, m=m, tol=tol, Nmax=Nmax, fit=T)
# Predict data in testing set
Pred <- predict.FLFM(MFMM_fit, newdata=Y_test, argvals=argvals)
Ypred_MFMM <- Pred$Ypred

# 2. Get estimated by MFPCA based on univariate FPCA
sim <- multiFunData(lapply(1:J, function(x){
  funData(argvals=argvals, X = Y_train[[x]])}))
M <- 80
utype <- list(type="uFPCA", pve=pve)
MFPCA_fit <- MFPCA(sim, M=M,
                   uniExpansions=list(utype,utype,utype,utype,utype,utype,utype,
                                      utype,utype,utype,utype,utype,utype,utype,
                                      utype,utype,utype,utype,utype,utype), fit=TRUE)
time2 <- Sys.time()
MFPCA_eigfun <- MFPCA_fit$functions
```

```r
MFPCA_eigval <- MFPCA_fit$values
# Predict data in testing set
Pred_MFPCA <- predict.MFPCA(MFPCA_fit, data=Y_train, newdata=Y_test,
                           argvals=argvals, pve=pve)
Ypred_MFPCA <- Pred_MFPCA$Ypred

time2 - time1
```

```
## Time difference of 49.41657 secs
```

1.3.2. Compute RISE and the prediction error for FLFM, sparse FLFM , MFMM and MFPCA

```r
# 1. For FLFM
num_FLFM <- matrix(0, nrow=J, ncol=J)
den_FLFM <- matrix(0, nrow=J, ncol=J)
for(j1 in 1:J){
  for(j2 in j1:J){
    Cov_true <- Reduce('+', lapply(1:ncol(beta_true), function(x){
      beta_true[j1,x]*beta_true[j2,x]*C_true[[x]] })) #true value
    Cov_hat  <- Reduce('+', lapply(1:L_se, function(x){
      beta_hat[j1,x]*beta_hat[j2,x]*C_hat[[x]] })) #estimated value
    if(j1==j2){
      Cov_true <- Cov_true + G_true[[j1]] #true value
      Cov_hat <- Cov_hat + G_hat[[j1]] #estimated value
    }
    num_FLFM[j1,j2] <- sum((Cov_true - Cov_hat)^2)
    den_FLFM[j1,j2] <- sum(Cov_true^2)
  } #end j2
} #end j1
num_FLFM <- 2*sum(num_FLFM) - sum(diag(num_FLFM))
den_FLFM <- 2*sum(den_FLFM) - sum(diag(den_FLFM))
RISE_FLFM <- num_FLFM/den_FLFM
## Compute the prediction error for test set
Ypred_FLFM <- Pred$Ypred
PE_out_FLFM <- sum(unlist(lapply(1:J, function(x){
  norm(Ypred_FLFM[[x]] - Y_test[[x]],"F")^2})))/(Ntest*J*t)

# 2. For MFMM
num_MFMM <- matrix(0, nrow=J, ncol=J)
den_MFMM <- matrix(0, nrow=J, ncol=J)
for(j1 in 1:J){
  for(j2 in j1:J){
    Cov_true <- Reduce('+', lapply(1:ncol(beta_true), function(x){
      beta_true[j1,x]*beta_true[j2,x]*C_true[[x]] })) #true value
    Cov_hat  <- MFMM_fit$beta[j1,1]*MFMM_fit$beta[j2,1]*MFMM_fit$C[[1]] #estimated value
    if(j1==j2){
      Cov_true <- Cov_true + G_true[[j1]] #true value
      Cov_hat <- Cov_hat + MFMM_fit$gamma[j1]^2*MFMM_fit$G #estimated value
    }
    num_MFMM[j1,j2] <- sum((Cov_true - Cov_hat)^2)
    den_MFMM[j1,j2] <- sum(Cov_true^2)
  } #end j2
} #end j1
num_MFMM <- 2*sum(num_MFMM) - sum(diag(num_MFMM))
den_MFMM <- 2*sum(den_MFMM) - sum(diag(den_MFMM))
RISE_MFMM <- num_FLFM/den_FLFM

## Compute the prediction error for test set
Ypred_MFMM <- Pred$Ypred
PE_out_MFMM <- sum(unlist(lapply(1:J, function(x){
  norm(Ypred_FLFM[[x]] - Y_test[[x]],"F")^2})))/(Ntest*J*t)

# 3. For sparse FLFM
num_sFLFM <- matrix(0, nrow=J, ncol=J)
den_sFLFM <- matrix(0, nrow=J, ncol=J)
for(j1 in 1:J){
  for(j2 in j1:J){
    Cov_true <- Reduce('+', lapply(1:ncol(beta_true), function(x){
      beta_true[j1,x]*beta_true[j2,x]*C_true[[x]] })) #true value
    Cov_hat  <- Reduce('+', lapply(1:L_se, function(x){
      sbeta_hat[j1,x]*sbeta_hat[j2,x]*sC_hat[[x]] })) #estimated value
    if(j1==j2){
      Cov_true <- Cov_true + G_true[[j1]] #true value
      Cov_hat <- Cov_hat + G_hat[[j1]] #estimated value
    }
    num_sFLFM[j1,j2] <- sum((Cov_true - Cov_hat)^2)
    den_sFLFM[j1,j2] <- sum(Cov_true^2)
    } #end j2
  } #end j1
num_sFLFM <- 2*sum(num_sFLFM) - sum(diag(num_sFLFM))
den_sFLFM <- 2*sum(den_sFLFM) - sum(diag(den_sFLFM))
RISE_sFLFM <- num_sFLFM/den_sFLFM
# Compute the prediction error for test set
PE_out_sFLFM <- sum(unlist(lapply(1:J, function(x){
  norm(Ypred_sFLFM[[x]] - Y_test[[x]],"F")^2})))/(Ntest*J*t)

# 4. For MFPCA
num_MFPCA <- matrix(0, nrow=J, ncol=J)
den_MFPCA<- matrix(0, nrow=J, ncol=J)
for(j1 in 1:J){
  for(j2 in 1:J){
    Cov_true <- Reduce('+', lapply(1:ncol(as.matrix(beta_true)), function(x){
      beta_true[j1,x]*beta_true[j2,x]*C_true[[x]] })) #true value
    Cov_hat <- Reduce('+', lapply(1:M, function(x){#estimated value
      MFPCA_eigval[x]*MFPCA_eigfun[[j1]]@X[x,]%*%t(MFPCA_eigfun[[j2]]@X[x,])}))
    if(j1==j2){Cov_true <- Cov_true + G_true[[j1]]} #true value
    num_MFPCA[j1,j2] <- sum((Cov_true - Cov_hat)^2)
    den_MFPCA[j1,j2] <- sum(Cov_true^2)
  } #end j2
} #end j1
num_MFPCA <- sum(num_MFPCA)
den_MFPCA <- sum(den_MFPCA)
RISE_MFPCA <- num_MFPCA/den_MFPCA

## Compute the prediction error for test set
```

```
PE_out_MFPCA <- sum(unlist(lapply(1:J, function(x){
  norm(Ypred_MFPCA[[x]] - Y_test[[x]],"F")^2})))/(Ntest*J*t)

# output
RISE <- c(RISE_FLFM, RISE_sFLFM, RISE_MFMM, RISE_MFPCA)
PE<- c(PE_out_FLFM, PE_out_sFLFM, PE_out_MFMM , PE_out_MFPCA)
result <- data.frame(RISE=RISE, PE=PE)
rownames(result) = c("FLFM", "sFLFM", "MFMM", "MFPCA")
round(result,2)
```

```
##       RISE   PE
## FLFM  0.06 0.89
## sFLFM 0.07 0.92
## MFMM  0.06 0.89
## MFPCA 0.15 0.96
```

2. EEG data

```
load("./data/EEG.RData")
data <- EEG
Nsubj <- length(data)
variables <- unique(data[[1]][,2])
J <- length(variables)
Y <- list()
for(j in 1:J){
  index <- which(data[[1]][,2]==variables[j])
  temp <- lapply(1:Nsubj, function(x){data[[x]][index,4]})
  Y[[j]] <- do.call("rbind", temp)
}
t <- ncol(Y[[1]])
argvals <- seq(0, 1, length=t)
label <- c(rep(1,65),rep(0,44),rep(1,12),0)
# reorder electrodes
ord0 <- c(1,3,9,10,5,7,14,16,12,25,19,23,21,36,28,32,30,34,39,45,41,43,48,52,
          50,58,54,57,55,63,61,27,4,8,11,17,18,26,2,24,20,35,29,15,13,46,38,
          6,40,44,47,53,31,33,59,60,22,56,62,51,49,42,64,37)
alcho0 <- matrix(data[[25]][,4], nc=J, nr=t, byrow=F)
contr0 <- matrix(data[[82]][,4], nc=J, nr=t, byrow=F)
ord <- order(ord0)
```

2.1. Get estimations by FLFM

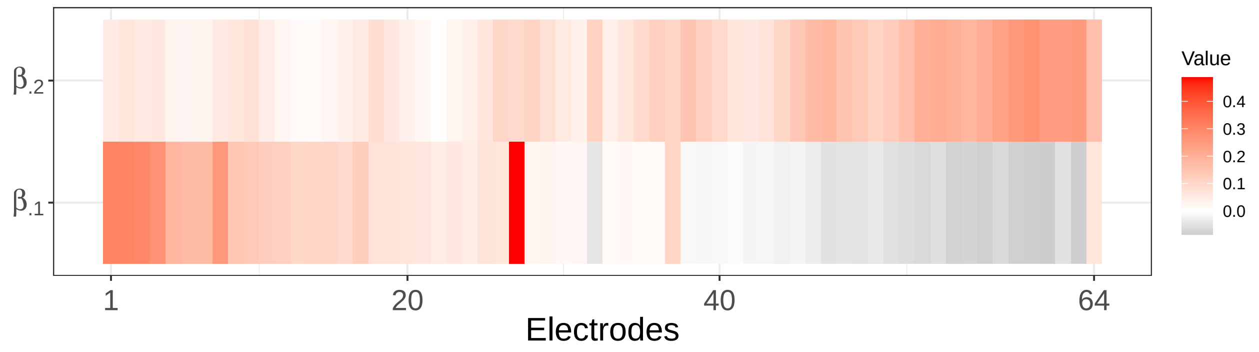
```
knots <- 30 #list of two vectors of knots or number of equidistant knots for all dimensions
p <- 3 #degrees of B-splines
m <- 2 #order of differencing penalty
tol <- 5*1e-3 #tolerance for iteration
Nmax <- 200  #number of maximal iteration

# 1. Select the best value L by cross validation
# time1 <- Sys.time()
# seqL = c(1,2,3,4)
# L_result <- Validation_L(Y=Y, Nsubj=Nsubj, argvals=argvals, seqL=seqL, sparse=FALSE, rho=0,
#                          kfold=5, knots=knots, p=p, m=m, tol=tol, Nmax=100)
```

```r
# L <- L_result$L_min_se
# time2 <- Sys.time()
# time2 - time1
# 2. Estimate parameters
L <- 2
LFM_fit <- FLFM(Y=Y, Nsubj=Nsubj, argvals=argvals, L=L, sparse=FALSE, rho=0.1,
                pve1=0.95, pve2=0.90, knots=knots, p=p, m=m, tol=tol, Nmax=Nmax, fit=T)

# LFM_score <- t(rbind(LFM_fit$xi, LFM_fit$zeta))
# pred_LFM <- c()
# for(i in 1:Nsubj){
#   Y_train <- LFM_score[-i,]
#   label_train <- label[-i]
#   Y_test <- LFM_score[i,]
#   data_train <- data.frame(group=label_train, data=as.matrix(Y_train))
#   data_test <- data.frame(data=t(as.matrix(Y_test)))
#
#   # Fit model
#   cv.lasso <- cv.glmnet(x=Y_train, y=label_train, alpha=1, family="binomial", nfolds=nrow(Y_train))
#   mylogit <- glmnet(x=Y_train, y=label_train, family = "binomial", alpha=1, lambda=cv.lasso$lambda.mi
#   fitted.results <- predict(mylogit,newx=t(as.matrix(Y_test)), type='response')
#   pred_LFM <- c(pred_LFM, fitted.results)
#   print(i)
# }
# label_pre <- (pred_LFM>0.5)
# # Classification accuracy
# LFM_acc <- sum((label_pre-label)==0)/Nsubj
# print("Classification ccuracy for FLFM")
# LFM_acc
```

2.2. Visualize results for the EEG data

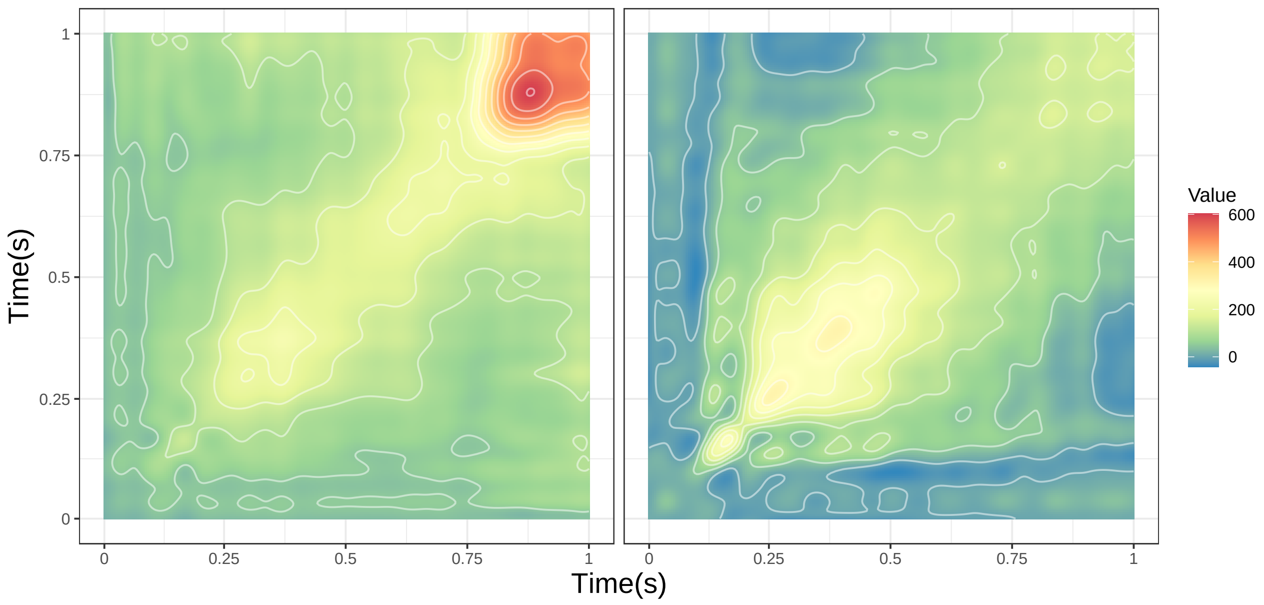
2.2.1. Visualize the estimated orthonormal loading coefficient vectors

```r
beta <- LFM_fit$beta
beta <- beta[ord,]
colnames(beta) <- 1:2
rownames(beta) <- 1:64
longData <- melt(beta)

my.labs=c(expression(beta[.][1]),expression(beta[.][2]))
ggplot(longData, aes(x=rep(1:64,2), y=rep(1:2,each=64))) +
  geom_raster(aes(fill=value)) +
  scale_fill_gradient2(low="black",mid="white",high="red",midpoint=0) +
  labs(x="Electrodes", y=" ", title=" ", fill="Value") +
  scale_y_discrete(limits=c("beta1","beta2"), labels=my.labs) +
  scale_x_continuous(breaks = c(1, 20, 40, 64)) +
  theme_bw() +
  theme(axis.text.x=element_text(size=16, angle=0, vjust=0.3),
        axis.text.y=element_text(size=16),
        axis.title.x = element_text(size = 18),
        axis.title.y = element_text(size = 22),
        axis.title=element_text(size=20))
```



2.2.2. Visualize estimated covariance functions of shared latent factors

```r
C <- LFM_fit$C
C1 <- C[[1]]
C2 <- C[[2]]

timep = 1:256
x = rep(timep, 256)
y = sort(rep(timep, 256))
df1 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C1)), label=rep("C1",256*256))
df2 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C2)), label=rep("C2",256*256))
df <- rbind(df1,df2)

ggplot(df, aes(x=x, y=y, z=Value, fill=Value)) +
  geom_tile() + coord_equal() +
  geom_contour(color = "white", alpha = 0.5) +
  facet_grid(.~label) +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  scale_x_continuous(breaks=c(1,64,128,192,256), labels=c(0, 0.25, 0.5, 0.75, 1)) +
  scale_y_continuous(breaks=c(1,64,128,192,256), labels=c(0, 0.25, 0.5, 0.75, 1)) +
  xlab("Time(s)") + ylab("Time(s)") +
  theme_bw() +
  theme(strip.text.x=element_blank(),
        axis.title.x = element_text(size = 16),
        axis.title.y = element_text(size = 16))
```



Time(s)
1
0.75
0.5
0.25
0
0
0.25
0.5
0.75
1
Time(s)
Value
600
400
200
0