Fast MFPCA and MFPCA

Erjia Cui, Ruonan Li

9/2021

Introduction

This document provides two examples on the time difference between traditional MFPCA and fast MFPCA under different sizes of the simulated data. For the first example, we simulate a complete unbalanced multilevel functional dataset with $I = 100, J = 2, L = 100$. In this case, both methods run fast. For the second example, we simulate a complete unbalanced multilevel functional dataset with $I = 100, J = 2, L = 1000$. In this case, the traditional MFPCA takes several hours, while the fast MFPCA finishes in less than 10 seconds. Readers are encouraged to change parameters to compare the computational performance between two methods.

```
library(refund)
library(MASS)
library(Matrix)
library(mgcv)
library(simex)
library(splines)
library(rARPACK)
library(dplyr)
source("./backup.R")
source("./GeneData.R")
source("./mfpca.face.R")
```

Example 1: small data

```
## set parameters
I <- 100
J <- 2
L <- 100
K1 <- 4
K2 <- 4
design <- "regular"
balance <- FALSE
sigma <- 1

## create a data frame storing simulation results
nsim <- 1
sim_res <- data.frame(matrix(NA, nrow = nsim*2, ncol = 8), rep(NA,nsim*2))
colnames(sim_res) <- c("I", "J", "L", "iteration",
                       "comptime", "MISE(Y)", "MISE(Phi)", "MISE(Psi)", "method")
```

```r
## start simulation
ind <- 1
for(iter in 1:nsim){
  set.seed(iter)
  data <- GeneData(I = I, J = J, L = L, design = design, sigma = sigma,
                   balanced = balance, level = 0.4)
  Y <- data$Y
  ## true eigenvalues and eigenfunctions
  evalues_true <- data$evalues
  eigenf_true <- data$eigenfunctions
  ## other parameters for estimation
  id <- data$id

  #################################################################
  ## fit MFPCA using mfpca.face() (fast MFPCA)
  ptm <- proc.time()
  fit_fast <- mfpca.face(Y = data$Y, id = id, weight = "obs")
  time_fast <- proc.time() - ptm

  ## 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(fit_fast$Xhat[i,idx]-Y[i,idx])^2)
  }
  MISE2_Y <- diff2/num
  ## MISE of eigenfucntions
  MISE2_eigen1 <- sum(unlist(lapply(1:K1, function(x){
    min(sum((eigenf_true[[1]][,x]-fit_fast$efunctions[[1]][,x])^2),
        sum((eigenf_true[[1]][,x]+fit_fast$efunctions[[1]][,x])^2))})))/(K1*L)
  MISE2_eigen2 <- sum(unlist(lapply(1:K2, function(x){
    min(sum((eigenf_true[[2]][,x]-fit_fast$efunctions[[2]][,x])^2),
        sum((eigenf_true[[2]][,x]+fit_fast$efunctions[[2]][,x])^2))})))/(K2*L)
  sim_res[ind,1:8] <- round(c(I, J, L, iter, time_fast[3], MISE2_Y,
                              MISE2_eigen1, MISE2_eigen2),4)
  sim_res[ind,9] <- "mfpca.face"
  ind <- ind + 1

  #################################################################
  ## fit MFPCA using mfpca.sc() (traditional MFPCA)
  ptm <- proc.time()
  fit_sc <- mfpca.sc(Y = data$Y, id = id, twoway = TRUE)
  time_sc <- proc.time() - ptm

  ## 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(fit_sc$Yhat[i,idx]-Y[i,idx])^2)
  }
  MISE1_Y <- diff1/num
  ## MISE of eigenfucntions
  MISE1_eigen1 <- sum(unlist(lapply(1:K1, function(x){
    min(sum((eigenf_true[[1]][,x]-fit_sc$efunctions[[1]][,x])^2),
        sum((eigenf_true[[1]][,x]+fit_sc$efunctions[[1]][,x])^2))})))/(K1*L)
  MISE1_eigen2 <- sum(unlist(lapply(1:K2, function(x){
    min(sum((eigenf_true[[2]][,x]-fit_sc$efunctions[[2]][,x])^2),
        sum((eigenf_true[[2]][,x]+fit_sc$efunctions[[2]][,x])^2))})))/(K2*L)
  sim_res[ind,1:8] <- round(c(I, J, L, iter, time_sc[3], MISE1_Y,
                            MISE1_eigen1, MISE1_eigen2),4)
  sim_res[ind,9] <- "mfpca.sc"
  ind <- ind + 1

  # print(iter)
}

## summarise the results of nsim iterations
sim_res_lite <- sim_res %>%
  group_by(I, J, L, method) %>%
  summarise(comptime = median(comptime), `MISE(Y)` = median(`MISE(Y)`),
          `MISE(Phi)` = median(`MISE(Phi)`), `MISE(Psi)` = median(`MISE(Psi)`))
```

```
## ‘summarise()‘ has grouped output by ’I’, ’J’, ’L’. You can override using the ‘.groups‘ argument.
```

```
sim_res_lite
```

```
## # A tibble: 2 x 8
## # Groups:   I, J, L [1]
##       I     J     L method     comptime ‘MISE(Y)‘ ‘MISE(Phi)‘ ‘MISE(Psi)‘
##   <dbl> <dbl> <dbl> <chr>         <dbl>     <dbl>       <dbl>       <dbl>
## 1   100     2   100 mfpca.face     0.24     0.951      0.0544      0.0337
## 2   100     2   100 mfpca.sc       9.74     0.932      0.142       0.142
```

In this small data example, the fast MFPCA finishes calculation in less than 1 second, while the traditional MFPCA takes about 10 seconds on a regular laptop (Apple M1 processor).

Example 2: large data

```
## set parameters
I <- 100
J <- 2
L <- 1000
K1 <- 4
K2 <- 4
design <- "regular"
balance <- FALSE
sigma <- 1

## create a data frame storing simulation results
```

```r
nsim <- 1
sim_res <- data.frame(matrix(NA, nrow = nsim*2, ncol = 8), rep(NA,nsim*2))
colnames(sim_res) <- c("I", "J", "L", "iteration",
                       "comptime", "MISE(Y)", "MISE(Phi)", "MISE(Psi)", "method")
## start simulation
ind <- 1
for(iter in 1:nsim){
  set.seed(iter)
  data <- GeneData(I = I, J = J, L = L, design = design, sigma = sigma,
                   balanced = balance, level = 0.4)
  Y <- data$Y
  ## true eigenvalues and eigenfunctions
  evalues_true <- data$evalues
  eigenf_true <- data$eigenfunctions
  ## other parameters for estimation
  id <- data$id

  #############################################################
  ## fit MFPCA using mfpca.face() (fast MFPCA)
  ptm <- proc.time()
  fit_fast <- mfpca.face(Y = data$Y, id = id, weight = "obs")
  time_fast <- proc.time() - ptm

  ## 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(fit_fast$Xhat[i,idx]-Y[i,idx])^2)
  }
  MISE2_Y <- diff2/num
  ## MISE of eigenfucntions
  MISE2_eigen1 <- sum(unlist(lapply(1:K1, function(x){
    min(sum((eigenf_true[[1]][,x]-fit_fast$efunctions[[1]][,x])^2),
        sum((eigenf_true[[1]][,x]+fit_fast$efunctions[[1]][,x])^2))})))/(K1*L)
  MISE2_eigen2 <- sum(unlist(lapply(1:K2, function(x){
    min(sum((eigenf_true[[2]][,x]-fit_fast$efunctions[[2]][,x])^2),
        sum((eigenf_true[[2]][,x]+fit_fast$efunctions[[2]][,x])^2))})))/(K2*L)
  sim_res[ind,1:8] <- round(c(I, J, L, iter, time_fast[3], MISE2_Y,
                              MISE2_eigen1, MISE2_eigen2),4)
  sim_res[ind,9] <- "mfpca.face"
  ind <- ind + 1

  #############################################################
  ## fit MFPCA using mfpca.sc() (traditional MFPCA)
  ptm <- proc.time()
  fit_sc <- mfpca.sc(Y = data$Y, id = id, twoway = TRUE)
  time_sc <- proc.time() - ptm

  ## 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(fit_sc$Yhat[i,idx]-Y[i,idx])^2)
  }
  MISE1_Y <- diff1/num
  ## MISE of eigenfucntions
  MISE1_eigen1 <- sum(unlist(lapply(1:K1, function(x){
    min(sum((eigenf_true[[1]][,x]-fit_sc$efunctions[[1]][,x])^2),
        sum((eigenf_true[[1]][,x]+fit_sc$efunctions[[1]][,x])^2))})))/(K1*L)
  MISE1_eigen2 <- sum(unlist(lapply(1:K2, function(x){
    min(sum((eigenf_true[[2]][,x]-fit_sc$efunctions[[2]][,x])^2),
        sum((eigenf_true[[2]][,x]+fit_sc$efunctions[[2]][,x])^2))})))/(K2*L)
  sim_res[ind,1:8] <- round(c(I, J, L, iter, time_sc[3], MISE1_Y,
                              MISE1_eigen1, MISE1_eigen2),4)
  sim_res[ind,9] <- "mfpca.sc"
  ind <- ind + 1

  # print(iter)
}

## summarise the results of nsim iterations
sim_res_lite <- sim_res %>%
  group_by(I, J, L, method) %>%
  summarise(comptime = median(comptime), `MISE(Y)` = median(`MISE(Y)`),
            `MISE(Phi)` = median(`MISE(Phi)`), `MISE(Psi)` = median(`MISE(Psi)`))
```

```
## `summarise()` has grouped output by 'I', 'J', 'L'. You can override using the `.groups` argument.
```

```r
sim_res_lite
```

```
## # A tibble: 2 x 8
## # Groups:   I, J, L [1]
##       I     J     L method      comptime `MISE(Y)` `MISE(Phi)` `MISE(Psi)`
##   <dbl> <dbl> <dbl> <chr>          <dbl>     <dbl>       <dbl>       <dbl>
## 1   100     2  1000 mfpca.face     0.356     1.00       0.0637      0.0388
## 2   100     2  1000 mfpca.sc   12374.        0.998      0.164       0.125
```

In this large data example, the fast MFPCA finishes calculation in less than 1 second, while the traditional MFPCA takes several hours on the same laptop.