R code file for latent factor model for multivariate functional data

Ruonan Li* Luo Xiao†

April 13, 2022

Introduction

In this document, a complete implementation of the functional latent factor model (FLFM) for multivariate functional data is evaluated. Moreover, comparisons with MFPCA proposed in Happ and Greven (2018) are evaluated using a dataset with $n = 100$ subjects, $J = 20$ functional outcomes and $L = 3$. It takes about 2 minutes to compile this RMD file. The document is organized as follows.

Section 1 gives the code to generate datasets (function `GenerateData`) which are used to evaluate FLFM. The detail of the data from the `GenerateData` function is shown in the paper.

Section 2 shows examples of how to use the function `FLFM` to fit FLFM and sparse FLFM. With the fitted model, users can use the function `predict.FLFM` to predict data for test sets. Since selections of tunning parameters takes a long time (about 20 minutes), we commented this part by pound symbol. If you are interest in cross-validation, please delete pound symbols in this part and run the code.

Section 3 visualizes outputs, including the estimated orthonormal loading coefficient vectors and the estimated covariance functions of shared latent factors.

We compare FLFM with MFPCA on the same dataset in Section 4. To measure the estimation accuracy of covariance functions, we use the relative integrated squared error (RISE). For prediction accuracy, we draw 100 additional subjects as the test dataset.

```
require(Rcpp)
require(RSpectra)
require(refund)
require(refund)
require(face)
require(splines)
require(MASS)
require(MFPCA)
require(mgcv)
require(Matrix)
require(fields)
require(rARPACK)
require(quadprog)
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.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/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. Generate data

Date are generated from the model $Y_{ij}(t) = \mu_j(t) + \sum_{\ell=1}^{3} \beta_{j\ell} U_{i\ell}(t) + V_{ij}(t) + \epsilon_{ij}(t)$. $L = 3$ 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 <- 3 #L for generating data
FICC <- 2/3

set.seed(1)
data <- GenerateData(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)
```

2. Get estimations by FLFM

This section shows examples of how to use the function `FLFM`. The function `FLFM` requires a parameter L, which is selected by 5-fold cross-validation. Due to time limit, we put pound symbols on the cross-validation part. With the fitted model, users can use the function `predict.FLFMA` to predict data for test sets.

2.1 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(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 12.28985 secs
```

2.2 Get estimations by sparse FLFM

This part gives an example of how to fit the sparse FLFM model. The initial values of parameters for sparse FLFM are from FLFM. It highly recommends to set initial values as the estimation when "sparse=FALSE". Due to time limit, we put pound symbols on the cross-validation for tunning paramter ρ. Since the true coefficient vectors don't have much sparsity, *rho* given by cross-validation is 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
# 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
```

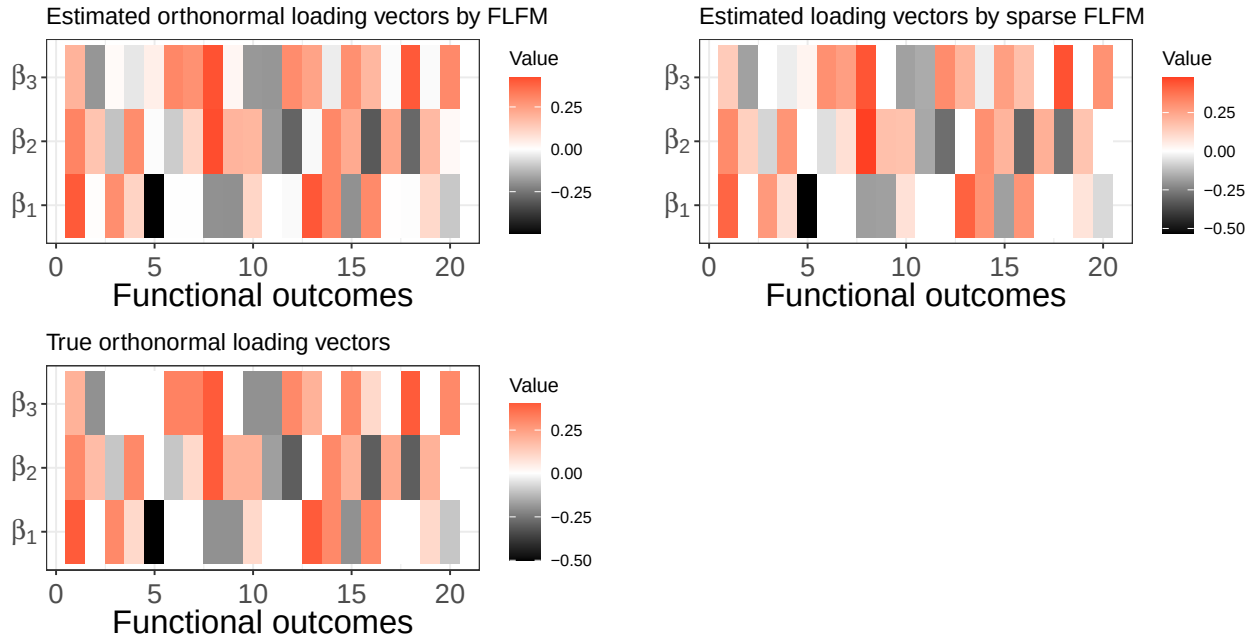
```
## Time difference of 16.53919 secs
```

3. Visualize results

3.1 Visualize the estimated orthonormal loading coefficient vectors.

```
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) +
  labs(x="Functional outcomes", y=" ",
       title="Estimated orthonormal loading vectors by FLFM") +
  labs(fill="Value") + theme_bw() +
  scale_y_discrete(limits=c("beta1","beta2","beta3"),labels=my.labs) +
  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) +
  labs(x="Functional outcomes", y=" ",
       title="Estimated loading vectors by sparse FLFM") +
  labs(fill="Value") + theme_bw() +
  scale_y_discrete(limits=c("beta1","beta2","beta3"),labels=my.labs) +
```

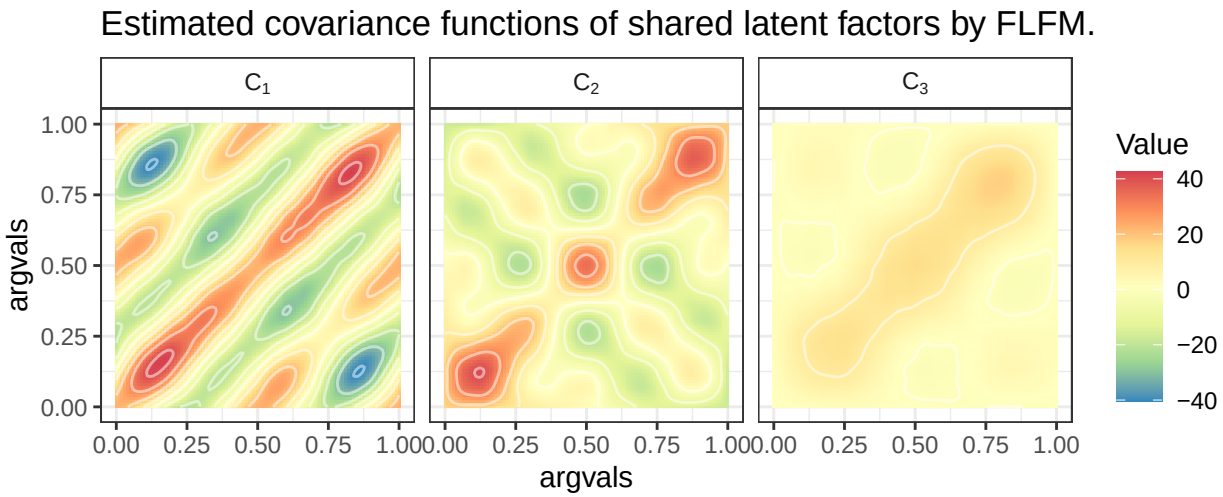
```
  theme(axis.text.x=element_text(size=15, angle=0, vjust=0.3),
        axis.text.y=element_text(size=15),
        axis.title.x = element_text(size = 18),
        axis.title.y = element_text(size = 22),
        axis.title=element_text(size=20))

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) +
  labs(x="Functional outcomes", y=" ",
       title="True orthonormal loading vectors") +
  labs(fill="Value") + theme_bw() +
  scale_y_discrete(limits=c("beta1","beta2","beta3"),labels=my.labs) +
  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)
```

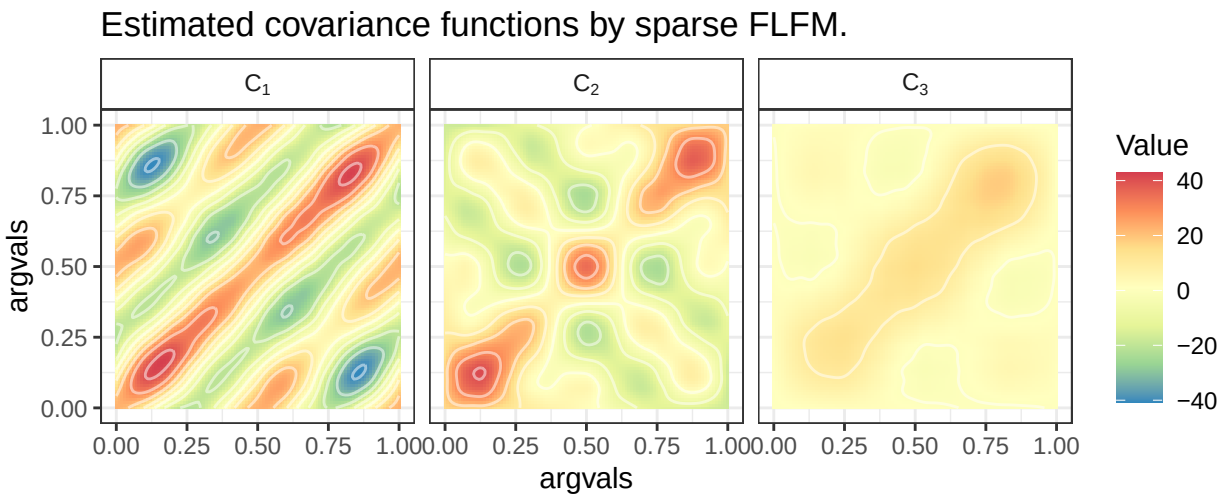


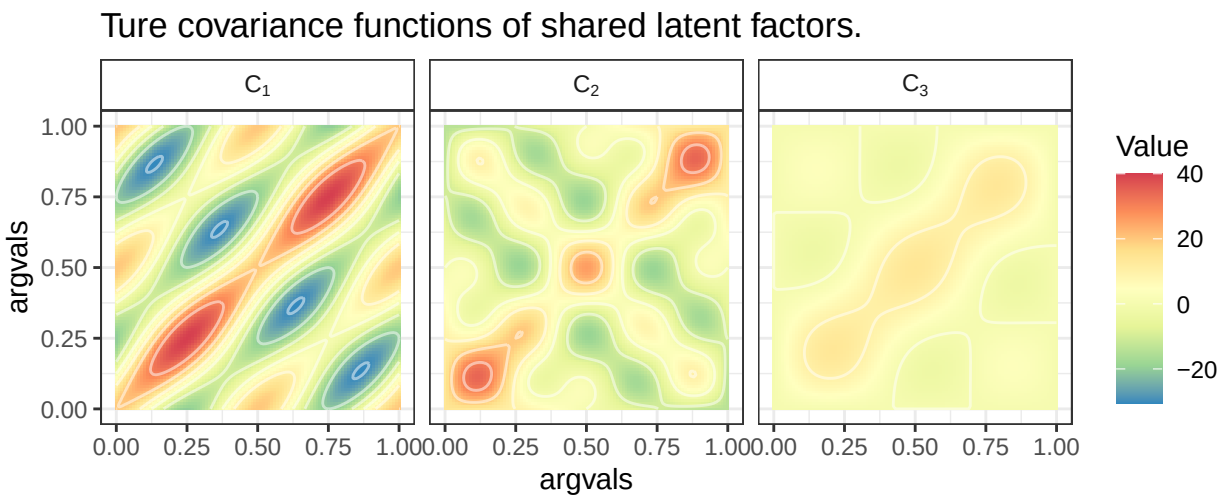
3.2 Visualize estimated covariance functions of shared latent factors.

```
x = rep(argvals, t)
y = rep(argvals, each=t)
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))
df3 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C_hat[[3]])), label=rep("C[3]",t^2))
df_est <- rbind(df1,df2,df3)

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))
df6 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(sC_hat[[3]])), label=rep("C[3]",t^2))
dfs_est <- rbind(df4,df5,df6)

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))
df9 <- data.frame(x=x, y=y, Value=as.vector(as.matrix(C_true[[3]])), label=rep("C[3]",t^2))
df_true <- rbind(df7,df8,df9)


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) + theme_bw() +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  theme(strip.background = element_rect(fill = "white")) +
  labs(x="argvals", y="argvals",
       title="Estimated covariance functions of shared latent factors by FLFM.")
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) + theme_bw() +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  theme(strip.background = element_rect(fill = "white")) +
  labs(x="argvals", y="argvals",
       title="Estimated covariance functions by sparse FLFM.")
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) + theme_bw() +
  scale_fill_distiller(palette="Spectral", na.value="white") +
  theme(strip.background = element_rect(fill = "white")) +
  labs(x="argvals", y="argvals",
       title="Ture covariance functions of shared latent factors.")

figure <- grid.arrange(p1, p2, p3, nrow=3)
```







4. Compare FLFM with MFPCA

We compare FLFM with 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.

4.1 Get estimated results by MFPCA.

Fit the data by 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.

```
sim <- multiFunData(lapply(1:J, function(x){
  funData(argvals=argvals, X = Y_train[[x]])}))

# MFPCA based on univariate FPCA
M <- 80
utype <- list(type="uFPCA", pve=pve)
time1 <- Sys.time()
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$functions
MFPCA_eigval <- MFPCA_fit$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 27.23139 secs
```

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

```
# For FLFM
## Compute RISE
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)

# For sparse FLFM
# Compute RISE
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)

# For MFPCA
## Compute RISE
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)


RISE <- c(RISE_FLFM, RISE_sFLFM, RISE_MFPCA)
```

```r
PE<- c(PE_out_FLFM, PE_out_sFLFM, PE_out_MFPCA)
result <- data.frame(RISE=RISE, PE=PE)
rownames(result) = c("FLFM","sFLFM", "MFPCA")
round(result,2)
```

```
##       RISE   PE
## FLFM  0.09 0.94
## sFLFM 0.10 0.94
## MFPCA 0.24 1.05
```