R code file for fixed-effects inference for longitudinal functional data

Ruonan Li[*]
Luo Xiao[†]
Ekaterina Smirnova[‡]
Erjia, Cui[§]
Andrew, Leroux[¶]
Ciprian M. Crainiceanu[‖]

March 19, 2021

Introduction

In this document, a complete implementation of the proposed fixed-effects estimate method considering complex functional mixed-effects is evaluated. Moreover, comparisons with the navie estimate by `pffr` function and the bootstrap method are evaluated using a dataset (100 subjects whose visit times follow Unif[3,6]) with unspeicified covariance structure. The document is organized as follows.

Section 1 gives the main code (function `fgee`) corresponding to the proposed fixed-effects estimate method, and the code (function `fgee.bp`), corresponding to the bootstrap method in the paper.

Section 2 gives the code to generate datasets (function `GenerateData`) which are used to evaluate fixed-effects estimate methods in section 1. We can use the `GenerateData` function to generate data with three different types of covariance structures. They are independent, exchangeable and unspecified. We apply the several methods on the same dataset in Section 2.2. In bootstrap method, the default number of resampling is 300. However, since the bootstrap method takes a long time to get results (about 16 minutes with resampling 300 times and about 2.5 minutes with resampling 50 times), we commented the part of implementing bootstrap by pound symbol. If you are interest in bootstrap results, please delete pound symbols in this part and run the code.

Section 3 visualizes the outputs, including eigendecomposition, the mean function estimated by `pffr` function and the mean function estimated by `fgee` function.

Section 4 shows hypothesis tests about covariance structures. Functions about testing are given and we show numerical results of testing the covariance structure of three types of data. Due to time limit, we only show the results of testing data with unspecifided covariance structure and put pound symbols on other types of data. The testing methods depend on bootstrap to get p-values and the default number of resmapling is 1000 (takeing about 1 minutes). It takes about 3 minutes to compile this Rmd file.

1. Main code

```
##      Package LibPath Version Priority Depends Imports LinkingTo Suggests
##      Enhances License License_is_FOSS License_restricts_use OS_type Archs
##      MD5sum NeedsCompilation Built
```

2. Generate a sample dataset (100 subjects, visit times follow Unif[3,6]), and evaluate the function `fgee`

2.1 Generating functon

[*]Department of Statistics, North Carolina State University
[†]Department of Statistics, North Carolina State University
[‡]Department of Biostatistics, Virginia Commonwealth University
[§]Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health
[¶]Department of Biostatistics and Informatics, Colorado School of Public Health
[‖]Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health

2.2 Generate data with unspecified covariance structure and estimate

The mean function $\mu_{ij}(s) = \sqrt{2}\cos(3\pi s) + 1 + x_{1ij}(2 + \cos(2\pi s)) + x_{2ij}(2 + \sin(\pi s))$. We estimate the mean function by five models. 1. The naive estimation by `pffr` function without considering any covariance structure. 2. The bootstrap method to get correct confidence bands for naive estimation (as menthioned before, since this part took a long time, we commented it). 3. The fgee estimation with independent covariance. 4. The fgee estimation with exchangeable covariance. 5. The fgee estimation with unspecified covariance.

```
library(ggplot2)
library(refund)
library(face)
library(fields)
library(mgcv)
library(gridExtra)

# generate dataset with unspecified covariance
set.seed(2021)
data <- GenerateData(Nsubj=100, numFunctPoints=101, min_visit=3, max_visit=6,
                     numLongiPoints=41, sigma_sq=1.5, sigma_z11=3, sigma_z12=1.5,
                     sigma_z21=2, sigma_z22=1, corstr = "unspecified")

# bootstrape to get correct confidence bands for naive estimation
# bp.estim <- fgee_initBp(formula = Y ~ X1 + X2,
#                                Y=data$data$Y, Cov=data$data$Cov,
#                                s=data$data$funcArg, subjID=data$data$subjID,
#                                Tij=data$data$Tij, n.Bp=300)

# estimate mean functions with independent covariance
iid.estim <- fgee(formula = Y ~ X1 + X2, Y=data$data$Y, Cov=data$data$Cov,
              s=data$data$funcArg, subjID=data$data$subjID,
              Tij=data$data$Tij, numLongiPoints=41,
              control=list(FPCA1.pve=0.95, FPCA1.knots=20),
              corstr = "independent")

# estimate mean functions with exchangeable covariance
exch.estim <- fgee(formula = Y ~ X1 + X2, Y=data$data$Y, Cov=data$data$Cov,
              s=data$data$funcArg, subjID=data$data$subjID,
              Tij=data$data$Tij, numLongiPoints=41,
              control=list(FPCA1.pve=0.95, FPCA1.knots=20),
              corstr = "exchangeable")

# estimate mean functions with unspecified covariance
unsp.estim <- fgee(formula = Y ~ X1 + X2, Y=data$data$Y, Cov=data$data$Cov,
              s=data$data$funcArg, subjID=data$data$subjID,
              Tij=data$data$Tij, numLongiPoints=41,
              control=list(FPCA1.pve=0.95, FPCA2.pve=0.95, FPCA1.knots=20, FPCA2.knots=15),
              corstr = "unspecified")
```

3. Visualize results

3.1 Visualize estimated eigenfunctions vs. true eigenfunctions

```
### FPCA ###
flip <-  function(fn_hat, fn){
  len = length(fn)
```

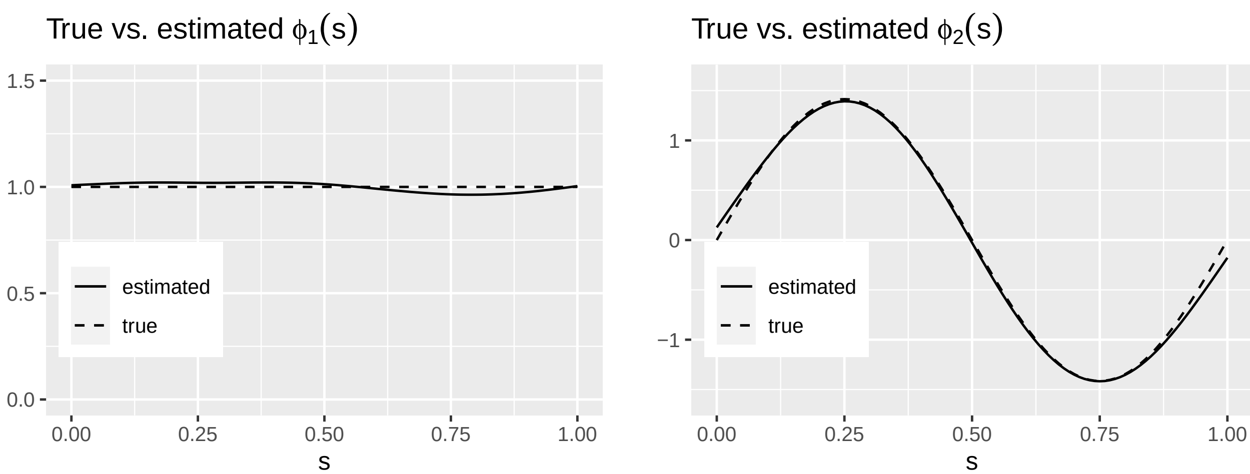
```
  dif1 <- sum((fn - fn_hat)^2)/len #if signs are the same
  dif2 <- sum((fn + fn_hat)^2)/len #if signs are opposite
  Ind <- which(c(dif1,dif2) == min(dif1, dif2))
  if(Ind==1){
    return(1)
  }else{
    return(-1)
    }
}

# true eigen-functions
efun.true <- data$phi
sign1 <- flip(unsp.estim$phi[,1], efun.true$phi_1)
sign2 <- flip(unsp.estim$phi[,2], efun.true$phi_2)
efun <- cbind(sign1*unsp.estim$phi[,1], sign2*unsp.estim$phi[,2])

phi1 <- data.frame(estimated=efun[,1], true=efun.true$phi_1, s=seq(0,1,length=101))
phi2 <- data.frame(estimated=efun[,2], true=efun.true$phi_2, s=seq(0,1,length=101))

g1 = ggplot() +
  geom_line(data=phi1, aes(x=s, y=estimated, linetype="estimated")) +
  geom_line(data=phi1, aes(x=s, y=true, linetype="true")) +
  labs(title = expression(paste("True vs. estimated ", phi[1](s))), x="s", y="") +
  scale_linetype_manual(values=c('estimated'='solid','true'='dashed')) +
  ylim(0,1.5)+theme(legend.position=c(0.17,0.33),legend.title=element_blank())

g2 = ggplot() +
  geom_line(data=phi2, aes(x=s, y=estimated, linetype="estimated")) +
  geom_line(data=phi2, aes(x=s, y=true, linetype="true")) +
  labs(title = expression(paste("True vs. estimated ", phi[2](s))), x="s", y="") +
  scale_linetype_manual(values=c('estimated'='solid','true'='dashed')) +
  ylim(-1.6,1.6)+theme(legend.position=c(0.17,0.33),legend.title=element_blank())
figure <- grid.arrange(g1, g2, ncol=2)
```



3.2 Visualize estimated β functions given by `fgee` with unspecified covariance and given by `pffr` function

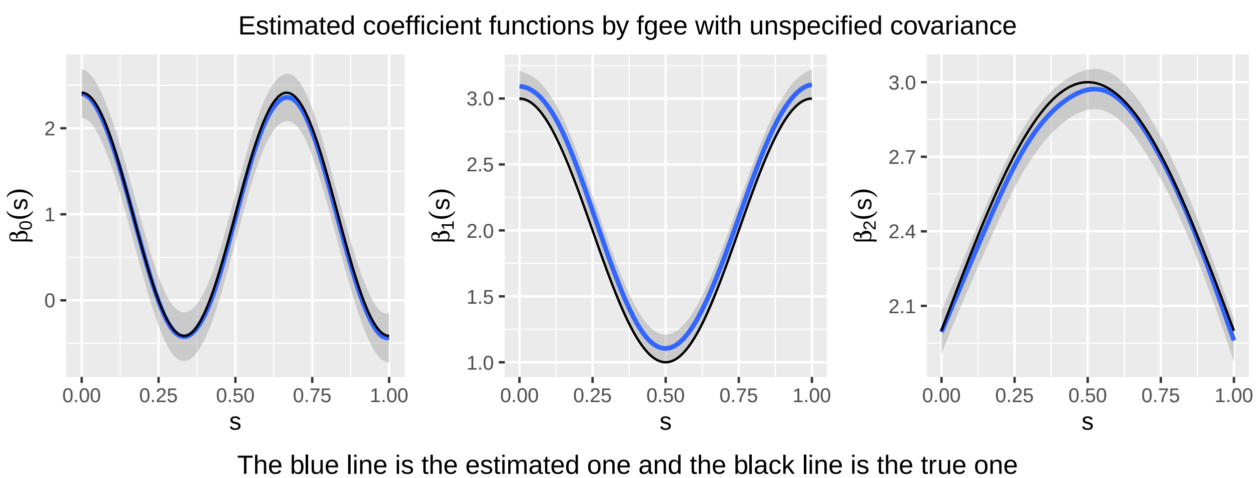
```
# estimated beta by fgee with unspecified covariance ###
beta.unsp <- unsp.estim$beta
```

```r
beta.unsp.se <- unsp.estim$beta.se

lower <- beta.unsp - qnorm(0.975)*beta.unsp.se
upper <- beta.unsp + qnorm(0.975)*beta.unsp.se
s <- seq(0, 1, length = 101)
unsp_beta <- data.frame(beta0=beta.unsp[,1], beta0.true=sqrt(2)*cos(3*pi*s)+1,
                        beta0_l=lower[,1], beta0_u=upper[,1],
                        beta1=beta.unsp[,2], beta1.true=2 + cos(2*pi*s),
                        beta1_l=lower[,2], beta1_u=upper[,2],
                        beta2=beta.unsp[,3], beta2.true=2 + sin(pi*s),
                        beta2_l=lower[,3], beta2_u=upper[,3],
                        s=seq(0, 1, length=101))

p1 <- ggplot(unsp_beta, aes(x=s, y=beta0)) +
  geom_smooth(aes(ymin=beta0_l, ymax=beta0_u), alpha=0.4, stat="identity") +
  geom_line(aes(x=s, y=beta0.true)) +
  xlab("s") + ylab(expression(beta[0](s)))
p2 <-  ggplot(unsp_beta, aes(x = s, y = beta1)) +
  geom_smooth(aes(ymin=beta1_l, ymax=beta1_u), alpha=0.4, stat="identity") +
  geom_line(aes(x=s, y=beta1.true)) +
  xlab("s") + ylab(expression(beta[1](s)))
p3 <- ggplot(unsp_beta, aes(x=s, y=beta2)) +
  geom_smooth(aes(ymin=beta2_l, ymax=beta2_u), alpha=0.4, stat="identity") +
  geom_line(aes(x=s, y=beta2.true)) +
  xlab("s") + ylab(expression(beta[2](s)))

figure <- grid.arrange(p1, p2, p3, ncol=3,
          top="Estimated coefficient functions by fgee with unspecified covariance",
  bottom ="The blue line is the estimated one and the black line is the true one")
```



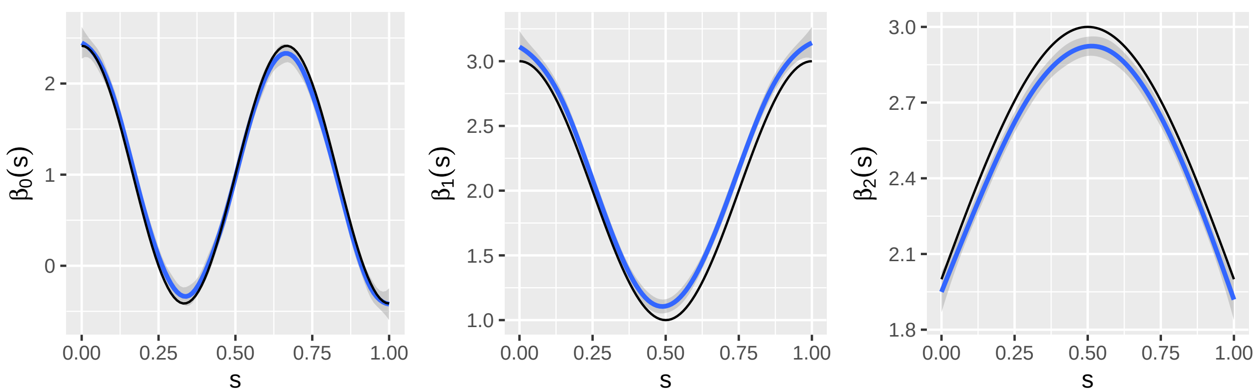

```r
# estimated beta by naive pffr function
beta.naive <- unsp.estim$beta.init
beta.naive.se <- unsp.estim$beta.init.se

lower <- beta.naive - qnorm(0.975)*beta.naive.se
upper <- beta.naive + qnorm(0.975)*beta.naive.se
naive_beta <- data.frame(beta0=beta.naive[,1], beta0.true=sqrt(2)*cos(3*pi*s)+1,
                         beta0_l=lower[,1], beta0_u=upper[,1],
                       beta1=beta.naive[,2], beta1.true=2 + cos(2*pi*s),
```

```
                                 beta1_l=lower[,2], beta1_u=upper[,2],
                                 beta2=beta.naive[,3], beta2.true=2 + sin(pi*s),
                                 beta2_l=lower[,3], beta2_u=upper[,3],
                                 s=seq(0, 1, length=101))

p1 <- ggplot(naive_beta, aes(x=s, y=beta0)) +
  geom_smooth(aes(ymin=beta0_l, ymax=beta0_u), alpha=0.4, stat="identity") +
  geom_line(aes(x=s, y=beta0.true)) +
  xlab("s") + ylab(expression(beta[0](s)))
p2 <-  ggplot(naive_beta, aes(x = s, y = beta1)) +
  geom_smooth(aes(ymin=beta1_l, ymax=beta1_u), alpha=0.4, stat="identity") +
  geom_line(aes(x=s, y=beta1.true)) +
  xlab("s") + ylab(expression(beta[1](s)))
p3 <- ggplot(naive_beta, aes(x=s, y=beta2)) +
  geom_smooth(aes(ymin=beta2_l, ymax=beta2_u), alpha=0.4, stat="identity") +
  geom_line(aes(x=s, y=beta2.true)) +
  xlab("s") + ylab(expression(beta[2](s)))
figure <- grid.arrange(p1, p2, p3, ncol=3,
                            top="Estimated coefficient functions by naive pffr function",
bottom ="The blue line is the estimated one and the black line is the true one")
```

Estimated coefficient functions by naive pffr function



The blue line is the estimated one and the black line is the true one

```
# estimated beta by bootstrap to get better confidence bands ###
# beta.Bp <- bp.estim$beta.init
# lower <- bp.estim$beta.band.lower
# upper <- bp.estim$beta.band.upper
# Bp_beta = data.frame(beta0=beta.Bp[,1], beta0.true=sqrt(2)*cos(3*pi*s)+1,
#                      beta0_l=lower[,1], beta0_u=upper[,1],
#                      beta1=beta.Bp[,2], beta1.true=2 + cos(2*pi*s),
#                      beta1_l=lower[,2], beta1_u=upper[,2],
#                      beta2=beta.Bp[,3], beta2.true=2 + sin(pi*s),
#                      beta2_l=lower[,3], beta2_u=upper[,3],
#                      s=seq(0, 1, length=101))
# p1 <- ggplot(Bp_beta, aes(x = s, y = beta0)) +
#   geom_smooth(aes(ymin=beta0_l, ymax=beta0_u), alpha=0.4, stat="identity") +
#   geom_line(aes(x=s, y=beta0.true)) +
#   xlab("s") + ylab(expression(beta[0](s)))
# p2 <-  ggplot(Bp_beta, aes(x = s, y = beta1)) +
#   geom_smooth(aes(ymin=beta1_l, ymax=beta1_u), alpha=0.4, stat="identity") +
#   geom_line(aes(x=s, y=beta1.true)) +
```

```
#   xlab("s") + ylab(expression(beta[1](s)))
# p3 <-  ggplot(Bp_beta, aes(x = s, y = beta2)) +
#   geom_smooth(aes(ymin=beta2_l, ymax=beta2_u), alpha=0.4, stat="identity") +
#   geom_line(aes(x=s, y=beta2.true)) +
#   xlab("s") + ylab(expression(beta[2](s)))
# figure <- grid.arrange(p1, p2, p3, ncol=3,
#                         top="Estimated coefficient functions by bootstrap",
#  bottom ="The blue line is the estimated one and the black line is the true one")
```

3.3 Calculate $\sqrt{\text{MISE}}$, IAW and IAC for each method

```
# function to estimate sqrt(MISE) of each beta
beta_MISE <- function(beta.hat, beta.true, numFunctPoints){
    beta0 <- sum((beta.hat[,1]-beta.true[,1])^2)/numFunctPoints
    beta1 <- sum((beta.hat[,2]-beta.true[,2])^2)/numFunctPoints
    beta2 <- sum((beta.hat[,3]-beta.true[,3])^2)/numFunctPoints
    return(c(sqrt(beta0), sqrt(beta1), sqrt(beta2)))
}

# function to estimate IAW and IAC of each beta
eval.beta.CI <- function(beta.hat, beta.se, beta.true, numFunctPoints){
  # fixed effects functions  beta(s)
  beta0 <- beta.true[,1]
  beta1 <- beta.true[,2]
  beta2 <- beta.true[,3]

  # IAW of 95% confidence interval
  IAW <- c(sum(2*qnorm(0.975)*beta.se[,1]),sum(2*qnorm(0.975)*beta.se[,2]),
                sum(2*qnorm(0.975)*beta.se[,3]))/numFunctPoints

  # beta0(s)
  lower <- beta.hat[,1] - qnorm(0.975)*beta.se[,1]
  upper <- beta.hat[,1] + qnorm(0.975)*beta.se[,1]
  # count number of points
  k1 <- 0
  for(i in 1:numFunctPoints){
    if(beta0[i]>=lower[i] & beta0[i]<=upper[i])
      k1 <- k1 + 1
  }
  IAC.beta0 <- k1/numFunctPoints
  # beta1(s)
  lower <- beta.hat[,2] - qnorm(0.975)*beta.se[,2]
  upper <- beta.hat[,2] + qnorm(0.975)*beta.se[,2]
  # count number of points
  k2 <- 0
  for(i in 1:numFunctPoints){
    if(beta1[i]>=lower[i] & beta1[i]<=upper[i])
      k2 <- k2 + 1
  }
  IAC.beta1 <- k2/numFunctPoints
  # beta2(s)
  lower <- beta.hat[,3] - qnorm(0.975)*beta.se[,3]
  upper <- beta.hat[,3] + qnorm(0.975)*beta.se[,3]
  # count number of points
```

```r
  k3 <- 0
  for(i in 1:numFunctPoints){
    if(beta2[i]>=lower[i] & beta2[i]<=upper[i])
      k3 <- k3 + 1
  }
  IAC.beta2 <- k3/numFunctPoints

  IAC <- c(IAC.beta0, IAC.beta1, IAC.beta2)

  return(list(IAW=IAW, IAC=IAC))
}

# function to estimate IAW and IAC of bootstrap confiddence bands
eval.betaBp.CI <- function(beta.lower, beta.upper, beta.true, numFunctPoints){
  # fixed effects functions  beta(s)
  beta0 <- beta.true[,1]
  beta1 <- beta.true[,2]
  beta2 <- beta.true[,3]

  beta0.band <- cbind(beta.lower[,1], beta.upper[,1])
  beta1.band <- cbind(beta.lower[,2], beta.upper[,2])
  beta2.band <- cbind(beta.lower[,3], beta.upper[,3])

  # IAW of 95% confidence interval
  IAW <- c(sum(beta0.band[,2]-beta0.band[,1]),sum(beta1.band[,2]-beta1.band[,1]),
           sum(beta2.band[,2]-beta2.band[,1]))/numFunctPoints

  # beta0(s)
  k1 <- 0
  for(i in 1:numFunctPoints){
    if(beta0[i]>=beta0.band[i,1] & beta0[i]<=beta0.band[i,2])
      k1 <- k1+1
  }
  IAC.beta0 <- k1/numFunctPoints
  # beta1(s)
  k2 <- 0
  for(i in 1:numFunctPoints){
    if(beta1[i]>=beta1.band[i,1] & beta1[i]<=beta1.band[i,2])
      k2 <- k2+1
  }
  IAC.beta1 <- k2/numFunctPoints
  # beta2(s)
  k3 <- 0
  for(i in 1:numFunctPoints){
    if(beta2[i]>=beta2.band[i,1] & beta2[i]<=beta2.band[i,2])
      k3 <- k3+1
  }
  IAC.beta2 <- k3/numFunctPoints

  IAC <- c(IAC.beta0, IAC.beta1, IAC.beta2)

  return(list(IAW = IAW, IAC = IAC))
}
```

```r
# calculate
numFunctPoints = 101
beta.true <- data$beta
# naive pffr function
naive.MISE <- beta_MISE(beta.hat=unsp.estim$beta.init, beta.true, numFunctPoints)
naive.CI <- eval.beta.CI(beta.hat=unsp.estim$beta.init, beta.se=unsp.estim$beta.init.se,
                         beta.true=beta.true, numFunctPoints)
naive.IAW <- naive.CI$IAW
print("MISE by naive pffr")
```

```
## [1] "MISE by naive pffr"
```

```r
naive.MISE
```

```
## [1] 0.07041862 0.11833865 0.07400034
```

```r
print("IAW by naive pffr")
```

```
## [1] "IAW by naive pffr"
```

```r
naive.IAW
```

```
## [1] 0.20892250 0.12474696 0.08632818
```

```r
# bootstrap
# bp.MISE <- beta_MISE(beta.hat=unsp.estim$beta.init, beta.true, numFunctPoints)
# bp.CI <- eval.betaBp.CI(beta.lower=bp.estim$beta.band.lower,
#          beta.upper=bp.estim$beta.band.upper, beta.true=beta.true, numFunctPoints)
# bp.IAW <- bp.CI$IAW
# print("MISE by naive pffr")
# bp.MISE
# print("IAW by bootstrap")
# bp.IAW

# fgee with independent covariance
iid.MISE <- beta_MISE(beta.hat=iid.estim$beta, beta.true, numFunctPoints)
iid.CI <- eval.beta.CI(beta.hat=iid.estim$beta, beta.se=iid.estim$beta.se,
                       beta.true=beta.true, numFunctPoints)
iid.IAW <- iid.CI$IAW
print("MISE by fgee with independent covariance")
```

```
## [1] "MISE by fgee with independent covariance"
```

```r
iid.MISE
```

```
## [1] 0.07684909 0.11854721 0.07322243
```

```r
print("IAW by fgee with independent covariance")
```

```
## [1] "IAW by fgee with independent covariance"
```

```r
iid.IAW
```

```
## [1] 0.9159996 0.4112016 0.3022921
```

```r
# fgee with exchangeable covariance
exch.MISE <- beta_MISE(beta.hat=exch.estim$beta, beta.true, numFunctPoints)
exch.CI <- eval.beta.CI(beta.hat=exch.estim$beta, beta.se=exch.estim$beta.se,
                        beta.true=beta.true, numFunctPoints)
```

```r
exch.IAW <- exch.CI$IAW
print("MISE by fgee with exchangeable covariance")
```

```
## [1] "MISE by fgee with exchangeable covariance"
```

```r
exch.MISE
```

```
## [1] 0.08274810 0.11868249 0.07330825
```

```r
print("IAW by fgee with exchangeable covariance")
```

```
## [1] "IAW by fgee with exchangeable covariance"
```

```r
exch.IAW
```

```
## [1] 0.9337658 0.4218025 0.3043496
```

```r
# fgee with unspecified covariance
unsp.MISE <- beta_MISE(beta.hat=unsp.estim$beta, beta.true, numFunctPoints)
unsp.CI <- eval.beta.CI(beta.hat=unsp.estim$beta, beta.se=unsp.estim$beta.se,
                        beta.true=beta.true, numFunctPoints)
unsp.IAW <- unsp.CI$IAW
print("MISE by fgee with unspecified covariance")
```

```
## [1] "MISE by fgee with unspecified covariance"
```

```r
unsp.MISE
```

```
## [1] 0.03694565 0.11378368 0.03097551
```

```r
print("IAW by fgee with unspecified covariance")
```

```
## [1] "IAW by fgee with unspecified covariance"
```

```r
unsp.IAW
```

```
## [1] 0.5569364 0.2176704 0.1678295
```

4. Test covariance structure

4.1 Main functions applied in the hypothesis test

4.2. Results of hypothesis tests

Show examples of covariance test. We test two types of null hypothesis. The first is $H_0 : G_k(\cdot,\cdot)$ is independent, and the second is $H_0 : G_k(\cdot,\cdot)$ is exchangeable. First, we test covariance of data with unspecified covariance structure that is generated in section 2. Obviouly, we would reject the null hypothesis and the p-values would be close to zero. Morevoer, we generate new data with exchangeable covariance (under $H_0 : G_k(\cdot,\cdot)$ is exchangeable) and independent covariance ($H_0 : G_k(\cdot,\cdot)$ is independent). Then, do hypothesis tests on these data. Due to time limit, we didn't run this part.

```r
library(refund)
library(face)
library(fields)
library(mgcv)
library(lme4)
library(MASS)
library(matrixcalc)
library(Matrix)
library(Bolstad)
library(splines)
```

```
#######################################################################
# Test covariance of data with unspecified structure
#######################################################################
dat <- data$data
Y <- dat$Y
Cov <- dat$Cov
numLongiPoints = 41
X1 <- Cov[,1]
X2 <- Cov[,2]
formula <- Y ~ X1 + X2
# initial mean estimates
fit_init <- pffr(formula, yind=dat$funcArg,
                     bs.yindex = list(bs = "ps", k = 10, m = c(2, 2)))
Y.mean.init <- fitted(fit_init)
Y.resid <- as.matrix(Y - Y.mean.init)

# score
fpca_margin <- fpca.face(Y.resid,center = FALSE, argvals=dat$funcArg,
                         knots=20, pve=0.95)
score <- fpca_margin$scores/sqrt(length(dat$funcArg))
score1 <- data.frame(.value=score[,1], .index=dat$Tij, .id=dat$subjID)
#score2 <- data.frame(.value=score[,2], .index=dat$Tij, .id=dat$subjID)

# test structure of G_k(T,T')
test1 <- test.cov.exch(score1, numLongiPoints=numLongiPoints, nbs=1000, nb=10)
#test2 <- test.cov.exch(score2, numLongiPoints=numLongiPoints, nbs=1000, nb=10)
test3 <- test.cov.iid(score1, numLongiPoints=numLongiPoints, nbs=1000, nb=10)
#test4 <- test.cov.iid(score2, numLongiPoints=numLongiPoints, nbs=1000, nb=10)

test1$p
```

```
## [1] 0.015
```

```
#test2$p
test3$p
```

```
## [1] 0.01
```

```
#test4$p

#######################################################################
# Test covariance of data with exchangeable structure
#######################################################################

# generate data with exchangeable covariance from function GenerateData
set.seed(2021)
data.exch <- GenerateData(Nsubj=100, numFunctPoints=101, min_visit=3, max_visit=6,
                         numLongiPoints = 41, sigma_sq = 1.5,
                         sigma_z11=3, sigma_z12=1.5, sigma_z21=2, sigma_z22=1,
                         corstr = "exchangeable")
dat.exch <- data.exch$data
Y <- dat.exch$Y
Cov <- dat.exch$Cov
numLongiPoints = 41
X1 <- Cov[,1]
```

```r
X2 <- Cov[,2]
formula <- Y ~ X1 + X2
# initial mean estimates
fit_init <- pffr(formula, yind=dat.exch$funcArg,
                     bs.yindex = list(bs = "ps", k = 10, m = c(2, 2)))
Y.mean.init <- fitted(fit_init)
Y.resid <- as.matrix(Y - Y.mean.init)

# score
fpca_margin <- fpca.face(Y.resid, center = FALSE, argvals=dat.exch$funcArg,
                         knots=20, pve=0.95)
score <- fpca_margin$scores/sqrt(length(dat.exch$funcArg))
score1 <- data.frame(.value=score[,1], .index=dat.exch$Tij, .id=dat.exch$subjID)
#score2 <- data.frame(.value=score[,2], .index=dat.exch$Tij, .id=dat.exch$subjID)

# test structure of G_k(T,T')
#test5 <- test.cov.exch(score1, numLongiPoints=numLongiPoints, nbs=1000, nb=10)
#test6 <- test.cov.exch(score2, numLongiPoints=numLongiPoints, nbs=1000, nb=10)

#test5$p
#test6$p

######################################################################
# Test covariance of data with independent structure
######################################################################

# generate data with independent covariance from function GenerateData
set.seed(2021)
data.iid <- GenerateData(Nsubj=100, numFunctPoints=101, min_visit=3, max_visit=6,
                         numLongiPoints = 41, sigma_sq=1.5,
                         sigma_z11=3, sigma_z12=1.5, sigma_z21=2, sigma_z22=1,
                         corstr = "independent")
dat.iid <- data.iid$data
Y <- dat.iid$Y
Cov <- dat.iid$Cov
numLongiPoints = 41
X1 <- Cov[,1]
X2 <- Cov[,2]
formula <- Y ~ X1 + X2
# initial mean estimates
fit_init <- pffr(formula, yind=dat.iid$funcArg,
                     bs.yindex = list(bs = "ps", k = 10, m = c(2, 2)))
Y.mean.init <- fitted(fit_init)
Y.resid <- as.matrix(Y - Y.mean.init)

# score
fpca_margin <- fpca.face(Y.resid,center = FALSE, argvals=dat.iid$funcArg,
                         knots=20, pve=0.95)
score <- fpca_margin$scores/sqrt(length(dat.iid$funcArg))
score1 <- data.frame(.value=score[,1], .index=dat.iid$Tij, .id=dat.iid$subjID)
#score2 <- data.frame(.value=score[,2], .index=dat.iid$Tij, .id=dat.iid$subjID)

# test structure of G_k(T,T')
```

```
#test7 <- test.cov.iid(score1, numLongiPoints=numLongiPoints, nbs=1000, nb=10)
#test8 <- test.cov.iid(score2, numLongiPoints=numLongiPoints, nbs=1000, nb=10)

#test7$p
#test8$p
```