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

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

05/30/2019

Introduction

In this document, a complete implementation of the proposed Equal-variance test as well as comparison with Bonferroni-corrected test, asLRT are evaluated using a training dataset (20 subjects, 50 visits), and estimation/prediction comparison of FMM and FLM are illustrated using a test dataset (10 new visits for each of 20 subjects). In addition, analysis using the multivariate functional predictor is also implemented. The document is organized as follows. Section 1 gives the main code (fmm_sofr.r) corresponding to the proposed test, Bonferroni-corrected test, asLRT, FMM estimation/prediction and FLM estimation/prediction for both univariate and multivariate functional predictor in the paper; In section 2, both a training dataset and 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, response of the new visits, estimation comparison of population effects and out-of-sample prediction comparison of response using FMM and FLM; Section 4 repeats the analysis process in section 2 and 3 for the multivariate functional predictor.

1. Main code

```
installed.packages(c("refund", "lme4", "nlme", "arm", "RLRsim",
    "MASS", "MuMIn", "ggplot2"))
```

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

```
FisherInfoInv <- function(fit, data) {
    ## this function calculates Fisher information matrix used for
    ## asLRT
    subj <- unique(data$ID)
    nsubj <- length(subj)
    npc <- (ncol(data) - 3)/2
    var.est <- as.data.frame(VarCorr(fit))[, 4]
    tau0 <- var.est[1]
    tau1 <- var.est[2]
    sigma2 <- var.est[3]
    FI <- matrix(0, npc + 3, npc + 3)

    for (i in 1:nsubj) {
        sel <- which(data$ID == subj[i])
        ni <- length(sel)
        datai <- data[sel, ]
        Sigmai <- sigma2 * diag(ni) + tau0 * datai$Z %x% t(datai$Z) +
```

```
            tau1 * rep(1, ni) %*% t(rep(1, ni))
        Sigmai_inv <- solve(Sigmai)
        for (j in 1:(npc + 1)) {
            for (k in 1:(npc + 1)) {
                FI[j, k] <- FI[j, k] + 1/2 * (sum(datai[, j +
                  2] * (Sigmai_inv %*% datai[, k + 2])))^2
            }
        }
        for (j in 1:(npc + 1)) {
            FI[npc + 2, j] <- FI[npc + 2, j] + 1/2 * sum((Sigmai_inv %*%
                datai[, j + 2])^2)
            FI[j, npc + 2] <- FI[npc + 2, j]
        }
        FI[npc + 2, npc + 2] <- FI[npc + 2, npc + 2] + 1/2 *
            sum(Sigmai_inv^2)

        ## for subject-specific random intercept
        for (j in 1:(npc + 1)) {
            FI[j, npc + 3] <- FI[j, npc + 3] + 1/2 * (sum(datai[,
                j + 2] * (Sigmai_inv %*% rep(1, ni))))^2
            FI[npc + 3, j] <- FI[j, npc + 3]
        }
        FI[npc + 2, npc + 3] <- FI[npc + 2, npc + 3] + 1/2 *
            sum((Sigmai_inv %*% rep(1, ni))^2)
        FI[npc + 3, npc + 2] <- FI[npc + 2, npc + 3]
        FI[npc + 3, npc + 3] <- FI[npc + 3, npc + 3] + 1/2 *
            (sum(rep(1, ni) * (Sigmai_inv %*% rep(1, ni))))^2

    }
    return(solve(FI)[1:npc, 1:npc])
}

Chi_bar_square <- function(lrt, R, alpha) {
    # this function evaluates the chi-bar asymptotic dist. for
    # asLRT
    npc <- nrow(R)
    RC <- diag(sqrt(1/diag(R))) %*% R %*% diag(sqrt(1/diag(R)))
    if (npc == 3) {

        RPC <- rep(0, 3)
        RPC[1] <- (RC[2, 3] - RC[3, 1] * RC[2, 1])/sqrt(1 - RC[3,
            1]^2)/sqrt(1 - RC[3, 1]^2)
        RPC[2] <- (RC[1, 3] - RC[1, 2] * RC[3, 2])/sqrt(1 - RC[1,
            2]^2)/sqrt(1 - RC[3, 2]^2)
        RPC[3] <- (RC[1, 2] - RC[1, 3] * RC[2, 3])/sqrt(1 - RC[1,
            3]^2)/sqrt(1 - RC[2, 3]^2)
        w <- rep(0, 4)
        w[4] <- 1/(4 * pi) * (2 * pi - acos(RC[1, 2]) - acos(RC[1,
            3]) - acos(RC[2, 3]))
        w[3] <- 1/(4 * pi) * (3 * pi - acos(RPC[3]) - acos(RPC[2]) -
            acos(RPC[1]))
        w[2] <- 1/2 - w[4]
        w[1] <- 1/2 - w[3]
```

```
    }

    if (npc == 2) {
        w <- rep(0, 3)
        w[1] <- 1/(2 * pi) * acos(RC[1, 2])
        w[2] <- 1/2
        w[3] <- 1/2 - w[1]
    }

    if (npc == 1) {
        w <- c(1/2, 1/2)
    }

    obj <- function(x) {
        prob <- 0
        for (i in 0:npc) {
            prob <- prob + w[i + 1] * pchisq(x, df = i)
        }
        return((prob - (1 - alpha))^2)
    }
    cutoff <- nlminb(w[1], obj)$par

    pvalue <- 0
    for (i in 1:npc) {
        pvalue <- pvalue + w[i + 1] * (1 - pchisq(lrt, df = i))
    }

    return(list(observed = lrt, cutoff = cutoff, pvalue = pvalue))
}

mfpca.face <- function(data, newdata = NULL, pve = 0.95, knots = 10,
    lambda = 0, center = T) {
    ## this is multivariate face for multivariate functional data

    require(refund)
    FPCA <- list()
    J <- length(data)
    for (j in 1:J) {
        FPCA[[j]] <- fpca.face(data[[j]], knots = knots, argvals = (1:ncol(data[[j]]))/ncol(data[[j]]),
            pve = pve, lambda = lambda, center = center)
    }

    ## first recover covariance functions
    C_list <- vector("list", J)


    for (j1 in 1:J) {
        for (j2 in 1:J) {
            if (j1 == j2)
                temp <- diag(FPCA[[j1]]$evalues)
            if (j1 != j2)
                temp <- cov(FPCA[[j1]]$scores, FPCA[[j2]]$scores)
            C_list[[j1]][[j2]] <- FPCA[[j1]]$efunctions %*% temp %*%
```

```
            t(FPCA[[j2]]$efunctions)
    }
}


## MFPCA
c <- unlist(lapply(data, function(x) {
    dim(x)[2]
}))
c_all <- sum(c)
Cs <- matrix(NA, c_all, c_all)
for (j1 in 1:J) {
    for (j2 in 1:J) {
        if (j1 == 1)
            sel1 <- 1:c[1]
        if (j1 > 1)
            sel1 <- sum(c[1:(j1 - 1)]) + (1:c[j1])
        if (j2 == 1)
            sel2 <- 1:c[1]
        if (j2 > 1)
            sel2 <- sum(c[1:(j2 - 1)]) + (1:c[j2])

        Cs[sel1, sel2] <- C_list[[j1]][[j2]]
    }
}
Eigen <- eigen(Cs)
d <- which.max(cumsum(Eigen$values)/sum(Eigen$values) >=
    pve)
U_joint <- Eigen$vectors[, 1:d]
lambda <- Eigen$values[1:d]


## for prediction of newdata
if (is.null(newdata))
    newdata = data
n_new <- dim(newdata[[1]])[1]
m <- lapply(newdata, function(x) {
    dim(x)[2]
})

mu_newdata <- list()
c_newdata <- list()
sc_newdata <- list()
for (j in 1:J) {
    mu_newdata[[j]] <- matrix(FPCA[[j]]$mu, nrow = n_new,
        ncol = m[[j]], byrow = TRUE)
    c_newdata[[j]] <- newdata[[j]] - mu_newdata[[j]]
    sc_newdata[[j]] <- c_newdata[[j]] %*% FPCA[[j]]$efunctions %*%
        t(FPCA[[j]]$efunctions)
}


scores <- do.call(cbind, sc_newdata) %*% U_joint
```

```
    sc_newdata_pred <- scores %*% t(U_joint)



    # prediction by MFPCA
    newdata_pred <- list()
    for (j1 in 1:J) {
        if (j1 == 1)
            sel <- 1:c[1]
        if (j1 > 1)
            sel <- sum(c[1:(j1 - 1)]) + (1:c[j1])
        newdata_pred[[j1]] <- sc_newdata_pred[, sel] + mu_newdata[[j1]]
    }

    ## prediction via BLUP

    sig2_list <- lapply(FPCA, function(x) {
        mean((t(x$Yhat - x$Y) - x$mu)^2)
    })
    for (j in 1:J) {

        if (j == 1)
            sel <- 1:c[1]
        if (j > 1)
            sel <- sum(c[1:(j - 1)]) + (1:c[j])

        Cs[sel, sel] <- Cs[sel, sel] + diag(length(sel)) * sig2_list[[j]]
    }

    Lambda <- diag(lambda)
    scores <- Lambda %*% t(U_joint) %*% solve(Cs) %*% t(do.call(cbind,
        c_newdata))
    scores <- t(scores)
    newdata_predu <- scores %*% t(U_joint)

    newdata_pred_blup <- list()
    for (j in 1:J) {
        if (j == 1)
            sel <- 1:c[1]
        if (j > 1)
            sel <- sum(c[1:(j - 1)]) + (1:c[j])
        newdata_pred_blup[[j]] <- mu_newdata[[j]] + newdata_predu[,
            sel]

    }

    res <- list(data = data, FPCA = FPCA, J = J, C_list = C_list,
        efunctions = U_joint, evalues = Eigen$values[1:d], npc = d,
        scores = scores, newdata_pred = newdata_pred, newdata_pred_blup = newdata_pred_blup,
        c = c, c_all = c_all, scores = scores)
    class(res) <- "mfpca.face"
    return(res)
}
```

```r
fmm_sofr <- function(data, multi = 1, 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 multi denotes the number of multivariate functional
    # predictor; an integer needs to be specified for multi; by
    # default, multi = 1

    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]/multi)/(dim(Xt)[2]/multi)  # time grid
    D <- length(t)  # time grid length

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

    ###############################################################################

    if (multi > 1) {
        ### mfpca on Xt, here Xt needs to be reformulated into a list
        ### ###
        Xt_list <- list()
        for (i in 1:multi) {
            Xt_list[[i]] <- as.matrix(Xt[, ((i - 1) * D + 1):(i *
                D)])
        }

        results <- mfpca.face(Xt_list, center = control$center,
            knots = 10, pve = control$pve, lambda = control$lambda)

    } else {
        ### 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
bscore <- ascore * Z  #interaction term

##############################################################################

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

## note by Luo: the design matrix has to be of this order for
## the (asymptotic) LRT

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

##############################################################################

### equal variance testing ###
z.sim.uni = c()
ID.uni <- c()
cscore <- 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]
    cscore <- c(cscore, rowSums(u.tra))
    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))
        bscore[1:nRep[k], ] <- u.tra %*% bscore[1:nRep[k],
            ]

    } 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))
        bscore[(sum(nRep[1:(k - 1)]) + 1):sum(nRep[1:k]),
            ] <- u.tra %*% bscore[(sum(nRep[1:(k - 1)]) +
            1):sum(nRep[1:k]), ]
    }
    # 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,
    bscore = bscore, cscore = cscore)

################################################################################

## equal-variance test ##
additive0.sim <- paste(1:npc, collapse = " + ascore.")
additive.sim.b <- paste(1:npc, collapse = " + bscore.")

model.sim <- as.formula(paste("Y ~ 1 + Z + ascore.", additive0.sim,
    " + bscore.", additive.sim.b, " + (0 + cscore | ID)",
    " + (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,
    " + bscore.", additive.sim.b, " + (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]
## end of equal-variance test

## 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,
        " + bscore.", additive.sim.b, " + (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,
        " + bscore.", additive.sim.b, " + (0 +", ii, " | ID)",
        " + (1 | 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")
## end of bonferroni test

## LRT (added by Luo Apr 4, 2019)
additive0.sim <- paste(1:npc, collapse = " + ascore.")
fixed <- paste("Y ~ 1 + Z + ascore.", additive0.sim, " + bscore.",
    additive.sim.b, sep = "")

scores <- sapply(sapply(1:npc, function(x) {
    paste("ascore.", x, sep = "")
}), function(x) {
    paste("(0+", x, "|ID)", sep = "")
})
scores <- c(scores)
random <- paste(scores, collapse = " + ")
random <- paste("(0 + Z|ID) + ", random, sep = "")
random <- paste0(random, " + (1 | ID)")


# fullReml is the model under alternative
model.sim <- as.formula(paste(fixed, "+", random, sep = ""))
fit.full <- lmer(model.sim, data = designMatrix.reg, REML = FALSE)

model.sim.null <- as.formula(paste(fixed, "+ (0+Z|ID)", " + (1 | ID)",
    sep = ""))
fit.null <- lmer(model.sim.null, data = designMatrix.reg,
    REML = FALSE)

lrt <- -2 * (as.numeric(logLik(fit.null)) - as.numeric(logLik(fit.full)))


R <- FisherInfoInv(fit.null, designMatrix.reg)
LRT.test <- Chi_bar_square(lrt, R, 0.05)
LRT.pvalue <- LRT.test$pvalue
## end of LRT

################################################################################
```

```
## estimation ##

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

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

## get fixed effect beta(t) ##

# FLM
fixeff1 <- fixef(FLM)
beta1_a <- efunctions %*% as.vector(fixeff1[grep("^ascore",
    names(fixeff1))])  # population effect beta(t)
beta1_b <- efunctions %*% as.vector(fixeff1[grep("^bscore",
    names(fixeff1))])  # delta(t)
raneffect1 <- 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_a is standard deviation for population effect beta(t)
se1_a = 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))
# se_b is standard deviation for delta(t)
se1_b = apply(efunctions, 1, function(x) sqrt(x %*% as.matrix(fixed.vcov1[match(paste0("bscore.",
    1:npc), names(fixeff1)), match(paste0("bscore.", 1:npc),
    names(fixeff1))]) %*% x))

yhat1 <- predict(FLM)

# FMM
fixeff2 <- fixef(FMM)
beta2_a <- efunctions %*% as.vector(fixeff2[grep("^ascore",
    names(fixeff2))])  # population effect beta(t)
beta2_b <- efunctions %*% as.vector(fixeff2[grep("^bscore",
    names(fixeff2))])  # population effect delta(t)
raneffect2 <- ranef(FMM)$ID
betai_2 <- efunctions %*% t(as.matrix(raneffect2[grep("^ascore",
    names(raneffect2))]))  # 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_a is standard deviation for population effect
    se2_a = 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))
    # se_b is standard deviation for delta(t)
    se2_b = apply(efunctions, 1, function(x) sqrt(x %*% as.matrix(fixed.vcov2[match(paste0("bscore.",
        1:npc), names(fixeff2)), match(paste0("bscore.", 1:npc),
        names(fixeff2))]) %*% x))
    yhat2 <- predict(FMM)

    ###############################################################################

    return(list(fPCA_result = list(fpca_result = results, npc = npc,
        ascore = ascore, efunctions = efunctions, evalues = evalues),
        test_result = list(`equal-variance` = list(EqualVar.test = EqualVar.test,
            EqualVar.pvalue = EqualVar.pvalue), bonferroni = list(bonf.test = bonf.test,
            bonf.pvalue = bonf.pvalue), asLRT = list(LRT.test = LRT.test,
            LRT.pvalue = LRT.pvalue)), estimation_result = list(FLM = list(fixed = list(`beta(t)` = beta
            `delta(t)` = beta1_b, coefficient = fixed.coeff1,
            vcov = fixed.vcov1, sigma = sigma1, `se_beta(t)` = se1_a,
            `se_delta(t)` = se1_b), random = list(Zi = raneffect1),
            yhat = yhat1, FLM.fit = FLM), FMM = list(fixed = list(`beta(t)` = beta2_a,
            `delta(t)` = beta2_b, coefficient = fixed.coeff2,
            vcov = fixed.vcov2, sigma = sigma2, `se_beta(t)` = se2_a,
            `se_delta(t)` = se2_b), 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)
library(ggplot2)

###############################################################################

### generate a training dataset ###
```

```r
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
indp = 0  # if 1, Xt is independent across subjects; if 0, Xt is correalted across subjects according t
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  # 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),
    ]

# generate random intercept
alphaI.true.i <- mapply(rnorm, nSubj, a.mean, rep(sqrt(gammaVar.true),
    1))
alphaI.true <- alphaI.true.i[rep(1:nrow(alphaI.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)
}
```

```r
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
if (indp == 0) {
    ascore.true_b <- mvrnorm(totalN, rep(a.mean, npc.true), diag(lambdaVec.true))  #exchangeable correl
    ascore.true_a <- mvrnorm(nSubj, rep(a.mean, npc.true), diag(lambdaVec.true))  #exchangeable correla
    ascore.true <- ascore.true_a[rep(1:nrow(ascore.true_a), each = nRep),
        ] + ascore.true_b
}
if (indp == 1) {
    ascore.true <- mvrnorm(totalN, rep(a.mean, npc.true), diag(lambdaVec.true))  #exchangeable correlat
}

Mt.true <- ascore.true %*% psi.true
error <- rnorm(totalN, mean = 0, sd = sd.epsilon)

if (heter == 0) {
    thetaIK.true1 <- mvrnorm(nSubj, rep(0, 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 + thetaK.true * dummyX) *
    ascore.true
betaM.true <- rowSums(betaM.true)

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

if (smooth == 0) {
    if (indp == 1) {
        Merror.Var <- sum(lambdaVec.true)/SNR  #SNR = sum(lambdaVec.true)/Merror.Var
    } else {
        Merror.Var <- 2 * 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 three testing procedures and etimation using FLM and FMM on the training dataset

```
##############################################################################

result <- fmm_sofr(traindata)

##############################################################################
```

2.3 Generate a test dataset containing 10 new visits for each subject

```
### generate a test dataset ###

### for each subject, generate newvisitN more new visits
gene_newdata <- function(newvisitN, multi = 1, mu, efunctions) {
    # newvisitN is the number of new visits for each subject
    # multi is the number of functional predictors; multi=1
    # denotes the univariate analysis mu is the population mean
    # (vector) efunctions is the eigenfunctions estimated from
    # fpca.face

    totalN <- nSubj * newvisitN
    # generate new Z
    dummyX <- rbinom(n = nSubj * newvisitN, size = 1, prob = 0.5)  # new Z

    # generate new x_ij(t) for new visit
    if (indp == 0) {
        ascore.true_b <- mvrnorm(totalN, rep(a.mean, npc.true),
            diag(lambdaVec.true))  #exchangeable correlation a_ik+b_ijk
        ascore.true_a <- mvrnorm(nSubj, rep(a.mean, npc.true),
            diag(lambdaVec.true))  #exchangeable correlation a_ik+b_ijk
        ascore.true <- ascore.true_a[rep(1:nrow(ascore.true_a),
            each = newvisitN), ] + ascore.true_b
    }
    if (indp == 1) {
        ascore.true <- mvrnorm(totalN, rep(a.mean, npc.true),
            diag(lambdaVec.true))  # no correlation a_ik
    }

    # 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)
    alphaI.true <- alphaI.true.i[rep(1:nrow(alphaI.true.i), each = newvisitN),
        ]
```

```
    thetaIK.true <- thetaIK.true1[rep(1:nrow(thetaIK.true1),
        each = newvisitN), ]
    betaM.true <- (thetaK.true + thetaIK.true + thetaK.true *
        dummyX) * 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 + alphaI.true + dummyX * gammaI.true + betaM.true +
        error
    ID <- rep(1:nSubj, each = newvisitN)

    if (smooth == 0) {
        if (indp == 1) {
            Merror.Var <- sum(lambdaVec.true)/SNR/multi  #SNR = sum(lambdaVec.true)/Merror.Var
        } else {
            Merror.Var <- 2 * sum(lambdaVec.true)/SNR/multi  #SNR = sum(lambdaVec.true)/Merror.Var
        }
        Mt.hat <- Mt.true + matrix(rnorm(totalN * D, mean = 0,
            sd = sqrt(Merror.Var)), totalN, D * multi)
    }
    if (smooth == 1) {
        Merror.Var <- 0  #SNR = sum(lambdaVec.true)/Merror.Var
        Mt.hat <- Mt.true
    }

    M <- Mt.hat
    Xt_demean <- M - as.matrix(rep(1, nSubj * newvisitN)) %*%
        t(data.matrix(mu))  # demean
    # projection
    ascore <- Xt_demean %*% efunctions/D
    bscore <- ascore * dummyX

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

    return(list(designMatrix = designMatrix, newY = Y, Xt = Xt_demean))
}


# generate new testdata
newvisit <- 10
set.seed(2000)
testdata <- gene_newdata(newvisit, 1, result$fPCA_result$fpca_result$mu,
    result$fPCA_result$efunctions)
```

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,
```

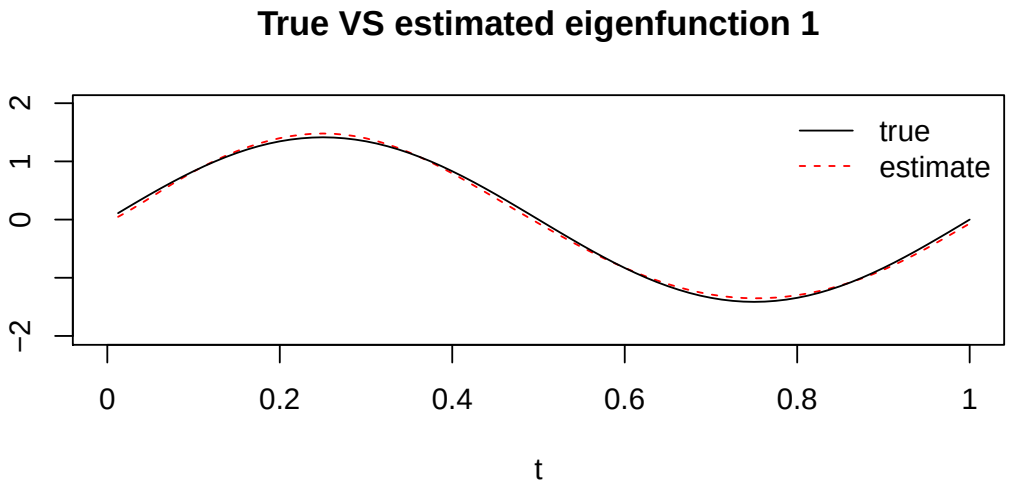


Figure 1: output

```
    testdata$designMatrix)
test_FMM.yhat <- predict(result$estimation_result$FMM$FMM.fit,
    testdata$designMatrix)
```

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
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 = 1)
}
```

3.2 Three testing results

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

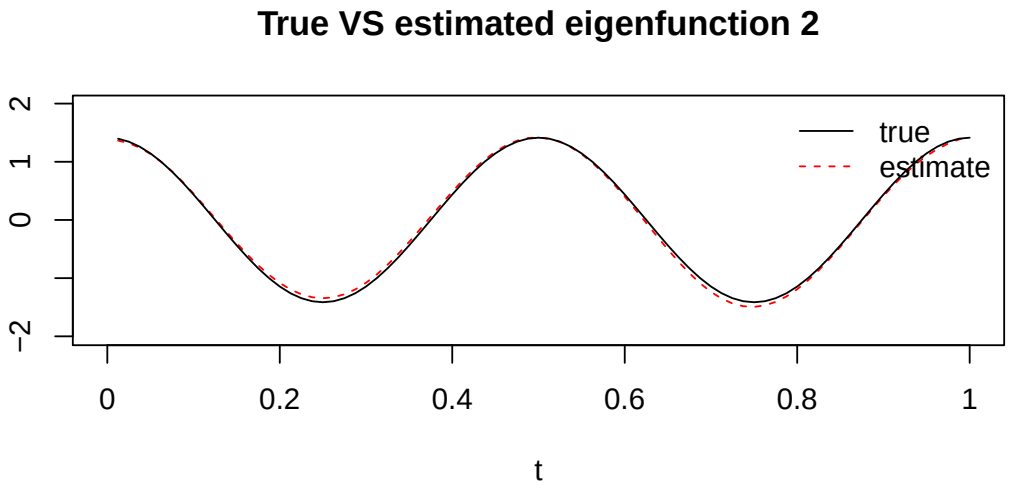


Figure 2: output

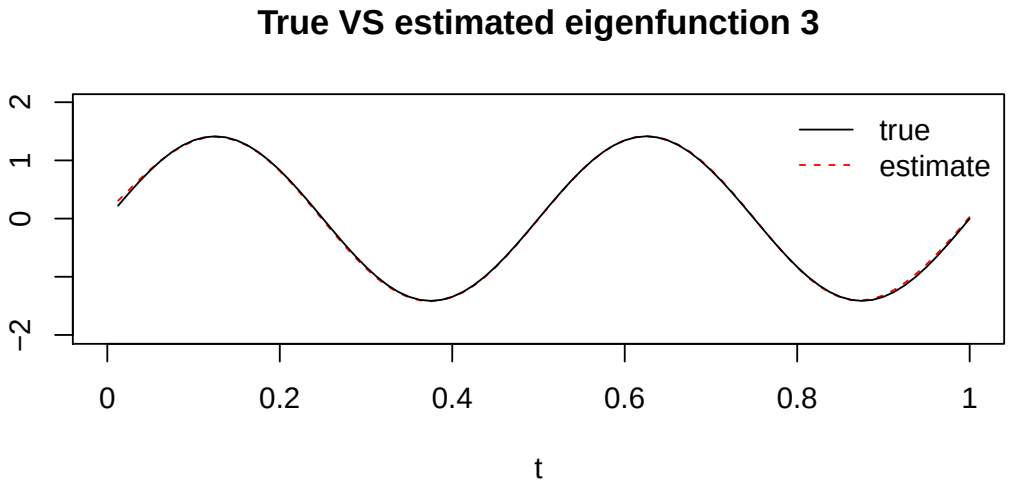


Figure 3: output

```
# $EqualVar.test
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 43.629, 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 = 38.331, p-value < 2.2e-16
#
#
# $bonf.test[[2]]
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 12.245, p-value = 1e-04
#
#
# $bonf.test[[3]]
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 1.819e-12, p-value = 0.4538
#
#
#
# $bonf.pvalue
# [1] 0e+00 3e-04 1e+00
```

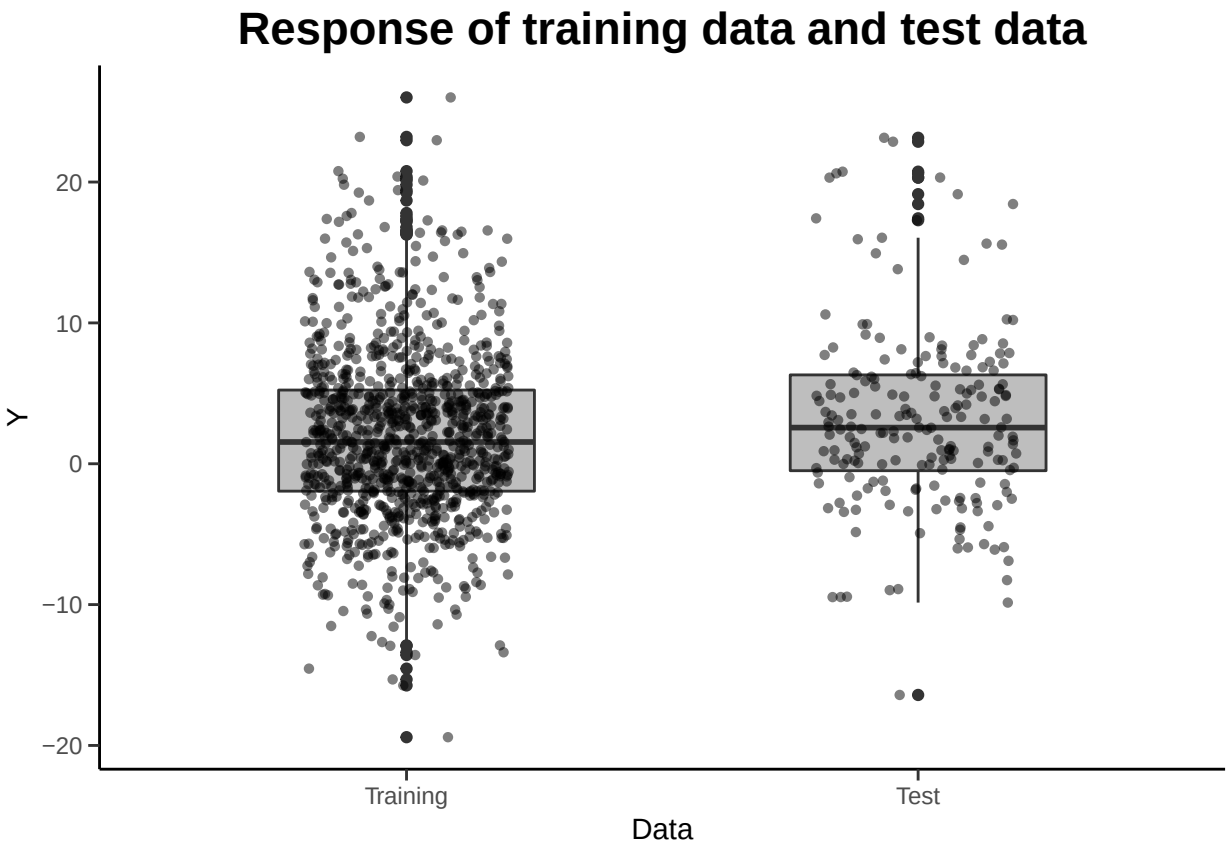
```
# asLRT result
result$test_result$asLRT
```

```
# $LRT.test
# $LRT.test$observed
```

```
# [1] 47.99627
#
# $LRT.test$cutoff
# [1] 5.436174
#
# $LRT.test$pvalue
# [1] 4.2504e-11
#
#
# $LRT.pvalue
# [1] 4.2504e-11
```

3.3 Visualize the response generated for the new visits

```
YY <- data.frame(Y = c(traindata$Y, testdata$newY), class = as.factor(c(rep(1,
    length(traindata$Y)), rep(2, length(testdata$newY)))))
ggplot(YY, aes(x = class, y = Y)) + geom_boxplot(fill = "gray",
    width = 0.5) + geom_jitter(shape = 16, position = position_jitter(0.2),
    alpha = 0.5) + labs(title = "Response of training data and test data",
    x = "Data", y = "Y") + scale_x_discrete(breaks = c("1", "2"),
    labels = c("Training", "Test")) + theme_classic() + theme(plot.title = element_text(hjust = 0.5,
    size = rel(1.5), face = "bold")) + theme(axis.ticks.length = unit(0.15,
    "cm"))
```



3.4 Visualize population effect estimation using FLM VS FMM

```
# true beta(t)
beta_t <- t(psi.true) %*% as.vector(rep(thetaK.true, npc.true))
# true delta(t)
delta_t <- beta_t
beta_t.FMM <- result$estimation_result$FMM$fixed$`beta(t)`
delta_t.FMM <- result$estimation_result$FMM$fixed$`delta(t)`
beta_t.FLM <- result$estimation_result$FLM$fixed$`beta(t)`
delta_t.FLM <- result$estimation_result$FLM$fixed$`delta(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: 0.092"
```

```
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: 0.105"
```

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

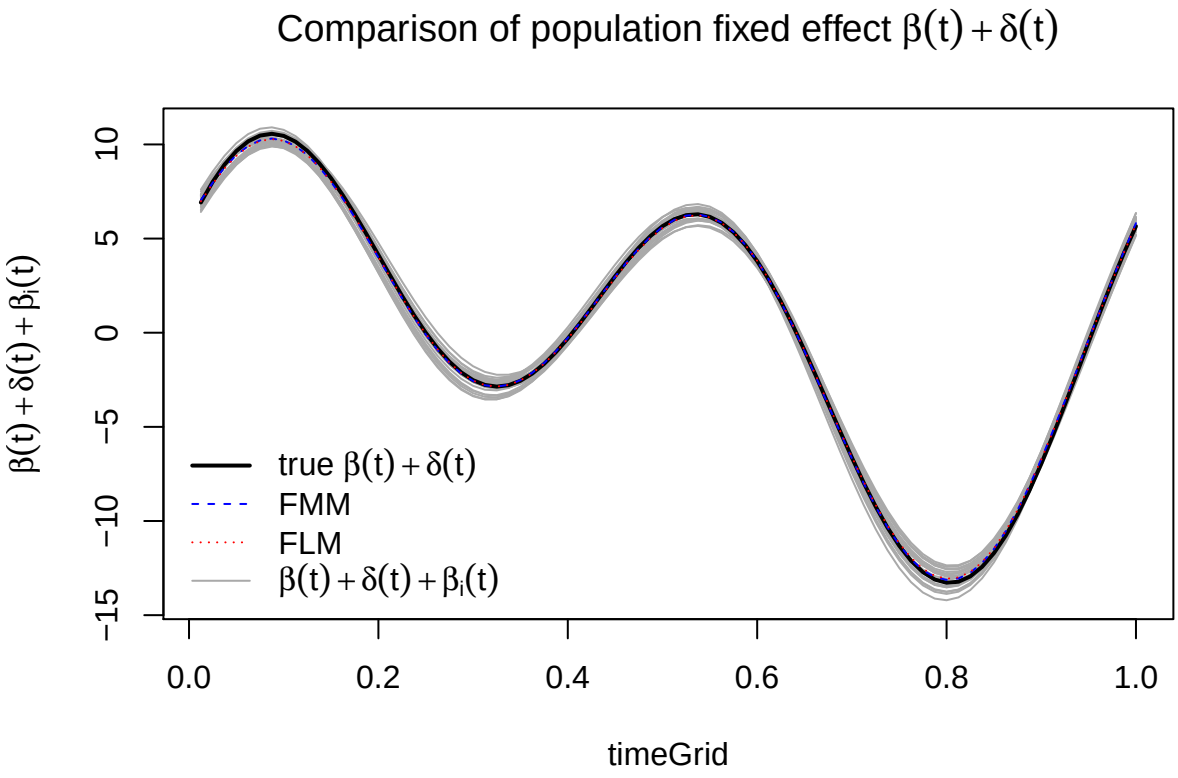
```
# [1] "MISE of estimated delta(t) using FMM is: 0.068"
```

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

```
# [1] "MISE of estimated delta(t) using FLM is: 0.058"
```

```
betait.FMM <- result$estimation_result$FMM$random$`beta_i(t)`
beta.comp <- cbind(apply(betait.FMM, 2, function(x) x + beta_t.FMM +
    delta_t.FMM), beta_t + delta_t, beta_t.FMM + delta_t.FMM,
    beta_t.FLM + delta_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) + delta(t))), ylab = expression(beta(t) + delta(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) + delta(t))),
    "FMM", "FLM", expression(beta(t) + delta(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 = 1)
```



3.5 Visualize out-of-sample prediction MSE for each subject

```
################################################################################

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

# MSE for each subject
ID <- testdata$designMatrix$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: 1.669"
```

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

```
# [1] "MSE of FLM is: 1.858"
```

```r
# 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])
```

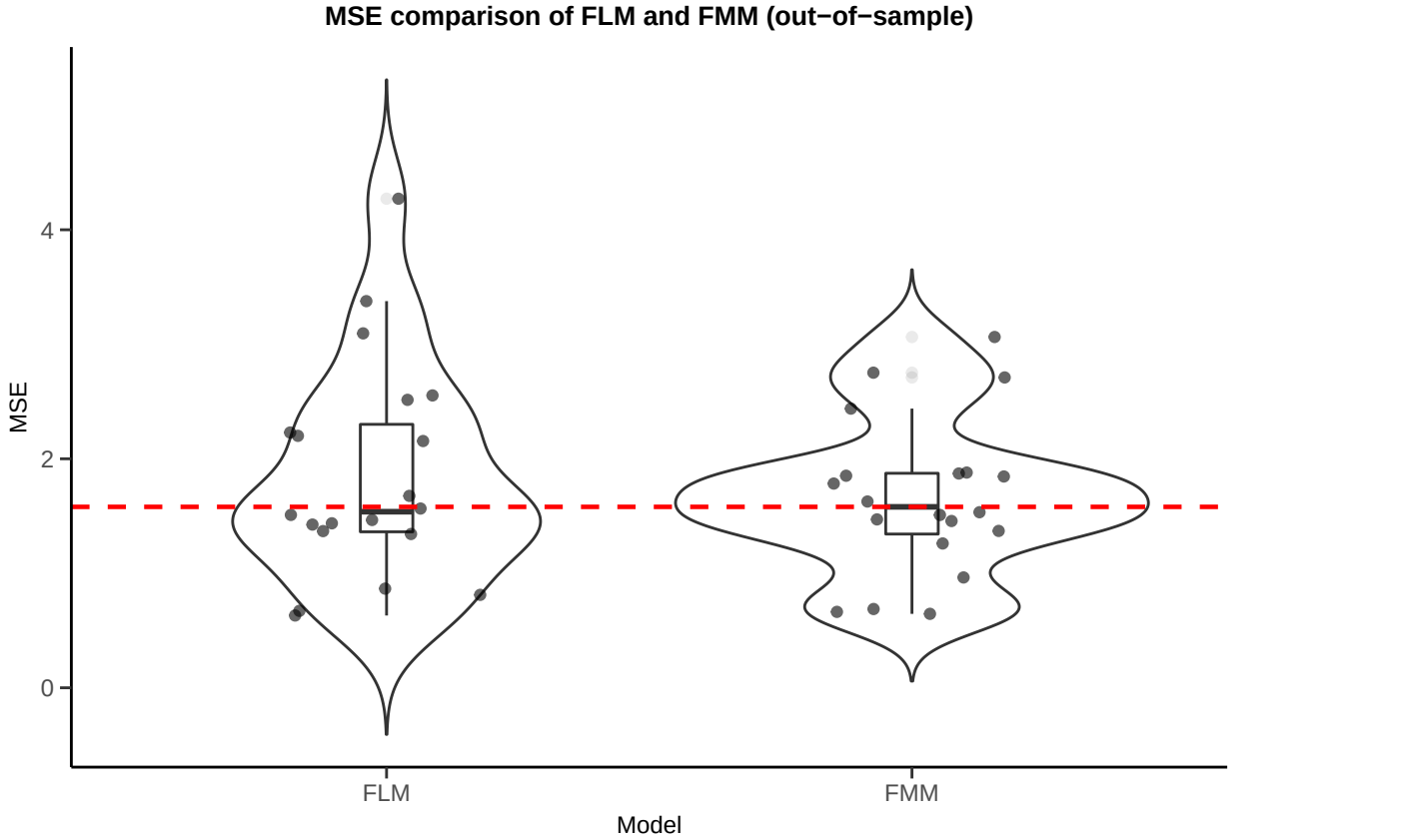
```
# [1] "R-square of FMM is: 0.958"
```

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

```
# [1] "R-square of FLM is: 0.954"
```

```r
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))) + theme(axis.ticks.length = unit(0.15,
    "cm"))
```



4. Multivariate functional predictor

4.1 Simulate a training dataset for the multivariate case

```
################################################################################

### generate a training dataset ###
multi <- 3
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
indp = 0  # if 1, Xt is independent across subjects; if 0, Xt is correalted across subjects according t
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),
    ]

# generate random intercept
alphaI.true.i <- mapply(rnorm, nSubj, a.mean, rep(sqrt(gammaVar.true),
    1))
alphaI.true <- alphaI.true.i[rep(1:nrow(alphaI.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 <- c(sqrt(2/3) * sinpi(2 * t), sqrt(2/3) * cospi(t),
```

```
            sqrt(2/3) * sinpi(t))
    } else if (degree == 2) {
        result <- c(sqrt(2/3) * cospi(4 * t), sqrt(2/3) * cospi(2 *
            t), sqrt(2/3) * sinpi(2 * t))
    } else if (degree == 3) {
        result <- c(sqrt(2/3) * sinpi(4 * t), sqrt(2/3) * cospi(3 *
            t), sqrt(2/3) * sinpi(3 * 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, byrow = TRUE)
psi.true <- t(apply(psi.true, 1, function(x) as.vector(matrix(x,
    ncol = multi, byrow = T))))

# generate functional covariates
if (indp == 0) {
    ascore.true_b <- mvrnorm(totalN, rep(a.mean, npc.true), diag(lambdaVec.true))  #exchangeable correl
    ascore.true_a <- mvrnorm(nSubj, rep(a.mean, npc.true), diag(lambdaVec.true))  #exchangeable correla
    ascore.true <- ascore.true_a[rep(1:nrow(ascore.true_a), each = nRep),
        ] + ascore.true_b
}
if (indp == 1) {
    ascore.true <- mvrnorm(totalN, rep(a.mean, npc.true), diag(lambdaVec.true))  #exchangeable correlat
}

Mt.true <- ascore.true %*% psi.true
error <- rnorm(totalN, mean = 0, sd = sd.epsilon)

if (heter == 0) {
    thetaIK.true1 <- mvrnorm(nSubj, rep(0, 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 + thetaK.true * dummyX) *
    ascore.true
betaM.true <- rowSums(betaM.true)

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

if (smooth == 0) {
```

```
    if (indp == 1) {
        Merror.Var <- sum(lambdaVec.true)/SNR/multi  #SNR = sum(lambdaVec.true)/Merror.Var
    } else {
        Merror.Var <- 2 * sum(lambdaVec.true)/SNR/multi  #SNR = sum(lambdaVec.true)/Merror.Var
    }
    Mt.hat <- Mt.true + matrix(rnorm(totalN * D * multi, mean = 0,
        sd = sqrt(Merror.Var)), totalN, D * multi)
}
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)
```

4.2 Perform three testing procedures and etimation using FLM and FMM on the multivariate training dataset

```
##############################################################################

result <- fmm_sofr(traindata, multi = 3)

##############################################################################
```

4.3 Generate a test dataset containing 10 new visits for each subject

```
# generate new testdata
newvisit <- 10
set.seed(2000)
testdata <- gene_newdata(newvisit, multi = 3, as.vector(sapply(result$fPCA_result$fpca_result$FPCA,
    function(x) x$mu)), result$fPCA_result$efunctions)
```

4.4 Etimation using FLM and FMM on the testing dataset

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

4.5 Visualize results for the multivariate case

4.5.1 Visualize estimated eigenfunctions vs true eigenfunctions

```
### FPCA ###
efunctions <- result$fPCA_result$efunctions
```

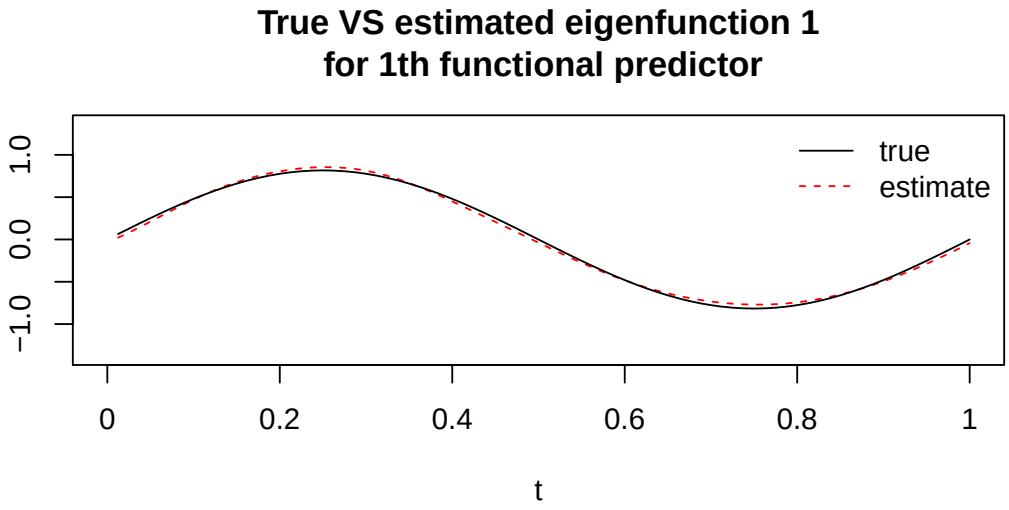


Figure 4: output

```
npc <- result$fPCA_result$npc
# correct for sign of eigen-functions
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) {
    for (j in 1:multi) {
        plot(timeGrid, efunctions.correct[((j - 1) * D + 1):(j *
            D), i], type = "l", xlab = "t", ylab = "", xaxt = "n",
            xlim = c(0, 1), main = paste0("True VS estimated eigenfunction ",
                i, "\n for ", j, "th functional predictor"),
            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, ((j - 1) * D + 1):(j * D)],
            col = "black")
        legend("topright", c("true", "estimate"), col = c("black",
            "red"), lty = c(1, 2), bty = "n", cex = 1)
    }
}
```

4.5.2 Three 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:
```

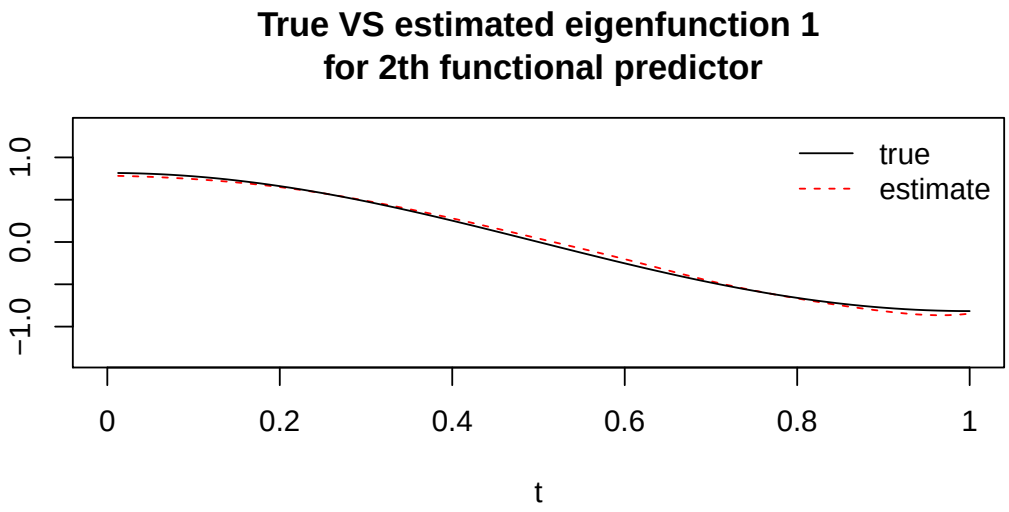


Figure 5: output

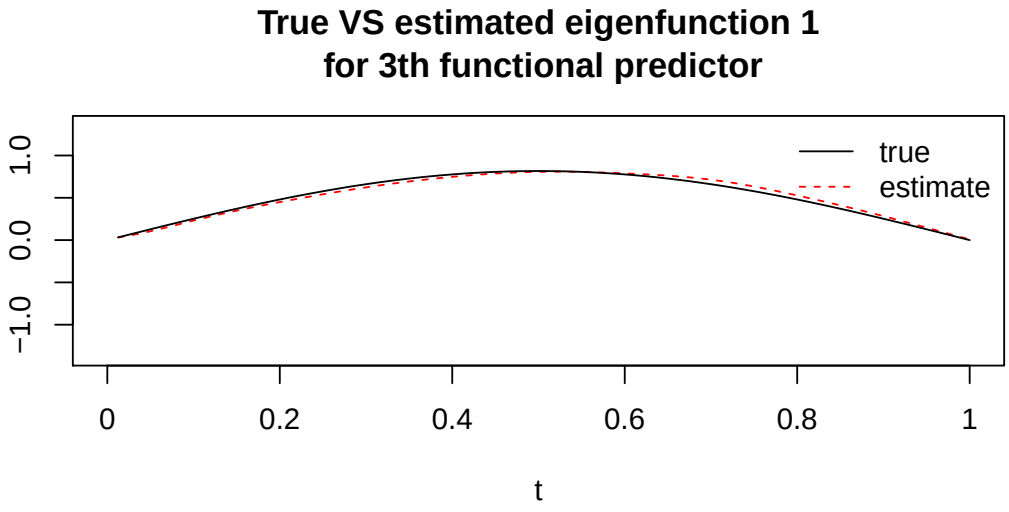


Figure 6: output

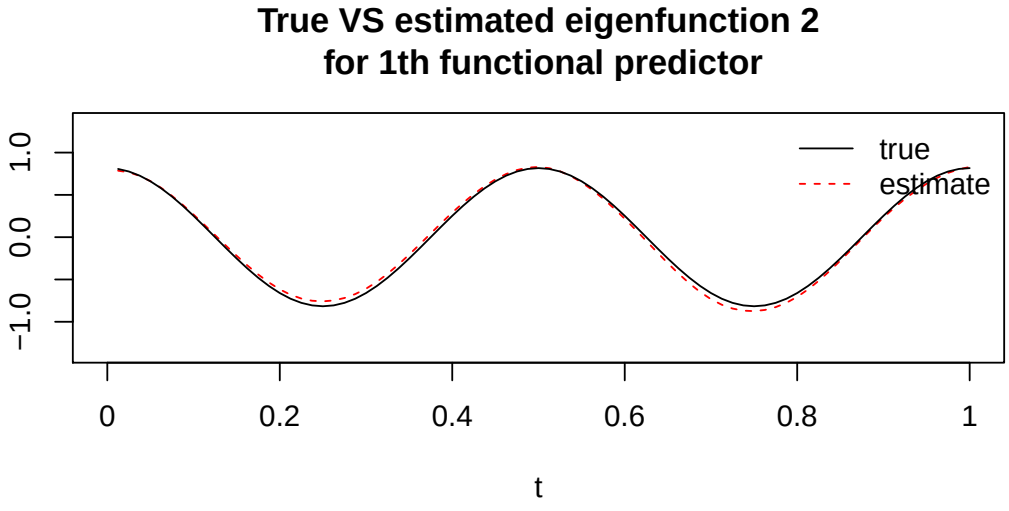


Figure 7: output

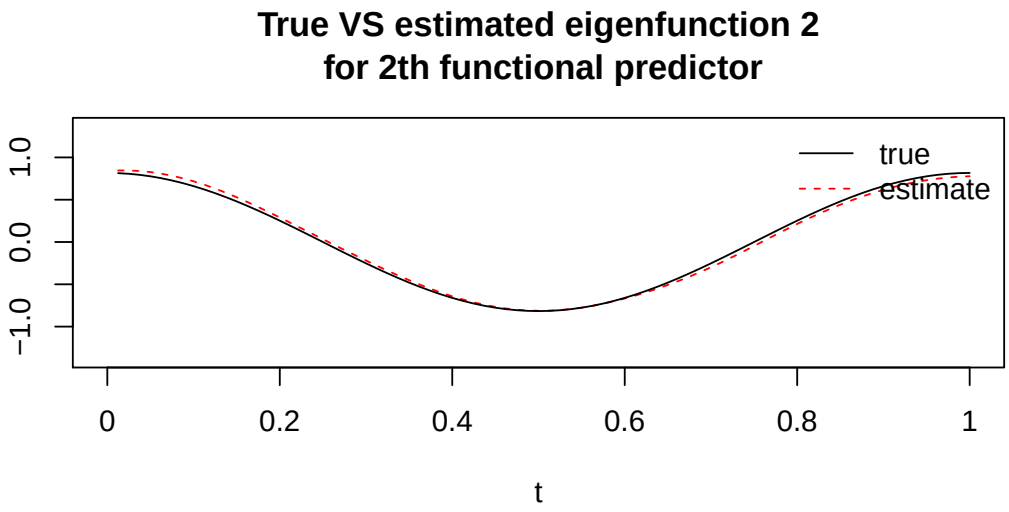


Figure 8: output

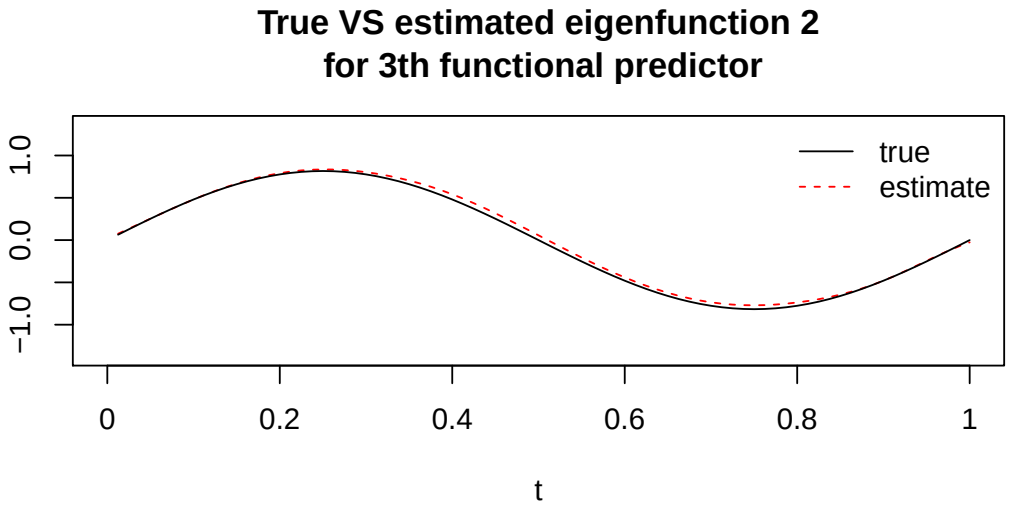


Figure 9: output

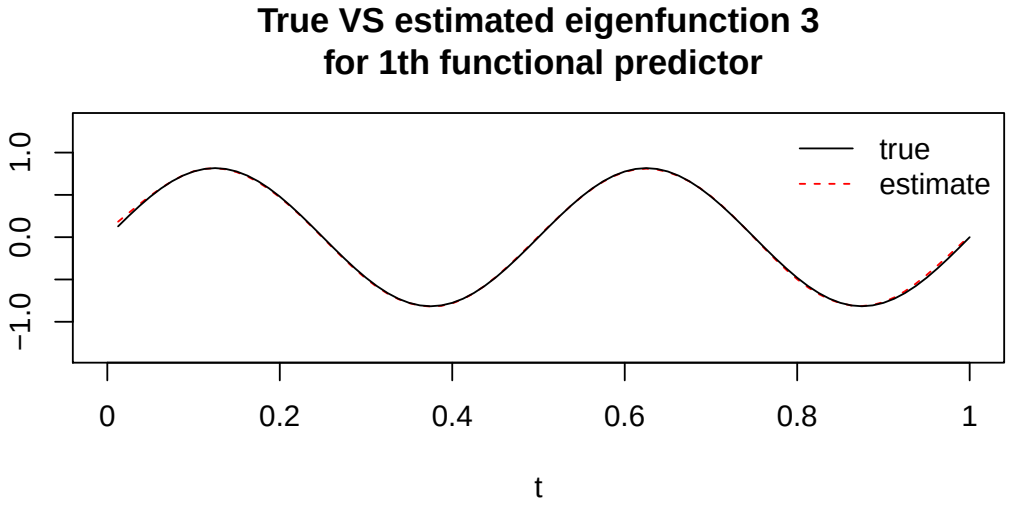


Figure 10: output

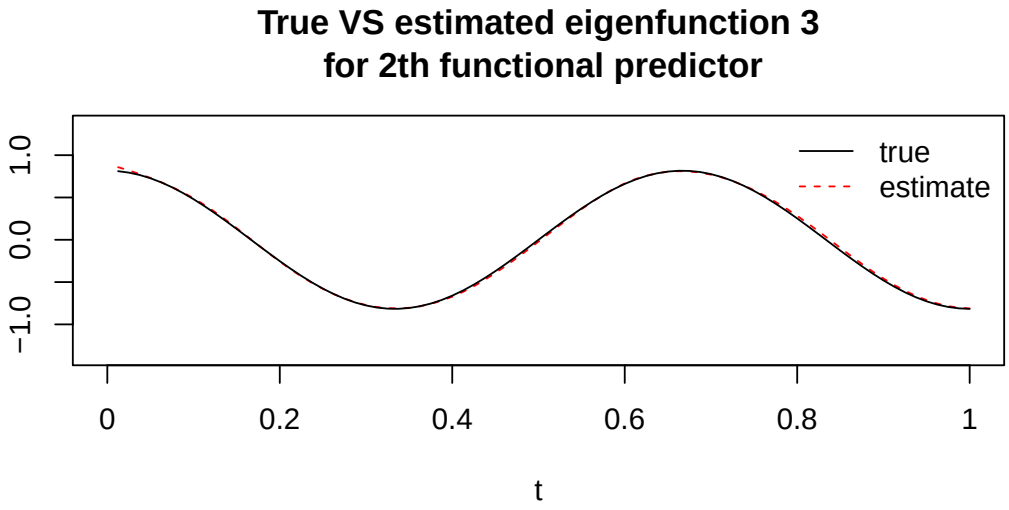


Figure 11: output

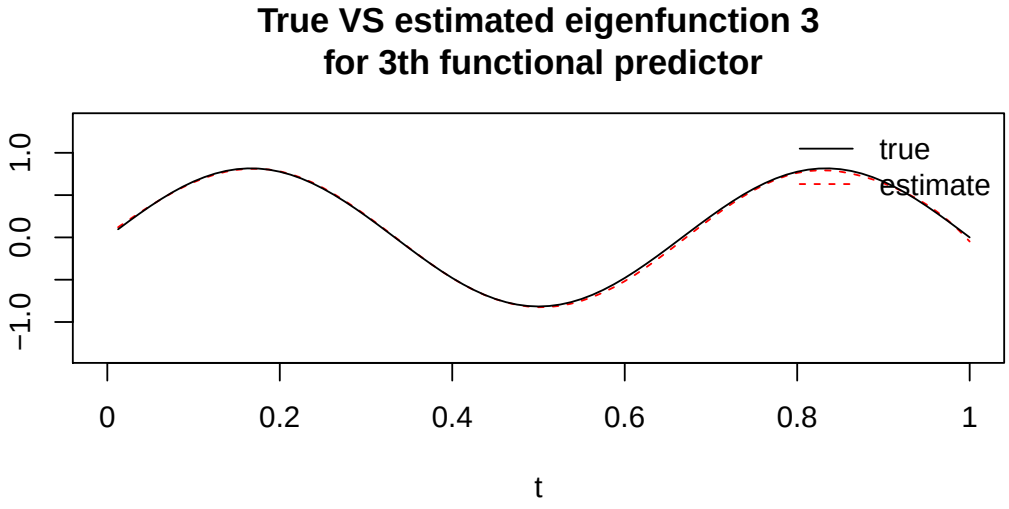


Figure 12: output

```
# RLRT = 66.723, 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 = 58.055, p-value < 2.2e-16
#
#
# $bonf.test[[2]]
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 15.022, p-value < 2.2e-16
#
#
# $bonf.test[[3]]
#
#   simulated finite sample distribution of RLRT.
#
#   (p-value based on 10000 simulated values)
#
# data:
# RLRT = 0, p-value = 1
#
#
#
# $bonf.pvalue
# [1] 0 0 1
```

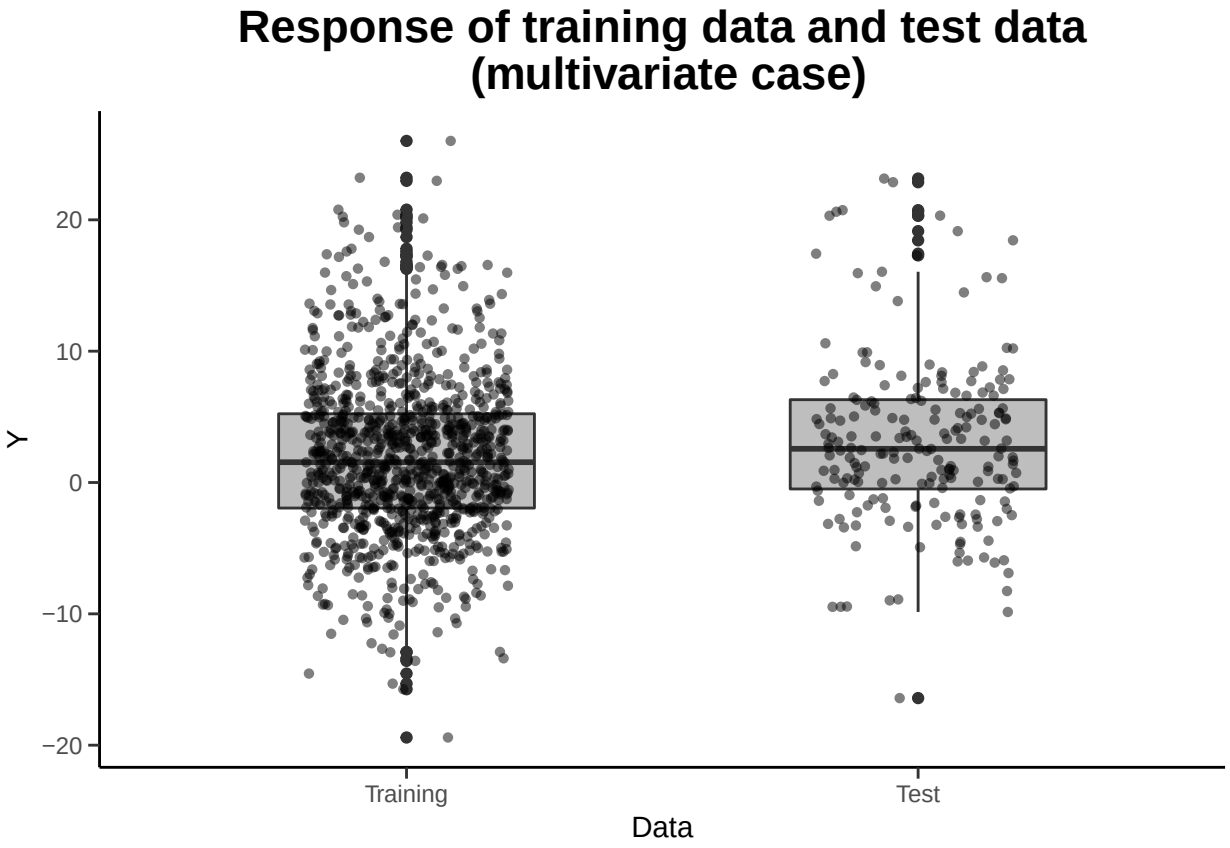
```
# asLRT result
result$test_result$asLRT
```

```
# $LRT.test
# $LRT.test$observed
# [1] 72.11116
#
# $LRT.test$cutoff
# [1] 5.437065
#
# $LRT.test$pvalue
# [1] 2.781351e-16
```

```
#
#
# $LRT.pvalue
# [1] 2.781351e-16
```

4.5.3 Visualize the response generated for the new visits

```
YY <- data.frame(Y = c(traindata$Y, testdata$newY), class = as.factor(c(rep(1,
    length(traindata$Y)), rep(2, length(testdata$newY)))))
ggplot(YY, aes(x = class, y = Y)) + geom_boxplot(fill = "gray",
    width = 0.5) + geom_jitter(shape = 16, position = position_jitter(0.2),
    alpha = 0.5) + labs(title = "Response of training data and test data\n (multivariate case)",
    x = "Data", y = "Y") + scale_x_discrete(breaks = c("1", "2"),
    labels = c("Training", "Test")) + theme_classic() + theme(plot.title = element_text(hjust = 0.5,
    size = rel(1.5), face = "bold")) + theme(axis.ticks.length = unit(0.15,
    "cm"))
```



4.5.4 Visualize population effect estimation using FLM VS FMM

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

```r
beta_t.FLM <- result$estimation_result$FLM$fixed$`beta(t)`
delta_t.FLM <- result$estimation_result$FLM$fixed$`delta(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: 0.014"
```

```r
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: 0.016"
```

```r
sprintf("MISE of estimated delta(t) using FMM is: %.3f", mean((delta_t.FMM -
    delta_t)^2))
```

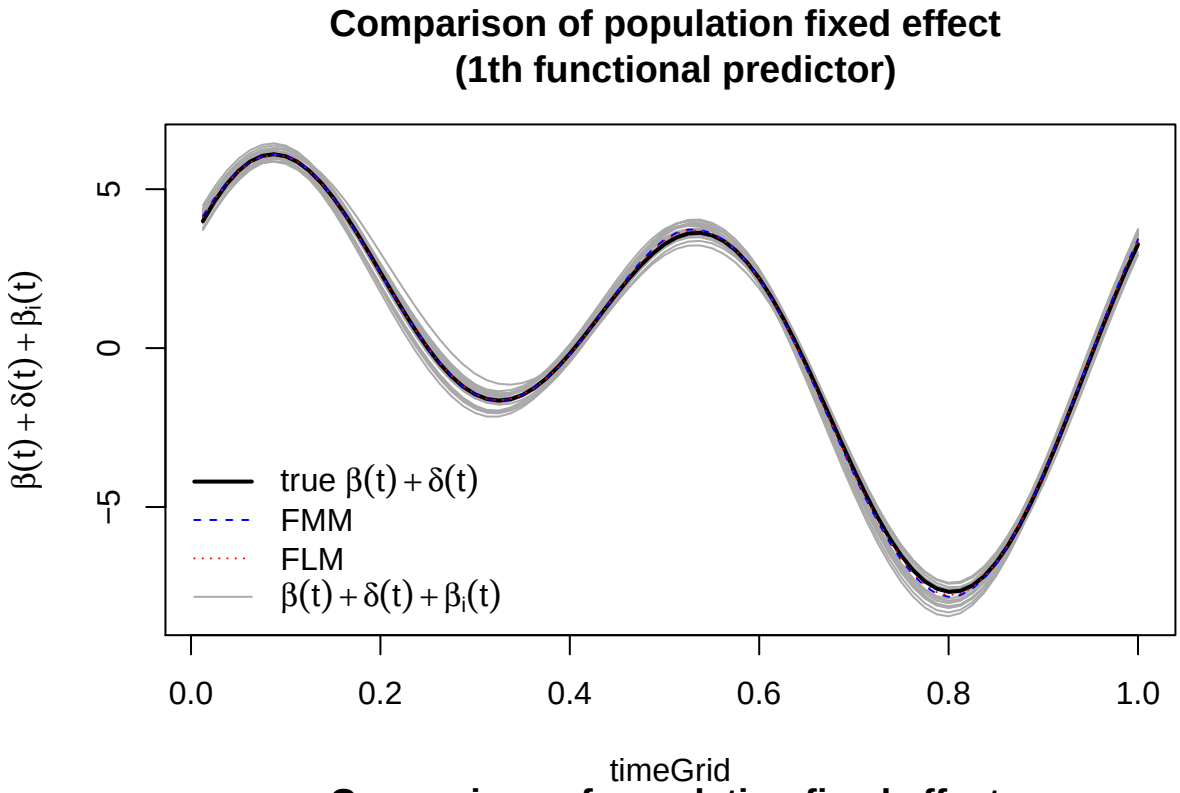
```
# [1] "MISE of estimated delta(t) using FMM is: 0.032"
```

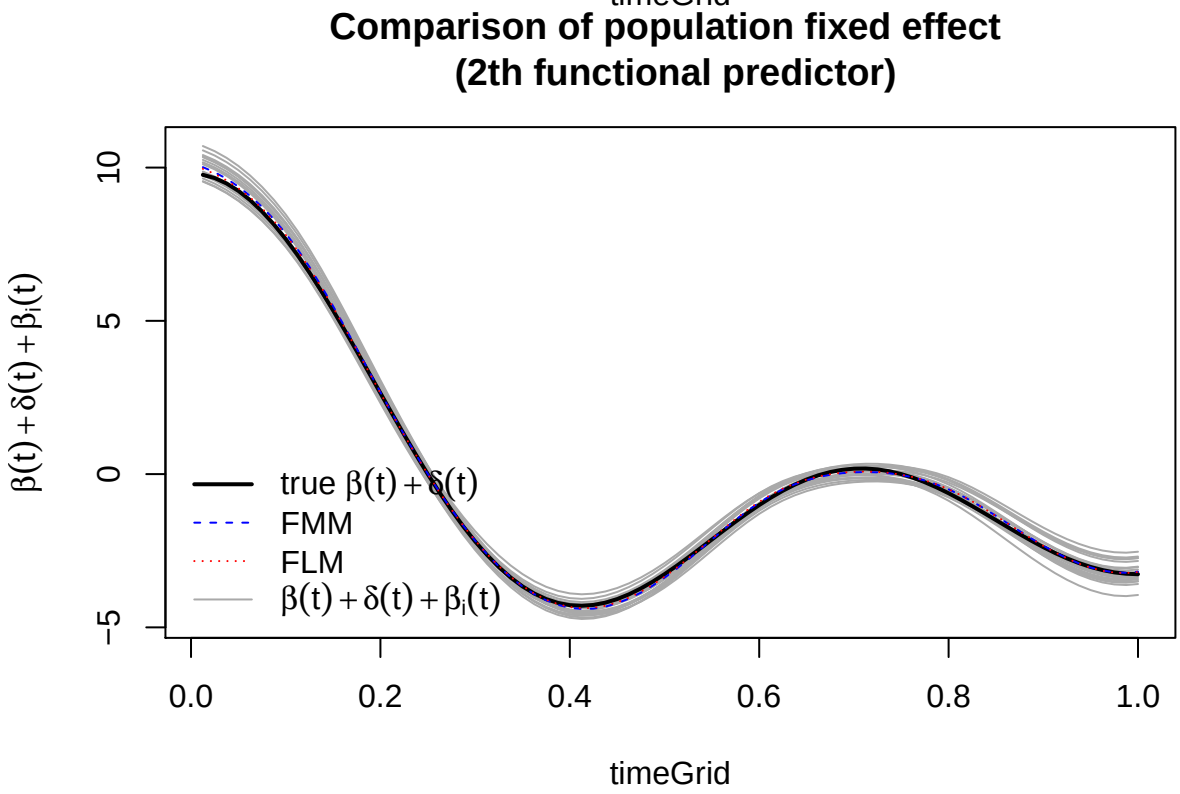
```r
sprintf("MISE of estimated delta(t) using FLM is: %.3f", mean((delta_t.FLM -
    delta_t)^2))
```

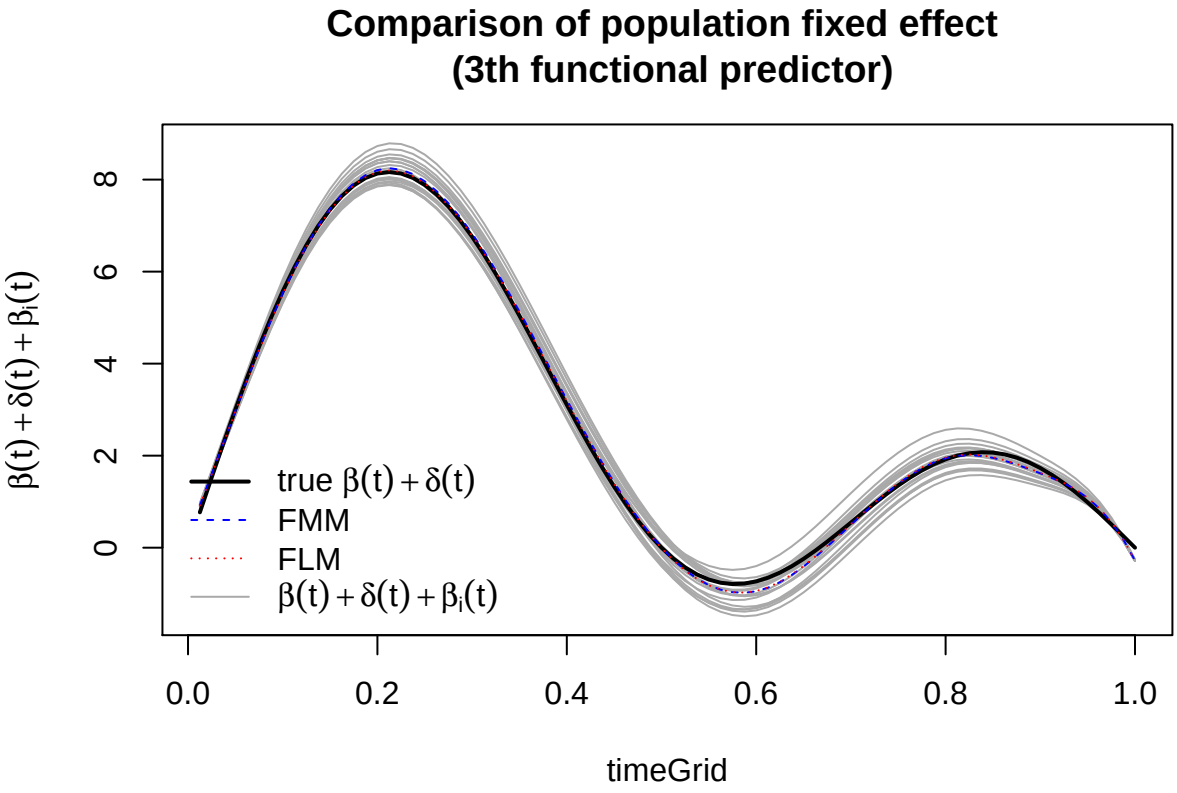
```
# [1] "MISE of estimated delta(t) using FLM is: 0.027"
```

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

# visualization
for (j in 1:multi) {
    matplot(timeGrid, beta.comp[((j - 1) * D + 1):(j * D), 1:nSubj],
        col = "darkgrey", type = "l", lty = 1, main = paste0("Comparison of population fixed effect \n
            j, "th functional predictor)"), ylab = expression(beta(t) +
            delta(t) + beta[i](t)))
    lines(timeGrid, beta.comp[((j - 1) * D + 1):(j * D), nSubj +
        1], col = "black", lty = 1, lwd = 2)
    lines(timeGrid, beta.comp[((j - 1) * D + 1):(j * D), nSubj +
        2], col = "blue", lty = 2)
    lines(timeGrid, beta.comp[((j - 1) * D + 1):(j * D), 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) +
        delta(t))), "FMM", "FLM", expression(beta(t) + delta(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 = 1)
}
```


Comparison of population fixed effect
(1th functional predictor)
β(t) + δ(t) + βi(t)
5
0
−5
true β(t) + δ(t)
FMM
FLM
β(t) + δ(t) + βi(t)
0.0
0.2
0.4
0.6
0.8
1.0
timeGrid


Comparison of population fixed effect
(2th functional predictor)
β(t) + δ(t) + βi(t)
10
5
0
−5
true β(t) + δ(t)
FMM
FLM
β(t) + δ(t) + βi(t)
0.0
0.2
0.4
0.6
0.8
1.0
timeGrid



4.5.5 Visualize out-of-sample prediction MSE for each subject

```
##############################################################################

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

# MSE for each subject
ID <- testdata$designMatrix$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: 1.492"
```

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

```
# [1] "MSE of FLM is: 1.715"
```

```
# 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: 0.968"
```

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

```
# [1] "R-square of FLM is: 0.963"
```

```
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)\n(mul
    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))) + theme(axis.ticks.length = unit(0.15,
    "cm"))
```