

R code file for A Functional Mixed Model for Scalar on Function Regression with Application to a Functional MRI Study

Wanying Ma, Bowen Liu, Luo Xiao, Martin A. Lindquist

10/11/2018

Introduction

In this document, a complete implementation of the proposed Equal-variance test as well as comparison with Bonferroni-corrected test are evaluated using a training dataset, and prediction comparison of FMM and FLM are illustrated using a test dataset. The document is organized as follows. Section 1 gives the main code (fmm_sofr.r) corresponding to the proposed test, Bonferroni-corrected test, FMM prediction and FLM prediction in the paper; in section 2, a training dataset as well as a test dataset are simulated, and is used to test fmm_sofr.r given in section 1; Section 3 visualizes the outputs, including eigendecomposition, testing performance of two methods, estimation comparison of population effects and prediction comparison of response using FMM and FLM.

1. Main code

```
fmm_sofr <- function(data, control = list(pve = 0.95, knots = 7,
  p = 5, m = 3, center = TRUE, lambda = 0)) {

  # input contains data and control both data and control are
  # lists

  Y <- data$Y # response vector Yij
  Z <- data$Z # numeric variable vector Zij
  Xt <- as.matrix(data$Xt) # functional covariate Xij(t) matrix
  ID <- data$ID # subject index

  subs <- unique(ID)
  nSubj <- length(subs)
  nRep <- sapply(subs, function(x) length(which(ID == x)))
  t <- 1:dim(Xt)[2]/dim(Xt)[1] # time grid
  D <- length(t) # time grid length

  library(refund)
  library(lme4)
  library(nlme)
  library(arm)
  library(RLRsim)
  library(MASS)

  #####

  ### fpca on Xt ###
```

```

results <- fpca.face(Xt, center = control$center, argvals = t,
  knots = control$knots, pve = control$pve, p = control$p,
  lambda = control$lambda)
npc <- results$npc
score <- results$scores
ascore <- score[, 1:npc]/sqrt(D)
efunctions <- results$efunctions * sqrt(D)
evalues <- results$evalues/D

#####

### data for bonferroni testing and FLM/FMM regression ###

designMatrix.reg <- data.frame(Y = Y, Z = Z, ID = as.factor(ID),
  ascore = ascore)

#####

### equal variance testing ###
z.sim.uni = c()
ID.uni <- c()
index <- matrix(1:(nSubj * npc), nrow = npc, ncol = nSubj)
for (i in 1:length(nRep)) {
  ID.uni = c(ID.uni, c(index[, i], rep(0, nRep[i] - npc)))
}
z.sim.uni = c()
# svd on random scores A_i for each subject i
for (k in 1:nSubj) {
  if (k == 1) {
    svd <- svd(ascore[1:nRep[1], ] %*% t(ascore[1:nRep[1],
      ])) # SVD on A_i
  } else {
    svd <- svd(ascore[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k]),
      ] %*% t(ascore[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k]),
        ])) #SVD on A_i
  }
  u.tra <- t(svd$v)
  u <- svd$u
  d <- (svd$d)[1:npc]
  if (k == 1) {
    Y[1:nRep[k]] <- u.tra %*% Y[1:nRep[k]]
    Z[1:nRep[k]] <- u.tra %*% Z[1:nRep[k]]
    ascore[1:nRep[k], ] <- rbind(u.tra[1:npc, ] %*% ascore[1:nRep[k],
      ], matrix(0, nrow = nRep[k] - npc, ncol = npc))
  } else {
    Y[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k])] <- u.tra %*%
      Y[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k])]
    Z[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k])] <- u.tra %*%
      Z[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k])]
    ascore[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k]),
      ] <- rbind(u.tra[1:npc, ] %*% ascore[(sum(nRep[1:(k -
        1)]) + 1):sum(nRep[1:k]), ], matrix(0, nrow = nRep[k] -
        npc, ncol = npc))
  }
}

```

```

    }
    # z.sim.uni is the coefficient for the random slope to be
    # tested
    z.sim.uni <- c(z.sim.uni, sqrt(d), rep(0, nRep[k] - npc))
  }

#####

### data for equal-variance testing ###

designMatrix <- data.frame(Y = Y, Z = Z, ID = as.factor(ID),
  ID.uni = as.factor(ID.uni), ascore = ascore, z.sim.uni = z.sim.uni)

#####

## equal-variance test ##
additive0.sim <- paste(1:npc, collapse = " + ascore.")
model.sim <- as.formula(paste("Y ~ 1 + Z + ascore.", additive0.sim,
  " + (0 + Z | ID) + (0 + z.sim.uni | ID.uni)", sep = ""))
# fullReml is the model under alternative
fullReml <- lmer(model.sim, data = designMatrix)
# m.slope only contains the random effect to be tested
f.slope <- as.formula(paste("Y ~ 1 + Z + ascore.", additive0.sim,
  " + (0 + z.sim.uni | ID.uni)", sep = ""))
m.slope <- lmer(f.slope, data = designMatrix)
# m0 is the model under the null
f0 <- as.formula(" . ~ . - (0 + z.sim.uni | ID.uni)")
m0 <- update(fullReml, f0)

EqualVar.test <- exactRLRT(m.slope, fullReml, m0)
EqualVar.pvalue <- EqualVar.test$p[1]

## bonferroni test ##
additive.heter <- paste0(" + (0 + ascore.", 1:npc, " | ID)",
  collapse = "")
bonf.test <- list()
for (i in 1:npc) {
  ii <- paste("ascore.", i, sep = "")
  # f.slope only contains the random effect to be tested
  f.slope <- as.formula(paste("Y ~ 1 + Z + ascore.", additive0.sim,
    " + (0 +", ii, " | ID)", sep = ""))
  m.slope <- lmer(f.slope, data = designMatrix.reg)
  # mA is the model under alternative
  mA <- as.formula(paste("Y ~ 1 + Z + ascore.", additive0.sim,
    " + (0 +", ii, " | ID)", "+ (0 + Z | ID)", sep = ""))
  fullReml <- lmer(mA, data = designMatrix.reg)
  # m0 is model under the null
  f0 <- as.formula(paste(" . ~ . - (0 +", ii, " | ID)"))
  m0 <- update(fullReml, f0)
  bonf.test[[i]] <- exactRLRT(m.slope, fullReml, m0)
}
multiTest <- sapply(bonf.test, function(x) {
  c(statistic = x$statistic[1], `p-value` = x$p[1])
})

```

```

})
# use bonferroni correctiong method to adjust p-value
bonf.pvalue <- p.adjust(multiTest[2, ], "bonferroni")

#####

## estimation ##

# FLM: estimation without subject-specific random effect
noRandom.simpert <- paste(1:npc, collapse = " + ascore.")
noRandom.sim <- as.formula(paste("Y ~ 1 + Z + ascore.", noRandom.simpert,
  " + (0 + Z | ID)", sep = ""))
FLM <- lmer(noRandom.sim, data = designMatrix.reg)

# FMM: estimation with subject-specific random effect
Random.simpert <- paste0(" + (0 + ascore.", 1:npc, "|ID) ",
  collapse = "")
Random.sim <- as.formula(paste("Y ~ 1 + Z + ascore.", noRandom.simpert,
  " + (0 + Z | ID) ", Random.simpert, sep = ""))
FMM <- lmer(Random.sim, data = designMatrix.reg)

#####

## get fixed effect beta(t) ##

# FLM
fixeff1 <- fixef(FLM)
beta1 <- efunctions %*% as.vector(fixeff1[match(paste0("ascore.",
  1:npc), names(fixeff1))]) # population effect beta(t)
ranef1 <- ranef(FLM)$ID
FLM.sum <- summary(FLM)
fixed.coeff1 <- FLM.sum$coefficients # fixed coefficient
fixed.vcov1 <- FLM.sum$vcov # coefficient covariance
# sigma is standard deviation for error term
sigma1 <- FLM.sum$sigma
# se is standard deviation for population effect beta(t)
se1 = apply(efunctions, 1, function(x) sqrt(x %*% as.matrix(fixed.vcov1[match(paste0("ascore.",
  1:npc), names(fixeff1)), match(paste0("ascore.", 1:npc),
  names(fixeff1))]) %*% x))
yhat1 <- predict(FLM)

# FMM
fixeff2 <- fixef(FMM)
beta2 <- efunctions %*% as.vector(fixeff2[match(paste0("ascore.",
  1:npc), names(fixeff2))]) # population effect beta(t)
ranef2 <- ranef(FMM)$ID
betai_2 <- efunctions %*% t(as.matrix(ranef2[, match(paste0("ascore.",
  1:npc), colnames(ranef2))])) # beta_i(t): subject deviation from population effect beta(t)
FMM.sum <- summary(FMM)
fixed.coeff2 <- FMM.sum$coefficients # fixed coefficient
fixed.vcov2 <- FMM.sum$vcov
sigma2 <- FMM.sum$sigma # error term se
# se is standard deviation for population effect

```

```

se2 = apply(efunctions, 1, function(x) sqrt(x %*% as.matrix(fixed.vcov2[match(paste0("ascore.",
1:npc), names(fixeff2)), match(paste0("ascore.", 1:npc),
names(fixeff2))]) %*% x))
yhat2 <- predict(FMM)

#####

return(list(fPCA_result = list(npc = npc, ascore = ascore,
efunctions = efunctions, evalues = evalues), test_result = list(`equal-variance` = list(EqualVar.
EqualVar.pvalue = EqualVar.pvalue), bonferroni = list(bonf.test = bonf.test,
bonf.pvalue = bonf.pvalue)), estimation_result = list(FLM = list(fixed = list(`beta(t)` = betai_1,
coefficient = fixed.coeff1, vcov = fixed.vcov1, sigma = sigma1,
`se_beta(t)` = se1), random = list(Zi = raneffect1),
yhat = yhat1, FLM.fit = FLM), FMM = list(fixed = list(`beta(t)` = beta2,
coefficient = fixed.coeff2, vcov = fixed.vcov2, sigma = sigma2,
`se_beta(t)` = se2), random = list(Zi = raneffect2[,
1], `beta_i(t)` = betai_2), yhat = yhat2, FMM.fit = FMM))))
}

```

2. Simulate a sample dataset (20 subjects, 50 visits for each subject as a training dataset; generate 10 more new visits for each of 20 subjects as a test dataset), and test function fmm_sofr.r

2.1 Simulate a training dataset

```

library(refund)
library(lme4)
library(nlme)
library(arm)
library(RLRSim)
library(MASS)

#####

### generate a training dataset ###

nSubj <- 20
nRep <- 50
smooth = 0 # if 0, Xt is with measurement error and thus unsmoothed; if smooth=1, no measurement error
heter = 1 # if 1, random scores are heterogeneous; if 0, random scores are homogeneous
r.sim = 0.08
set.seed(12345)

D <- 80 # time grid number
totalN <- nSubj * nRep
thetaK.true <- 2
timeGrid <- (1:D)/D

npc.true <- 3
SNR <- 3 # 5, signal noise ratio

```

```

sd.epsilon <- 1 # or 0.5
delta.true <- 0.5
a.mean <- 0

gamma.true <- 2
gammaVar.true <- 1
# generate random slope of dummy variable
gammaI.true.i <- mapply(rnorm, nSubj, gamma.true, rep(sqrt(gammaVar.true),
1))
gammaI.true <- gammaI.true.i[rep(1:nrow(gammaI.true.i), each = nRep),
]

dummyX <- rbinom(n = totalN, size = 1, prob = 0.5) # dummyX: Z

# lambda is variance of each random score
lambda.sim <- function(degree) {
  return(0.5^(degree - 1))
}
# eigen functions
psi.fourier <- function(t, degree) {
  result <- NA
  if (degree == 1) {
    result <- sqrt(2) * sinpi(2 * t)
  } else if (degree == 2) {
    result <- sqrt(2) * cospi(4 * t)
  } else if (degree == 3) {
    result <- sqrt(2) * sinpi(4 * t)
  }
  return(result)
}

lambdaVec.true <- mapply(lambda.sim, 1:npc.true)

# true eigen-functions
psi.true <- matrix(data = mapply(psi.fourier, rep(timeGrid, npc.true),
  rep(1:npc.true, each = D)), nrow = npc.true, ncol = D, byrow = TRUE)

# generate functional covariates
ascore.true <- mvrnorm(totalN, rep(a.mean, npc.true), diag(lambdaVec.true))
Mt.true <- ascore.true %*% psi.true
error <- rnorm(totalN, mean = 0, sd = sd.epsilon)

if (heter == 0) {
  thetaIK.true1 <- mvrnorm(nSubj, rep(thetaK.true, npc.true),
    diag(rep(r.sim, npc.true)))
}
if (heter == 1) {
  thetaIK.true1 <- mvrnorm(nSubj, rep(0, npc.true), diag(c(r.sim,
    r.sim/2, r.sim/4)))
}

thetaIK.true <- thetaIK.true1[rep(1:nrow(thetaIK.true1), each = nRep),
]

```

```

betaM.true <- (thetaK.true + thetaIK.true) * ascore.true
betaM.true <- rowSums(betaM.true)

Y <- delta.true + dummyX * gammaI.true + betaM.true + error
ID <- rep(1:nSubj, each = nRep)

if (smooth == 0) {
  Merror.Var <- sum(lambdaVec.true)/SNR #SNR = sum(lambdaVec.true)/Merror.Var
  Mt.hat <- Mt.true + matrix(rnorm(totalN * D, mean = 0, sd = sqrt(Merror.Var)),
    totalN, D)
}
if (smooth == 1) {
  Merror.Var <- 0 #SNR = sum(lambdaVec.true)/Merror.Var
  Mt.hat <- Mt.true
}

M <- Mt.hat
traindata <- list(Y = Y, Z = dummyX, Xt = M, ID = ID)

```

2.2 Perform two testing procedures and estimation using FLM and FMM on the training dataset

```

#####

result <- fmm_sofr(traindata)

#####

```

2.3 Generate a test dataset

```

### generate a test dataset ###

### for each subject, generate newvisitN more new visits
gene_newdata <- function(newvisitN) {
  # generate new Z
  dummyX <- rbinom(n = nSubj * newvisitN, size = 1, prob = 0.5) # new dummyX: Z
  # generate new x_ij(t) for new visit
  ascore.true <- mvrnorm(nSubj * newvisitN, rep(a.mean, npc.true),
    diag(lambdaVec.true))
  # sum new x_ij(t)(beta_t + betai_t)
  Mt.true <- ascore.true %*% psi.true
  # generate new error
  error <- rnorm(nSubj * newvisitN, mean = 0, sd = sd.epsilon)
  # generate new random scores
  thetaIK.true <- thetaIK.true1[rep(1:nrow(thetaIK.true1),
    each = newvisitN), ]
  betaM.true <- (thetaIK.true + thetaK.true) * ascore.true
  betaM.true <- rowSums(betaM.true)

  gammaI.true <- gammaI.true.i[rep(1:nrow(gammaI.true.i), each = newvisitN),

```

```

]

# generate new Y
Y <- delta.true + dummyX * gammaI.true + betaM.true + error

totalN <- nSubj * newvisitN
ID <- rep(1:nSubj, each = newvisitN)

if (smooth == 0) {
  Merror.Var <- sum(lambdaVec.true)/SNR #SNR = sum(lambdaVec.true)/Merror.Var
  Mt.hat <- Mt.true + matrix(rnorm(totalN * D, mean = 0,
    sd = sqrt(Merror.Var)), totalN, D)
}
if (smooth == 1) {
  Merror.Var <- 0
  Mt.hat <- Mt.true
}

M <- Mt.hat
# projection to get new xi_{ijk}
efunctions <- result$fPCA_result$efunctions
xi <- (M - as.vector(rep(1, dim(M)[1])) %*% t(colMeans(M))) %*%
  efunctions/D

designMatrix <- data.frame(Y = Y, Z = dummyX, ID = as.factor(ID),
  ascore = xi)
return(list(newY = Y, newdata = designMatrix, Xt = M))
}

# generate new testdata
newvisit <- 10
testdata <- gene_newdata(newvisit)

```

2.4 Etimation using FLM and FMM on the testing dataset

```

# yhat: predict for the new testdata
test_FLM.yhat <- predict(result$estimation_result$FLM$FLM.fit,
  testdata$newdata)
test_FMM.yhat <- predict(result$estimation_result$FMM$FMM.fit,
  testdata$newdata)

```

3. Visualize results

3.1 Visualize estimated eigenfunctions vs true eigenfunctions

```

### FPCA ###
efunctions <- result$fPCA_result$efunctions
npc <- result$fPCA_result$npc
# correct for sign of eigen-functions

```


True VS estimated eigenfunction 1

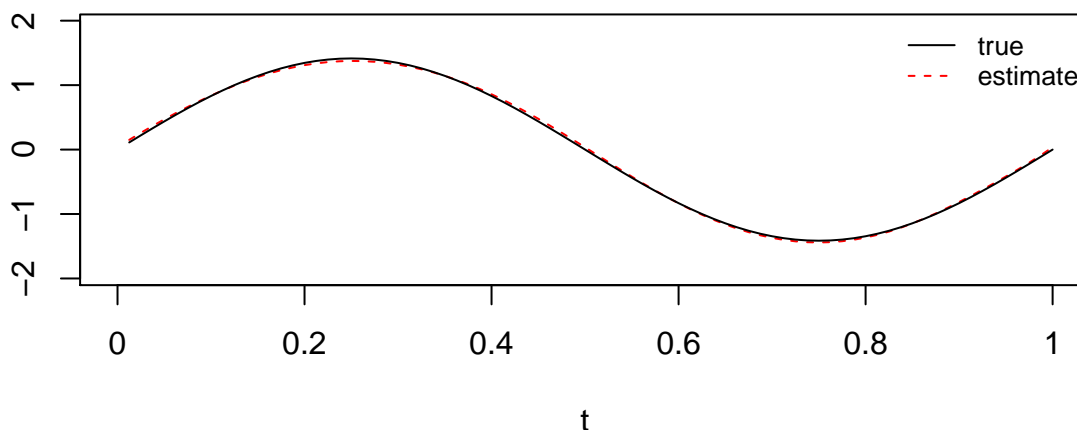


Figure 1: output

True VS estimated eigenfunction 2

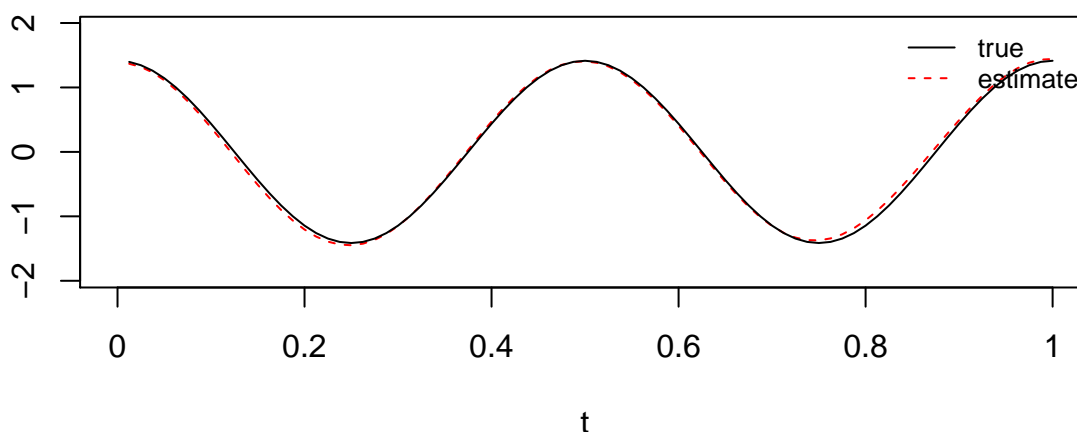


Figure 2: output

```
sign_correct <- diag(sign(colSums(efunctions * t(psi.true))))
efunctions.correct <- efunctions %*% sign_correct
myat <- seq(0, 1, by = 0.2)
for (i in 1:npc) {
  plot(timeGrid, efunctions.correct[, i], type = "l", xlab = "t",
       ylab = "", xaxt = "n", xlim = c(0, 1), main = paste0("True VS estimated eigenfunction ",
       i), lwd = 1, col = "red", lty = 2, ylim = c(min(efunctions.correct) -
       0.5, max(efunctions.correct) + 0.5))
  axis(1, at = myat, labels = round(myat, digits = 1))
  lines(timeGrid, psi.true[i, ], col = "black")
  legend("topright", c("true", "estimate"), col = c("black",
  "red"), lty = c(1, 2), bty = "n", cex = 0.8)
}
```

True VS estimated eigenfunction 3

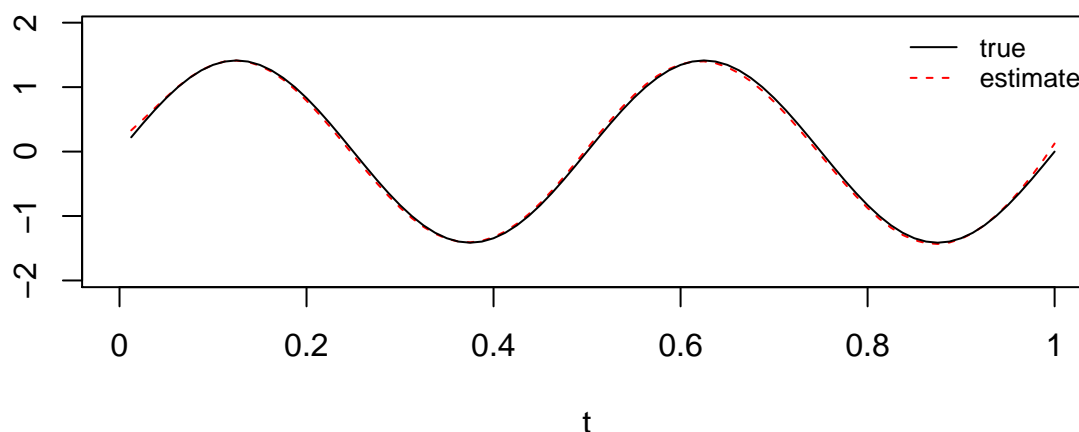


Figure 3: output

3.2 Two testing results

```
# equal-variance test result
result$test_result$`equal-variance`
```

```
# $EqualVar.test
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 82.399, p-value < 2.2e-16
#
#
# $EqualVar.pvalue
# [1] 0
```

```
# bonferroni test result
result$test_result$bonferroni
```

```
# $bonf.test
# $bonf.test[[1]]
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 85.811, p-value < 2.2e-16
#
#
# $bonf.test[[2]]
#
#   simulated finite sample distribution of RLRT.
```

```

#
# (p-value based on 10000 simulated values)
#
# data:
# RLRT = 2.9496, p-value = 0.0364
#
#
# $bonf.test[[3]]
#
# simulated finite sample distribution of RLRT.
#
# (p-value based on 10000 simulated values)
#
# data:
# RLRT = 2.0929, p-value = 0.0613
#
#
# $bonf.pvalue
# [1] 0.0000 0.1092 0.1839

```

3.3 Visualize population effect estimation using FLM VS FMM

```

# true beta(t): population effect
beta_t <- t(psi.true) %*% as.vector(rep(thetaK.true, npc.true))
beta_t.FMM <- result$estimation_result$FMM$fixed$`beta(t)`
beta_t.FLM <- result$estimation_result$FLM$fixed$`beta(t)`

sprintf("MISE of estimated beta(t) using FMM is: %.3f", mean((beta_t.FMM -
  beta_t)^2))

# [1] "MISE of estimated beta(t) using FMM is: 0.005"

sprintf("MISE of estimated beta(t) using FLM is: %.3f", mean((beta_t.FLM -
  beta_t)^2))

# [1] "MISE of estimated beta(t) using FLM is: 0.007"

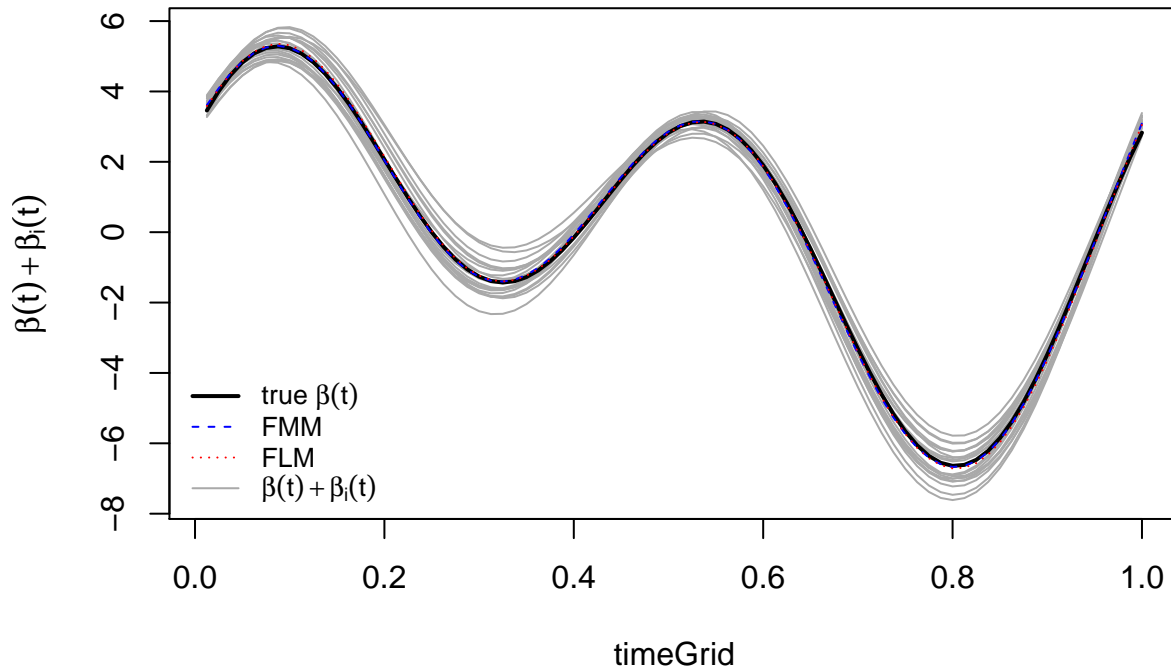
betait.FMM <- result$estimation_result$FMM$random$`beta_i(t)`
beta.comp <- cbind(apply(betait.FMM, 2, function(x) x + beta_t.FMM),
  beta_t, beta_t.FMM, beta_t.FLM)
colnames(beta.comp) <- c(paste0("Subject", 1:nSubj), "true_Fixed_effect",
  "Fixed_effect.FMM", "Fixed_effect.FLM")

# visualization
matplot(timeGrid, beta.comp[, 1:nSubj], col = "darkgrey", type = "l",
  lty = 1, main = expression(paste("Comparison of population fixed effect ",
  beta(t))), ylab = expression(beta(t) + beta[i](t)))
lines(timeGrid, beta.comp[, nSubj + 1], col = "black", lty = 1,
  lwd = 2)
lines(timeGrid, beta.comp[, nSubj + 2], col = "blue", lty = 2)
lines(timeGrid, beta.comp[, nSubj + 3], col = "red", lty = 3)
# lines(timeGrid, beta.comp[, 53] +
# 1.96*result$estimation_result$FMM$fixed$`se_beta(t)` ,

```

```
# col='blue',lty=3) lines(timeGrid, beta.comp[,53] -
# 1.96*result$estimation_result$FMM$fixed$`se_beta(t)`,
# col='blue',lty=3)
legend("bottomleft", c(expression(paste("true ", beta(t))), "FMM",
"FLM", expression(beta(t) + beta[i](t))), col = c("black",
"blue", "red", "darkgrey"), lty = c(1, 2, 3, 1), lwd = c(2,
1, 1, 1), bty = "n", cex = 0.8)
```

Comparison of population fixed effect $\beta(t)$



3.4 Visualize MSE for each subject

```
#####

### violin plot of MSE for each subject in the new testdata ###

# MSE for each subject
ID <- testdata$newdata$ID
subs <- unique(ID)
nSubj <- length(subs)
calculateMSE_i <- function(Y, EST) {
  out <- (Y - EST)^2
  mse <- rep(0, length(unique(ID)))
  for (i in c(1:nSubj)) {
    mse[i] <- mean(out[which(ID == i)])
  }
  return(mse)
}
```

```

# MSE for each subject
MSE_FMM <- calculateMSE_i(testdata$newY, test_FMM.yhat)
MSE_FLM <- calculateMSE_i(testdata$newY, test_FLM.yhat)
sprintf("MSE of FMM is: %.3f", mean(MSE_FMM))

# [1] "MSE of FMM is: 1.209"

sprintf("MSE of FLM is: %.3f", mean(MSE_FLM))

# [1] "MSE of FLM is: 1.483"

# library(MuMin)
rsquare_FMM <- MuMin::r.squaredGLMM(result$estimation_result$FMM$FMM.fit)
rsquare_FLM <- MuMin::r.squaredGLMM(result$estimation_result$FLM$FLM.fit)
sprintf("R-square of FMM is: %.3f", rsquare_FMM[, 2])

# [1] "R-square of FMM is: 0.888"

sprintf("R-square of FLM is: %.3f", rsquare_FLM[, 2])

# [1] "R-square of FLM is: 0.871"

violin <- data.frame(cbind(c(MSE_FLM, MSE_FMM), rep(1:2, each = nSubj)))
colnames(violin) <- c("MSE", "Model")
violin$Model <- as.factor(violin$Model)

library(ggplot2)
ggplot(violin, aes(x = Model, y = MSE)) + geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.1, outlier.alpha = 0.1) + geom_jitter(width = 0.2,
    alpha = 0.6) + geom_hline(aes(yintercept = median(MSE_FMM)),
    color = "red", linetype = "dashed", size = 0.8) + scale_x_discrete(breaks = c("1",
    "2"), labels = c("FLM", "FMM")) + labs(title = "MSE comparison of FLM and FMM (out-of-sample)") +
  theme_classic() + theme(plot.title = element_text(hjust = 0.5,
    size = rel(0.9), face = "bold")) + theme(axis.title.y = element_text(size = rel(0.8))) +
  theme(axis.title.x = element_text(size = rel(0.8)))

```

MSE comparison of FLM and FMM (out-of-sample)

