The modified mpca.sc function in refund

Ruonan Li

Introduction

The report is organized as following. We first discussthe part of the `mfpca.sc` function which costs long computation time in Section 1. We describe details of how to speed up the `mfpca.sc` function in Section 2. Section 3 guarantees the numerical accuracy of the updated function by simulation. In Section 4, we introduce a new parameter `weight` to select the way of computing smaple convariance.

The original `mfpca.sc` function requires data arranged by ID. If we restructure data, `mfpca.sc` doesn't work. The updated function fixed this problem. If we restructure datasets by shuffling rows, the updated function would give the same results of estimation. We show it in Section 3.3.

1 The part of the `mfpca.sc` function which makes it cost long computation time.

I checked the `mfpca.sc` function in the R package `refund`. There are two parts that cost much time. We assume that the subject i have n_i visits and T_{ij} observations per curve, so we observe $M_i = \sum_{j=1}^{n_i} T_{ij}$ points for subject i. Denote D by the full number of grid points in a curve. Thus, the irregular T_{ij} points are sampled from D points.

Part 1: The part to smooth sample covariances.

The `mfpca.sc` function applies the `gam` function to smooth sample covariances. We need to smooth two sample covariances. One is the total covariance and the other is the within covariance. A great choice is to use the sandwich smoother in Xiao et al. (2013) to smooth the sample covariance. Since the covariance matrix is symmetric, we only need to select one tuning parameter λ. It contributes to speed up. The implementation of the sandwich smoother is in the `fbps.cov` function.

Part 2: The part to get principal scores by $\begin{pmatrix}\mathbf{A}_i\\ \mathbf{B}_i\end{pmatrix}\boldsymbol{\Sigma}_{\boldsymbol{i}}^{-1}(\mathbf{Y}_i - \boldsymbol{\mu} - \boldsymbol{\eta}_j)$.

$\boldsymbol{\Sigma}_{\boldsymbol{i}} = \text{cov}(\mathbf{Y}_i)$ is a $M_i \times M_i$ matrix. If the number of observed points for subjec i, i.e. $M_i = \sum_{j=1}^{n_i} T_{ij}$, is large, it takes long time to get the inverse of $\boldsymbol{\Sigma}_{\boldsymbol{i}}$. Moreover, it may fail to get the inverse sometimes. Note that the `mfpca.sc` function does not separate regularly sampled functional data and irregularly sampled functional data. We need to compute $\boldsymbol{\Sigma}_{\boldsymbol{i}}^{-1}$ for each i. It requires $O(\sum_{i=1}^{n} M_i^3)$ flops to obation principal scores. I tried $D = 1000$ regularly sampled functional data. It took more than 10 minutes to compute one $\boldsymbol{\Sigma}_{\boldsymbol{i}}^{-1}$ and we failed to get inverse for some i. Therefore, the `mfpca.sc` function does not work for data with $D = 1000$.

2 The way to speed up the `mfpca.sc` function.

I speed up the `mfpca.sc` function by optimizing the above two parts. The other parts of `mfpca.sc` are remained. I call the new function as `mfpca.sc2`. It also allows for different number of visits per subject. The `mfpca.sc2` function seperates regularly sampled functoinal data and irregularly sampled functional data. Thus, I add a new input parameter `design`, default as `irregular`.

Part 1: The way of smoothing sample covariances.

For regularly spaced functional data, we apply the sandwich smoother to smooth sample covariances. For irregularly spaced fucntional data, we still apply the `gam` function as what the `mfpca.sc` function does.

Part 2: The estimate of principal scores.

Once the fixed functional effects $\mu(t)$, $\eta_j(t)$, the eigenvalues $\lambda_k^{(1)}$, $\lambda_l^{(2)}$ and the eigenfunctions $\phi_k(t)$, $\psi_l(t)$ are estimated, the MFPCA model can be re-writen as a linear mixed effects model

$$\tilde{Y}_{ij}(t_{ijs}) = \sum_{k=1}^{N_1} \xi_{ik}\phi_k(t_{ijs}) + \sum_{l=1}^{N_2} \zeta_{ijl}\psi_l(t_{ijs}) + \epsilon_{ijs} \ ,$$
$$\xi_{ik} \sim N\{0, \lambda_k^{(1)}\}, \zeta_{ijl} \sim N\{0, \lambda_l^{(2)}\}, \epsilon_{ijs} \sim N(0, \sigma^2) \ .$$

Let $\tilde{\mathbf{Y}}_{ij} = (\tilde{Y}_{ij1}, \cdots, \tilde{Y}_{ijT_{ij}})^T$, $\boldsymbol{\xi}_i = (\xi_{i1}, \cdots, \xi_{iN_1})^T$ and $\boldsymbol{\zeta}_{ij} = (\zeta_{ij1}, \cdots, \zeta_{ijN_2})^T$. Let $\boldsymbol{\phi}_{k,ij} = (\phi_k(t_{ij1}), \cdots, \phi_k(t_{ijT_{ij}}))^T$, $\boldsymbol{\psi}_{l,ij} = (\psi_l(t_{ij1}), \cdots, \psi_l(t_{ijT_{ij}}))^T$, $\boldsymbol{\Phi}_{ij} = [\boldsymbol{\phi}_{1,ij}, \cdots, \boldsymbol{\phi}_{N_1,ij}] \in \mathbb{R}^{T_{ij}\times N_1}$ and $\boldsymbol{\Psi}_{ij} = [\boldsymbol{\psi}_{1,ij}, \cdots, \boldsymbol{\psi}_{N_2,ij}] \in \mathbb{R}^{T_{ij}\times N_2}$. Let $\boldsymbol{\epsilon}_{ij} = (\epsilon_{ij1}, \cdots, \epsilon_{ijT_{ij}})^T$. Then

$$\tilde{\mathbf{Y}}_{ij} = \boldsymbol{\Phi}_{ij}\boldsymbol{\xi}_i + \boldsymbol{\Psi}_{ij}\boldsymbol{\zeta}_{ij} + \boldsymbol{\epsilon}_{ij} \ .$$

Let $\boldsymbol{\Phi}_i = [\boldsymbol{\Phi}_{i1}^T, \cdots, \boldsymbol{\Phi}_{in_i}^T]^T \in \mathbb{R}^{M_i\times N_1}$ and $\boldsymbol{\Psi}_i = \text{blockdiag}(\boldsymbol{\Psi}_{i1}, \cdots, \boldsymbol{\Psi}_{in_i}) \in \mathbb{R}^{M_i\times(n_iN_2)}$. We denote the second score vector by $\boldsymbol{\zeta}_i = (\boldsymbol{\zeta}_{i1}^T, \cdots, \boldsymbol{\zeta}_{in_i}^T)^T \in \mathbb{R}^{n_iN_2}$. Note that the score $\boldsymbol{\xi}_i$ has a diagonal covariance matrix $\boldsymbol{\Lambda}_1 = \text{diag}(\lambda_1^{(1)}, \cdots, \lambda_{N_1}^{(1)})$ and $\boldsymbol{\zeta}_i$ has a diagonal covariance matrix $\boldsymbol{\Lambda}_2 = \mathbf{I}_{n_i} \otimes \text{diag}(\lambda_1^{(2)}, \cdots, \lambda_{N_2}^{(2)})$. Let $\tilde{\mathbf{Y}}_i = (\tilde{\mathbf{Y}}_{i1}^T, \cdots, \tilde{\mathbf{Y}}_{in_i}^T)^T \in \mathbb{R}^{M_i}$ and $\boldsymbol{\epsilon}_i = (\boldsymbol{\epsilon}_{i1}^T, \cdots, \boldsymbol{\epsilon}_{in_i}^T)^T \in \mathbb{R}^{M_i}$. Note that $\boldsymbol{\epsilon}_i$ has a diagonal covariance matrix $\sigma^2\mathbf{I}_{M_i}$. Finally, we obtain the mixed model representation as

$$\tilde{\mathbf{Y}}_i = \boldsymbol{\Phi}_i\boldsymbol{\xi}_i + \boldsymbol{\Psi}_i\boldsymbol{\zeta}_i + \boldsymbol{\epsilon}_i \ ,$$
$$\begin{pmatrix}\boldsymbol{\xi}_i\\ \boldsymbol{\zeta}_i\\ \boldsymbol{\epsilon}_i\end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix}\mathbf{0}_{N_1}\\ \mathbf{0}_{n_iN_2}\\ \mathbf{0}_{M_i}\end{pmatrix}, \begin{pmatrix}\boldsymbol{\Lambda}_1 & 0 & 0\\ 0 & \boldsymbol{\Lambda}_2 & 0\\ 0 & 0 & \sigma^2\mathbf{I}_{M_i}\end{pmatrix}\right] \ .$$

The MLE for the random effects is

$$\begin{pmatrix}\hat{\boldsymbol{\xi}}_i\\ \hat{\boldsymbol{\zeta}}_i\end{pmatrix} = \begin{pmatrix}\boldsymbol{\Lambda}_1\boldsymbol{\Phi}_i^T\\ \boldsymbol{\Lambda}_2\boldsymbol{\Psi}_i^T\end{pmatrix}(\boldsymbol{\Phi}_i\boldsymbol{\Lambda}_1\boldsymbol{\Phi}_i^T + \boldsymbol{\Psi}_i\boldsymbol{\Lambda}_2\boldsymbol{\Psi}_i^T + \sigma^2\mathbf{I}_{M_i})^{-1}\,\tilde{\mathbf{Y}}_i \ , \tag{1}$$

where $\text{cov}(\tilde{\mathbf{Y}}_i) = (\boldsymbol{\Phi}_i\boldsymbol{\Lambda}_1\boldsymbol{\Phi}_i^T + \boldsymbol{\Psi}_i\boldsymbol{\Lambda}_2\boldsymbol{\Psi}_i^T + \sigma^2\mathbf{I}_{M_i})$. The `mfpca.sc` function applies the equation (1) to compute principal scores. As $\text{cov}(\tilde{\mathbf{Y}}_i) \in \mathcal{R}^{M_i\times M_i}$ can be of huge dimensions, its inverse requires $O(M_i^3)$ flops. Henderson (1950) presented the mixed model equations (MME) to estimate random effects, without the need for computing the inverse of $\text{cov}(\tilde{\mathbf{Y}}_i)$. By MME, we have

$$\begin{pmatrix}\boldsymbol{\Phi}_i^T\boldsymbol{\Phi}_i/\sigma^2 + \boldsymbol{\Lambda}_1^{-1} & \boldsymbol{\Phi}_i^T\boldsymbol{\Psi}_i/\sigma^2\\ \boldsymbol{\Psi}_i^T\boldsymbol{\Phi}_i/\sigma^2 & \boldsymbol{\Psi}_i^T\boldsymbol{\Psi}_i/\sigma^2 + \boldsymbol{\Lambda}_2^{-1}\end{pmatrix}\begin{pmatrix}\boldsymbol{\xi}_i\\ \boldsymbol{\zeta}_i\end{pmatrix} = \begin{pmatrix}\boldsymbol{\Phi}_i^T/\sigma^2\\ \boldsymbol{\Psi}_i^T/\sigma^2\end{pmatrix}\tilde{\mathbf{Y}}_i \ , \tag{2}$$

where the dimension of $\begin{pmatrix}\boldsymbol{\Phi}_i^T\boldsymbol{\Phi}_i/\sigma^2 + \boldsymbol{\Lambda}_1^{-1} & \boldsymbol{\Phi}_i^T\boldsymbol{\Psi}_i/\sigma^2\\ \boldsymbol{\Psi}_i^T\boldsymbol{\Phi}_i/\sigma^2 & \boldsymbol{\Psi}_i^T\boldsymbol{\Psi}_i/\sigma^2 + \boldsymbol{\Lambda}_2^{-1}\end{pmatrix} \in \mathcal{R}^{(N_1+n_iN_2)\times(N_1+n_iN_2)}$ is much smallers than $\text{cov}(\tilde{\mathbf{Y}}_i)$. Moreover, its inverse can be computed by block. Therefore, we can save huge time to compute principal scores by MME, especially when M_i is large. The `mfpca.sc2` function estimates principal scores by MME. If the estimation of $\sigma^2 = 0$, then we have

$$\begin{pmatrix}\boldsymbol{\Phi}_i^T\boldsymbol{\Phi}_i & \boldsymbol{\Phi}_i^T\boldsymbol{\Psi}_i\\ \boldsymbol{\Psi}_i^T\boldsymbol{\Phi}_i & \boldsymbol{\Psi}_i^T\boldsymbol{\Psi}_i\end{pmatrix}\begin{pmatrix}\boldsymbol{\xi}_i\\ \boldsymbol{\zeta}_i\end{pmatrix} = \begin{pmatrix}\boldsymbol{\Phi}_i^T\\ \boldsymbol{\Psi}_i^T\end{pmatrix}\tilde{\mathbf{Y}}_i \ .$$

3 Compare the computation time and estimate accuracy of `mfpca.sc` and `mfpca.sc2`.

3.1 Simulation

For regular functional data, I used the data generated from a true model in Di et al. (2009). For irregular functional data, curves were sampled on a sparse set of grid points. I considered $n = 100$ subjects, $n_i = 3$ visits per subject and

$D = 100, 500, 1000$ points per curve. The magnitude of noise $\sigma = 0, 1$. For irregular senarios, the number of observed points per function, i.e. T_{ij}, is $0.1 \times D$ or $0.5 \times D$.

I simulated 100 data sets for each scenario and compared `mfpca.sc` and `mfpca.sc2` by computation time, mean integraed square errors (MISE) for eigenfunctions and observations. Results are shown in Table 1. For regularly sampled functional data, `mfpca.sc2` is much more computationally efficient. One reason is a more efficient way of smoothing and another reason is a better estimation of principal scores by MME. For irregular functional data, if there is not too many missings, `mfpca.sc2` is still much more efficient than `mfpca.sc`. If there is only several observations per curve, the computation times of `mfpca.sc` and `mfpca.sc2` are comparable. For the regular case with $D = 1000$, I only show results of `mfpca.sc2` since `mfpca.sc` does not work for some simulated data sets.

Table 1: The average MISE for eigenfunctions and $\mathbf{Y}$ across 100 simulations.

	mfpca.sc				mfpca.sc2			
model	time(mins)	MISE($\mathbf{Y}$)	MISE(Φ)	MISE(Ψ)	time(mins)	MISE($\mathbf{Y}$)	MISE(Φ)	MISE(Ψ)
	$D = 100$ & $\sigma = 0$							
regular	**0.31**	0.0003	0.0817	0.0199	**0.01**	0.0000	0.0816	0.0197
irregular 0.1	**0.04**	0.0053	0.2148	0.1841	**0.03**	0.0072	0.2153	0.1820
irregular 0.5	**0.08**	0.0042	0.0925	0.0303	**0.03**	0.0046	0.0926	0.0303
	$D = 100$ & $\sigma = 1$							
regular	**0.35**	0.9529	0.0858	0.0227	**0.01**	0.9506	0.0865	0.0232
irregular 0.1	**0.04**	0.6092	0.2693	0.2591	**0.03**	0.6100	0.2693	0.2580
irregular 0.5	**0.08**	0.9059	0.1098	0.0365	**0.04**	0.9061	0.1100	0.0365
	$D = 500$ & $\sigma = 0$							
regular	**42.17**	0.0002	0.0815	0.0193	**0.07**	0.0000	0.0813	0.0192
irregular 0.1	**0.83**	0.0060	0.0966	0.0387	**0.77**	0.0060	0.0966	0.0387
irregular 0.5	**5.14**	0.0012	0.0837	0.0221	**0.68**	0.0012	0.0837	0.0221
	$D = 500$ & $\sigma = 1$							
regular	**55.12**	0.9904	0.0718	0.0178	**0.07**	0.9858	0.0794	0.0193
irregular 0.1	**0.84**	0.9017	0.0992	0.0462	**0.77**	0.9017	0.0992	0.0462
irregular 0.5	**6.04**	0.9815	0.0845	0.0220	**0.82**	0.9815	0.0845	0.0220
	$D = 1000$ & $\sigma = 0$							
regular					**0.33**	0.0000	0.0748	0.0201
irregular 0.1	**4.57**	0.0042	0.0913	0.0289	**4.14**	0.0042	0.0913	0.0289
irregular 0.5	**46.65**	0.0007	0.0749	0.0214	**3.55**	0.0007	0.0749	0.0214
	$D = 1000$ & $\sigma = 1$							
regular					**0.33**	0.9813	0.0873	0.0212
irregular 0.1	**4.61**	0.9531	0.0904	0.0325	**4.11**	0.9531	0.0904	0.0325
irregular 0.5	**59.72**	0.9908	0.0836	0.0205	**4.54**	0.9908	0.0836	0.0205

3.1.1 This exmaple applies regularly sampled functional data with $D = 100$ and $\sigma = 0$.

```r
setwd("~/Users/new_mfpca/codes")
rm(list=ls())
## Check for packages needed to run analyses
pckgs <- c("refund","MASS","splines","simex","mgcv")
sapply(pckgs, function(x) if(!require(x,character.only=TRUE,quietly=TRUE)) {
  install.packages(x)
  require(x, character.only=TRUE)
})
```

```
## $refund
## NULL
##
## $MASS
## NULL
##
## $splines
## NULL
##
## $simex
## NULL
##
## $mgcv
## NULL
```

```r
rm(list=c("pckgs"))
source("GeneData.R")
source("fbps.cov.R")
source("mfpca.sc2.R")


set.seed(1)
Nsub=100;J=3;D=100;K1=4;K2=4;design="regular";weight="obs"
data <- GeneData(M=Nsub, J=J, N=D, design=design, level=0.1, sigma=0)
Y <- data$Y

# True values
evalues_true <- data$evalues
eigenf_true <- data$eigenfunctions
# Parameters
id <- rep(1:Nsub, each=J)
twoway <- TRUE


#1. mfpca.cs in refund
s1 <- Sys.time()
fit1 <-  mfpca.sc(Y=Y, id=id, twoway=twoway)
s2 <- Sys.time()
time1 <- difftime(s2, s1, units = "mins")
# MISE of observations
diff1=0;num=0
for(i in 1:nrow(Y)){
  idx = which(!is.na(Y[i, ]))
  num = num + length(idx)
  diff1 = diff1 + sum(abs(fit1$Yhat[i,idx]-Y[i,idx])^2)
}
MISE1_Y <- diff1/num
# MISE of eigenfucntions
MISE1_eigen1 <- sum(unlist(lapply(1:K1, function(x){
  min(sum((eigenf_true[[1]][,x]-fit1$efunctions[[1]][,x])^2),
      sum((eigenf_true[[1]][,x]+fit1$efunctions[[1]][,x])^2))})))/(K1*D)
MISE1_eigen2 <- sum(unlist(lapply(1:K2, function(x){
  min(sum((eigenf_true[[2]][,x]-fit1$efunctions[[2]][,x])^2),
      sum((eigenf_true[[2]][,x]+fit1$efunctions[[2]][,x])^2))})))/(K2*D)
result1 <- round(c(as.numeric(time1), MISE1_Y, MISE1_eigen1, MISE1_eigen2),4)


#2. the modified mfpca function: mfpca.cs2
s1 <- Sys.time()
fit2 <-  mfpca.sc2(Y=Y, id=id, twoway=twoway, design=design, weight=weight)
s2 <- Sys.time()
time2 <- difftime(s2, s1, units = "mins")
# MISE of observations
diff2=0; num=0
for(i in 1:nrow(Y)){
  idx = which(!is.na(Y[i, ]))
  num = num + length(idx)
  diff2 = diff2 + sum(abs(fit2$Yhat[i,idx]-Y[i,idx])^2)
}
MISE2_Y <- diff2/num
# MISE of eigenfucntions
MISE2_eigen1 <- sum(unlist(lapply(1:K1, function(x){
  min(sum((eigenf_true[[1]][,x]-fit2$efunctions[[1]][,x])^2),
      sum((eigenf_true[[1]][,x]+fit2$efunctions[[1]][,x])^2))})))/(K1*D)
MISE2_eigen2 <- sum(unlist(lapply(1:K2, function(x){
  min(sum((eigenf_true[[2]][,x]-fit2$efunctions[[2]][,x])^2),
      sum((eigenf_true[[2]][,x]+fit2$efunctions[[2]][,x])^2))})))/(K2*D)
result2 <- round(c(as.numeric(time2), MISE2_Y, MISE2_eigen1, MISE2_eigen2),4)


result <- rbind(result1,result2)
colnames(result) <- c("computation time", "MISE(Y)", "MISE(Phi)", "MISE(Psi)")
print("The result from mfpca.sc is")
```

```
## [1] "The result from mfpca.sc is"
```

```r
print(result[1,])
```

```
## computation time          MISE(Y)        MISE(Phi)        MISE(Psi)
##           0.4694           0.0003           0.0286           0.0117
```

```r
print("The result from mfpca.sc2 is")
```

```
## [1] "The result from mfpca.sc2 is"
```

```r
print(result[2,])
```

```
## computation time          MISE(Y)        MISE(Phi)        MISE(Psi)
##           0.0119           0.0000           0.0282           0.0115
```

3.1.2 This exmaple applies regularly sampled functional data with $D = 1000$ and $\sigma = 1$.

I only used the `mfpca.cs2` function. It took less than one minute to get results while `mfpca.cs` took too long time to work for this case. Note that there is a new input parameter in `mfpca.cs2`, `design` with default value `irregular`. For regular data, `design=irregular` also works. The major difference is the way of smoothing sample covariances. Thus, I also set `design=irregular` so as to compare the computation time of the two smoothing methods.

```r
set.seed(1)
Nsub=100;J=3;D=1000;K1=4;K2=4
data <- GeneData(M=Nsub, J=J, N=D, design="regular", level=0.1, sigma=1)
Y <- data$Y
# Parameters
id <- rep(1:Nsub, each=J)
twoway <- TRUE

#1. the modified mfpca function: mfpca.cs2  with design="regular"
s1 <- Sys.time()
fit1 <-  mfpca.sc2(Y=Y, id=id, twoway=twoway, design="regular")
s2 <- Sys.time()
time1 <- difftime(s2, s1, units = "mins")

#2. the modified mfpca function: mfpca.cs2  with design="irregular"
s1 <- Sys.time()
fit2 <-  mfpca.sc2(Y=Y, id=id, twoway=twoway, design="irregular")
s2 <- Sys.time()
time2 <- difftime(s2, s1, units = "mins")


print(paste("The computetation time of mfpca.sc2 with design=regular is",
            round(as.numeric(time1),2),"mins"))
```

```
## [1] "The computetation time of mfpca.sc2 with design=regular is 0.22 mins"
```

```r
print(paste("The computetation time of mfpca.sc2 with design=irregular is",
             round(as.numeric(time2),2),"mins"))
```

```
## [1] "The computetation time of mfpca.sc2 with design=irregular is 6.72 mins"
```

3.2 Real data

I used DTI data shown in the `mfpca.sc` document. There are 130 subjects and $D = 93$ with some missings in the DTI data. The number of visits varies from one to five. The `mfpca.sc2` function saves much more time than `mfpca.sc` and the MISEs of two functions are the same.

```r
data(DTI)
DTI = subset(DTI, Nscans < 6)
id  = DTI$ID
Y = DTI$cca
#Y[which(is.na(Y)==1)] = 0

#1. mfpca.cs in refund
s1 <- Sys.time()
fit1 <-  mfpca.sc(Y=Y, id=id, twoway=TRUE)
s2 <- Sys.time()
time1 <- difftime(s2, s1, units = "mins")
```

```r
# MISE of observations
diff1=0;num=0
for(i in 1:nrow(Y)){
  idx = which(!is.na(Y[i, ]))
  num = num + length(idx)
  diff1 = diff1 + sum(abs(fit1$Yhat[i,idx]-Y[i,idx])^2)
}
MISE1_Y <- diff1/num
result1 <- round(c(as.numeric(time1), MISE1_Y),4)

#2. the modified mfpca function: mfpca.cs2
s1 <- Sys.time()
fit2 <-  mfpca.sc2(Y=Y, id=id, twoway=TRUE, design="irregular", weight="obs")
s2 <- Sys.time()
time2 <- difftime(s2, s1, units = "mins")
# MISE of observations
diff2=0; num=0
for(i in 1:nrow(Y)){
  idx = which(!is.na(Y[i, ]))
  num = num + length(idx)
  diff2 = diff2 + sum(abs(fit2$Yhat[i,idx]-Y[i,idx])^2)
}
MISE2_Y <- diff2/num
result2 <- round(c(as.numeric(time2), MISE2_Y),4)


result <- rbind(result1,result2)
colnames(result) <- c("computation time", "MISE(Y)")
print("The result from mfpca.sc is")
```

```
## [1] "The result from mfpca.sc is"
```

```r
print(result[1,])
```

```
## computation time          MISE(Y)
##           1.3727           0.0006
```

```r
print("The result from mfpca.sc2 is")
```

```
## [1] "The result from mfpca.sc2 is"
```

```r
print(result[2,])
```

```
## computation time          MISE(Y)
##           0.0730           0.0005
```

3.3 Restructure datasets

I used 3 cases to show that the `mfpca.sc2` function gives the same results for restructured datasets. The first one is the DTI dataset. The second is a dense dataset with unbalanced visits and the last one is a sparsee dataset with unbalanced visits.

3.3.1 The DTI datasets

```
data(DTI)
DTI = subset(DTI, Nscans < 6)
id  = DTI$ID
visit = ave(id, id, FUN=seq_along)
Y = DTI$cca
fit1 <-  mfpca.sc2(Y=Y, id=id, twoway=TRUE, visit=visit,
                   design="irregular", weight="obs")

# restructure DTI
set.seed(1)
idx = sample(1:nrow(Y), nrow(Y))
id_new  = id[idx]
visit_new = visit[idx]
Y_new = Y[idx,]
fit2 <-  mfpca.sc2(Y=Y_new, id=id_new, visit=visit_new,
                   twoway=TRUE, design="irregular", weight="obs")

# difference between mean functions
diff_mu = sum(abs(fit1$mu - fit2$mu))
diff_eta = sum(abs(fit1$eta - fit2$eta))
# difference between eigen components
diff_evalues = sum(abs(unlist(fit1$evalues) - unlist(fit2$evalues)))
diff_efunctions = sum(unlist(lapply(1:length(fit1$evalues[[1]]), function(x){
      min(sum(abs(fit1$efunctions[[1]][,x]-fit2$efunctions[[1]][,x])),
          sum(abs(fit1$efunctions[[1]][,x]+fit2$efunctions[[1]][,x])))
  + min(sum(abs(fit1$efunctions[[2]][,x]-fit2$efunctions[[2]][,x])),
        sum(abs(fit1$efunctions[[2]][,x]+fit2$efunctions[[2]][,x])))
    })))
# difference between estimated observation
diff_Yhat = sum(abs(fit1$Yhat-fit2$Yhat[order(idx),]))

difference <- t(as.matrix(c(diff_mu, diff_eta, diff_evalues,
                            diff_efunctions, diff_Yhat)))
colnames(difference) <- c("mu", "eta", "eigenvalues", "eigenfunctions", "Yhat")
print("Difference between estimations")
```

```
## [1] "Difference between estimations"
```

```
print(difference)
```

```
##      mu eta  eigenvalues eigenfunctions         Yhat
## [1,]  0   0 1.137396e-17   3.667957e-11 6.206657e-11
```

3.3.2 A dense dataset with unbalanced visits

```
set.seed(8)
Nsub=100;J=5;D=200;K1=4;K2=4
data <- GeneData(M=Nsub, J=J, N=D, design="regular", level=0.5, sigma=1)
Y0 <- data$Y
id0 <- rep(1:Nsub, each=J)
visit0 <- rep(1:J, Nsub)
```

```r
# select rows
set.seed(5)
idx <- sort(sample(1:(J*Nsub), 0.8*J*Nsub))
Y <- Y0[idx,]
visit <- visit0[idx]
id <- id0[idx]
fit1 <-  mfpca.sc2(Y=Y, id=id, visit=visit, design="regular", weight="obs")

# restructure the dataset
set.seed(1)
idx = sample(1:nrow(Y), nrow(Y))
id_new  = id[idx]
visit_new = visit[idx]
Y_new = Y[idx,]
fit2 <-  mfpca.sc2(Y=Y_new, id=id_new, visit=visit_new,
                   design="regular", weight="obs")

# difference between mean functions
diff_mu = sum(abs(fit1$mu - fit2$mu))
diff_eta = sum(abs(fit1$eta - fit2$eta))
# difference between eigen components
diff_evalues = sum(abs(unlist(fit1$evalues) - unlist(fit2$evalues)))
diff_efunctions = sum(unlist(lapply(1:K1, function(x){
      min(sum(abs(fit1$efunctions[[1]][,x]-fit2$efunctions[[1]][,x])),
          sum(abs(fit1$efunctions[[1]][,x]+fit2$efunctions[[1]][,x])))
  + min(sum(abs(fit1$efunctions[[2]][,x]-fit2$efunctions[[2]][,x])),
        sum(abs(fit1$efunctions[[2]][,x]+fit2$efunctions[[2]][,x])))
    })))
# difference between estimated observation
diff_Yhat = sum(abs(fit1$Yhat-fit2$Yhat[order(idx),]))

difference <- t(as.matrix(c(diff_mu, diff_eta, diff_evalues,
                            diff_efunctions, diff_Yhat)))
colnames(difference) <- c("mu", "eta", "eigenvalues", "eigenfunctions", "Yhat")
print("Difference between estimations")
```

```
## [1] "Difference between estimations"
```

```r
print(difference)
```

```
##      mu eta  eigenvalues eigenfunctions         Yhat
## [1,]  0   0 3.407193e-12   2.998448e-08 4.040061e-06
```

3.3.3 An irregular dataset with unbalanced visits

```r
set.seed(10)
Nsub=100;J=5;D=200;K1=4;K2=4
data <- GeneData(M=Nsub, J=J, N=D, design="irregular", level=0.5, sigma=1)
Y0 <- data$Y
id0 <- rep(1:Nsub, each=J)
visit0 <- rep(1:J, Nsub)
# select rows
set.seed(55)
```

```r
idx <- sort(sample(1:(J*Nsub), 0.8*J*Nsub))
Y <- Y0[idx,]
visit <- visit0[idx]
id <- id0[idx]
fit1 <-  mfpca.sc2(Y=Y, id=id, visit=visit,
                   design="irregular", weight="obs")

# restructure the dataset
set.seed(10)
idx = sample(1:nrow(Y), nrow(Y))
id_new  = id[idx]
visit_new = visit[idx]
Y_new = Y[idx,]
fit2 <-  mfpca.sc2(Y=Y_new, id=id_new, visit=visit_new,
                   design="irregular", weight="obs")

# difference between mean functions
diff_mu = sum(abs(fit1$mu - fit2$mu))
diff_eta = sum(abs(fit1$eta - fit2$eta))
# difference between eigen components
diff_evalues = sum(abs(unlist(fit1$evalues) - unlist(fit2$evalues)))
diff_efunctions  = sum(unlist(lapply(1:K1, function(x){
      min(sum(abs(fit1$efunctions[[1]][,x]-fit2$efunctions[[1]][,x])),
          sum(abs(fit1$efunctions[[1]][,x]+fit2$efunctions[[1]][,x])))
  + min(sum(abs(fit1$efunctions[[2]][,x]-fit2$efunctions[[2]][,x])),
        sum(abs(fit1$efunctions[[2]][,x]+fit2$efunctions[[2]][,x])))
    })))
# difference between estimated observation
diff_Yhat  = sum(abs(fit1$Yhat-fit2$Yhat[order(idx),]))

difference <- t(as.matrix(c(diff_mu, diff_eta, diff_evalues,
                            diff_efunctions, diff_Yhat)))
colnames(difference) <- c("mu", "eta", "eigenvalues", "eigenfunctions", "Yhat")
print("Difference between estimations")
```

```
## [1] "Difference between estimations"
```

```r
print(difference)
```

```
##      mu eta  eigenvalues eigenfunctions         Yhat
## [1,]  0   0 5.218048e-15   3.450336e-12 6.039643e-10
```

4 The new parameter `weight`.

Considering the number of visits may change across subjects, we introduce a new parameter `weight` to compromise the difference of visits per subject. If `weight=obs`, the sample estimate is weighted by observations. If `weight=subjs`, the sample estimate is weighted by subjects. It would help the cases in which some subjects have lots of visits while others may have few. Note that `weight` only works for `regular` design. For `irregular` data, the sample covariance is always estimated under `weight=obs`.

4.1

Part 1: `weight=obs`.

Let $Y_{ij}(t) = X_{ij}(t) - \mu(t) - \eta_j(t)$. The sample mean is estimated by $\hat{\mu}(t) = \sum_{i=1}^{n}\sum_{j=1}^{n_i} X_{ij}(t)/\sum_{i=1}^{n} n_i$. Furthermore, $K_T(t_s, t_r) = \sum_{i=1}^{n}\sum_{j=1}^{n_i}\{Y_{ij}(t_s)Y_{ij}(t_r)\}/\sum_{i=1}^{n} n_i$ and $K_B(t_s, t_r) = 2\sum_{i=1}^{n}\sum_{j_1<j_2}\{Y_{ij_1}(t_s)Y_{ij_2}(t_r)\}/\sum_{i=1}^{n} n_i(n_i-1)$.

Part 2: `weight=subj`.

The sample mean is estimated by $\hat{\mu}(t) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{n_i}\sum_{j=1}^{n_i} X_{ij}(t)$. Furthermore, $K_T(t_s, t_r) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{n_i}\sum_{j=1}^{n_i} Y_{ij}(t_s)Y_{ij}(t_r)$ and $K_B(t_s, t_r) = \frac{2}{n}\sum_{i=1}^{n}\frac{1}{n_i(n_i-1)}\sum_{j_1<j_2} Y_{ij_1}(t_s)Y_{ij_2}(t_r)$.

4.2 Simualtion

I used the regular functional data with $D = 100$ in Section~3. The number of visits is 3 per subject. Thus, we would get the same results under `weight=obs` and `weight=subj`. For the second scenario, I generated regular functional data with n_i varies from 1 to 5. I simulated 100 data sets for each scenario. Results are shown in Table 2.

Table 2: The average MISE for eigenfunctions and $\mathbf{Y}$ across 100 simulations.

	weight=obs			weight=subj		
	MISE($\mathbf{Y}$)	MISE(Φ)	MISE(Ψ)	MISE($\mathbf{Y}$)	MISE(Φ)	MISE(Ψ)
		$n_i = 3$				
$\sigma = 0$	0	0.0815	0.0197	0	0.0815	0.0197
$\sigma = 1$	0.9512	0.0865	0.0232	0.9512	0.0865	0.0232
		$n_i \sim Unif[1,5]$				
$\sigma = 0$	0.0398	0.1475	0.0688	0.0359	0.1122	0.1138
$\sigma = 1$	0.9876	0.1356	0.0919	0.9761	0.1192	0.1065