```r
## THIS FILE bivFns_3c.r IS  BASED ON bivFns_lik.r, USED FOR BIVARIATE ANALYSIS FOR BOTH SIMULATION
## AND REAL DATASETS, for subj. with 3 concecutive cycles.
## IT OUTPUTS THE PARAMETER ESTIMATES, BIASES AND COVARIANCES.

library(sde) # rsOU()
library(MASS) # mvrnorm()
library(psych) # tr()
library(magic) # adiag()
library(Rcpp) # TO C++
sourceCpp("matrixmulti_1.cpp") # NEED TO INSTALL Package 'RcppEigen'
library(e1071) # rwiener()

#=====================================================================================
# SIMULATIONS OF DATASETS
#=====================================================================================
#-------------------------------------------------------------------------------
# A FUNCTION that RETURNS a BIV. LONGITUDINAL DATASET by SIMULATION
# m is the number of subjects
# tp is the number of distinct points, assuming each subj. has the same obs'd time pt.
# beta is a vector of fixed effect parameters
# cycle is the number of cycles to be simulated

newData <- function(theta, beta, m, tp, process, cycle){

  n <- m*tp*cycle
  # number of observations each subj. has
  subjlen <- tp*cycle

  #---------------------------------------------------------------------------
  # FIXED EFFECTS
  # X
  age <- sample(20:44, size = m, replace = T)
  age <- rep(age, each = subjlen)

  # PERIODIC SMOOTH FUNCTIONS
  f1 <- sine(tp/10, rep((1:tp)/10, cycle))
  f2 <- cosine(tp/10, rep((1:tp)/10, cycle))

  #---------------------------------------------------------------------------
  # RANDOM INTERCEPTS
  D <- matrix(c(theta[3], theta[4], theta[4], theta[5]), 2, 2) # variance for bi
  b <- mvrnorm(m, c(0, 0), D)
  b1 <- rep(b[,1], each = subjlen)
  b2 <- rep(b[,2], each = subjlen)

  # MEASUREMENT ERROR - epsilon
  sigma <- diag(c(theta[6], theta[7])) # variance for epsilon
  eps <- mvrnorm(n, c(0, 0), sigma)

  # BIVARIATE GAUSSIAN FIELD
  if (process == "OU") {
    u1 <- rsOU(n, theta=c(0, theta[8], theta[9]))
    u2 <- rsOU(n, theta=c(0, theta[10], theta[11]))
  } else if (process == "NOU") {
    t <- (1:subjlen %% 28)/10
    tPlus <- c(t[2:subjlen], 0.1)
    exponent1 <- abs(tPlus - t)*log(2*theta[8]) + theta[9] + theta[10]*s(28/30, t) + theta[11]*s(56/30, t)
    exponent2 <- abs(tPlus - t)*log(2*theta[12]) + theta[13] + theta[14]*s(28/30, t) + theta[15]*s(56/30, t)
    theta13 <- rep(exp(exponent1/2), m)
    theta23 <- rep(exp(exponent2/2), m)
    u1 <- sapply(1:n, function(v) return(rsOU(1, theta=c(0, theta[8], theta13[v]))))
    u2 <- sapply(1:n, function(v) return(rsOU(1, theta=c(0, theta[12], theta23[v]))))
  } else if (process == "Wiener") {
    u1 <- theta[8]*sapply(rep(1, m), function(x) return(rwiener(28, x))) # tp * m matrix
    u2 <- theta[9]*sapply(rep(1, m), function(x) return(rwiener(28, x)))
    u1 <- as.vector(u1)
    u2 <- as.vector(u2)
  }

  #---------------------------------------------------------------------------
  # BIVARIATE CYCLIC RESPONSE Y
  Y1 <- beta[1]*age + rep(f1, m) + b1 + u1 + eps[,1]
  Y2 <- beta[2]*age + rep(f2, m) + b2 + u2 + eps[,2]

  id <- rep(1:m, each = subjlen)
  day <- rep(1:subjlen, m)
  simData <- data.frame(id, day, Y1, Y2, age)

  return(simData)
}

#-------------------------------------------------------------------------------
# HELPER FUNCTIONS that RETURN A PERIODIC FUNCTION
sine <- function(tp, x) {
  f1 <- 5*sin((2*pi/tp)*x)
```

```
  return(f1)
}
cosine <- function(tp, x) {
  f2 <-  3*cos((2*pi/tp)*x)
  return(f2)
}

# A HELPER FUNCTION that RETURNS coefficients of periodic cubic splines
s <- function(knot, t) {
  T <- 28
  a <- -T*(T - knot)/2 + 3*(T - knot)^2/2 - (T - knot)^3/T
  b <- 3*(T - knot)/2 - 3*(T - knot)^2/(2*T)
  c <- -(T - knot)/T
  sj <- a*t + b*t^2 + c*t^3 + (t - knot)^3*((t - knot)^3>0)
  return(sj)
}

#=========================================================================================
# HELPER FUNCTIONS
#=========================================================================================
#-------------------------------------------------------------------------------
# A HELPER FUNCTION that RETURNS X, Y, K, N1, N2, r, B1dou, B2dou & niVec from the given dataframe.
helperFn <- function(data, time, id, fixed, response) {

  #-----------------------------------------------------------------------------
  # niVec
  uniqId <- unique(id)
  m <- length(uniqId)
  niVec <- sapply(1:m, function(v) return(nrow(data[which(id == uniqId[v]),])))
  n <- length(id)

  #-----------------------------------------------------------------------------
  # r, different than before
  # tprime <- time %% 28 # mod(time, 28)
  tprime <- (time %% 28)/10 # mod(time, 28)
  uniquet <- sort(unique(tprime))
  r <- length(uniquet)

  #-----------------------------------------------------------------------------
  # N1 & N2
  colIndex <- sapply(tprime, function(v) return(match(v, uniquet)))
  N <- matrix(0, n, r)
  N[cbind(1:n, colIndex)] <- 1
  N1 <- do.call(rbind, lapply(1:n, function(v) return(rbind(N[v,], rep(0, r)))))
  N2 <- do.call(rbind, lapply(1:n, function(v) return(rbind(rep(0, r), N[v,]))))

  #-----------------------------------------------------------------------------
  # K - different than that in bivFns_lik.r
  h <- c(uniquet[2:r], 28/10) - uniquet

  # testing
  # r <- 5
  # h <- c(0.5, rep(0.1,3), 0.5)

  # Q
  Q <- matrix(0, r, r)

  Q[1,1] <- -1/h[1] - 1/h[r]
  diag(Q) <- c(Q[1,1], (-1/h[1:(r-1)] - 1/h[2:r]))

  Q[1,r] <- 1/h[r]
  Q[r,1] <- Q[1,r]

  diag(Q[-r, -1]) <- 1/h[1:(r-1)]
  diag(Q[-1, -r]) <- 1/h[1:(r-1)]

  # R
  R <- matrix(0, r, r)

  R[1,1] <- (h[1] + h[r])/3
  diag(R) <- c(R[1,1], (h[1:(r-1)] + h[2:r])/3)

  R[1,r] <- h[r]/6
  R[r,1] <- R[1,r]

  diag(R[-r, -1]) <- h[1:(r-1)]/6
  diag(R[-1, -r]) <- h[1:(r-1)]/6

  K <- eigenMapMatMult3(Q, ginv(R), t(Q))

  #-----------------------------------------------------------------------------
  # B
  B <- Q %*% ginv(crossprod(Q)) %*% t(chol(R))

  # LTranspose <- chol(K, pivot=T, LDL = T) # pivot = T to handle positive-semi-definite
```

```
  # L <- t(LTranspose)
  # B <- L %*% ginv(crossprod(L))

  # B1* B2*, TO BE USED IN findPar, SCORE
  B1dou <- tcrossprod(N1 %*% B)
  B2dou <- tcrossprod(N2 %*% B)

  #------------------------------------------------------------------------
  # Y
  Y <- matrix(as.vector(rbind(response[,1], response[,2])))

  #------------------------------------------------------------------------
  # X
  fixed <- as.matrix(fixed)
  nX <- ncol(fixed)
  X1 <- do.call(rbind, lapply(1:n, function(v) return(rbind(fixed[v,], rep(0, nX)))))
  X2 <- do.call(rbind, lapply(1:n, function(v) return(rbind(rep(0, nX), fixed[v,]))))
  X <- cbind(X1, X2)

  # RETURN
  return(list("niVec" = niVec, "K" = K, "N1" = N1, "N2" = N2, "X" = X, "Y" = Y, "r" = r, "B1dou" = B1dou, "B2dou" =
B2dou, "tprime" = tprime))
}

#--------------------------------------------------------------------------
# A HELPER FUNCTION that RETURNS Vi
# ni is the number of observations for subject i
matrixVi <- function(theta, ni, ti, process) {

  # VARIANCE for bi
  Zi <- cbind(rep(c(1, 0), ni), rep(c(0, 1), ni))
  D <- matrix(c(theta[3], theta[4], theta[4], theta[5]), 2, 2)

  # VARIANCE OF MEASUREMENT ERROR
  sigma_i <- diag(rep(c(theta[6], theta[7]), ni))

  # VARIANCE FOR GAUSSIAN FIELD
  if (process == "OU") {
    G1 <- outer(ti, ti, ouVar, theta[8], theta[9])
    G2 <- outer(ti, ti, ouVar, theta[10], theta[11])
    Gammai <- GammaiFn(G1, G2, ni)
  } else if (process == "NOU") {
    G1 <- outer(ti, ti, nouVar, theta[8], theta[9], theta[10], theta[11])
    G2 <- outer(ti, ti, nouVar, theta[12], theta[13], theta[14], theta[15])
    Gammai <- GammaiFn(G1, G2, ni)
  } else if (process == "Wiener") {
    G1 <- sapply(ti, function(y) return(sapply(ti, function(x) return(wVar(x, y, theta[8])))))
    G2 <- sapply(ti, function(y) return(sapply(ti, function(x) return(wVar(x, y, theta[9])))))
    Gammai <- GammaiFn(G1, G2, ni)
  }
  Vi <- Zi %*% D %*% t(Zi) + sigma_i + Gammai
  return(list("Vi" = Vi, "D" = D, "Gammai" = Gammai, "Zi" = Zi))
}

#--------------------------------------------------------------------------
# HELPER FUNCTIONS that RETURN COVARIANCE FUNCTION for GAUSSIAN FIELDS
# OU
# ouVar <- function(x, y, theta12, theta13, theta22, theta23) {
# ouV1 <- (theta13^2/(2*theta12)) * exp(-theta12*abs(x-y))
# ouV2 <- (theta23^2/(2*theta22)) * exp(-theta22*abs(x-y))
# offDiag <- (theta13^2/(2*theta12)) * (theta23^2/(2*theta22))
# return(matrix(c(ouV1, offDiag, offDiag, ouV2), 2, 2))
# }
ouVar <- function(x, y, theta12, theta13) {
  ouV1 <- (theta13^2/(2*theta12)) * exp(-theta12*abs(x-y))
  return(ouV1)
}

# NOU
nouVar <- function(x, y, rho, a0, a1, a2) {
  logrho <- log(rho)
  avg <- (a1*(s(28/30, x) + s(28/30, y)) + a2*(s(56/30, x) + s(56/30, y)))/2
  nouV <- exp(logrho*abs(x-y) + a0 + avg)
  return(nouV)
}

# Wiener
wVar <- function(x, y, xi) {
  wV <- xi * min(x, y)
  return(wV)
}

# HELPER FUNCTION THAT RETURNS 2*ni by 2*ni VARIANCE FUNCTION, BASED ON ni by ni VARIANCE MATRIX
GammaiFn <- function(G1, G2, ni) {
  # Merge matrices
```

```
  G1offDiag <- matrix(rbind(G1, matrix(0, ni, ni)), nrow = ni)
  G2offDiag <- matrix(rbind(matrix(0, ni, ni), G2), nrow = ni)

  # Gammai
  Gi <- rbind(G1offDiag, G2offDiag)
  Gi <- Gi[c(matrix(1:nrow(Gi), nrow = 2, byrow = T)),]
  return(Gi)
}

#-------------------------------------------------------------------------------
# A HELPER FUNCTION that RETURNS TRUE IF ALL SUBJ HAVE THE SAME NUMBER OF OBSERVATIONS
test1 <- function(id, time) {
  for (i in 1:(max(id)-1)) {
    if (length(time[which(id== i)]) != length(time[which(id == i+1)])) {
      return(FALSE)
    }
  }
  return(TRUE)
}

#-------------------------------------------------------------------------------
# A HELPER FUNCTION that RETURNS TRUE IF ALL SUBJ HAVE THE SAME OBSERVATION TIME POINT
test2 <- function(id, time) {
  for (i in 1:(max(id)-1)) {
    if (all(time[which(id == i)] != time[which(id == i+1)])) {
      return(FALSE)
    }
  }
  return(TRUE)
}

#-------------------------------------------------------------------------------
# A FUNCTION THAT RETURNS PARAMETER ESTIMATES, GIVEN VARIANCE COMPONENTS
est <- function(theta, X, Y, N1, N2, K, niVec, r, time, id, process) {
  m <- length(unique(id))

  #-----------------------------------------------------------------------------
  # VARIANCE MATRIX

  if (test1(id, time) && test2(id, time)){ # IF ALL SUBJ HAVE THE SAME OBSN TIME PT
    varMatrix <- matrixVi(theta, ni = niVec[1], ti = time[which(id == 1)], process)
    Vi <- varMatrix$Vi
    if(any(is.infinite(Vi))) {
      return(NA)
    }
    Wi <- ginv(Vi)
    V <- do.call("adiag", replicate(m, Vi, simplify=F))
    W <- do.call("adiag", replicate(m, Wi, simplify=F))
    D <- do.call("adiag", replicate(m, varMatrix$D, simplify=F))
    Z <- do.call("adiag", replicate(m, varMatrix$Zi, simplify=F))
    Gamma <- do.call("adiag", replicate(m, varMatrix$Gammai, simplify=F))
  } else {
    uniqId <- unique(id)
    Vi <- lapply(1:m, function(v) return(matrixVi(theta, niVec[v], time[which(id == uniqId[v])], process)$Vi))
    Zi <- lapply(1:m, function(v) return(matrixVi(theta, niVec[v], time[which(id == uniqId[v])], process)$Zi))
    singD <- matrixVi(theta, ni = niVec[1], ti = time[which(id == uniqId[1])], process)$D
    D <- do.call("adiag", replicate(m, singD, simplify=F))
    Gammai <- lapply(1:m, function(v) return(matrixVi(theta, niVec[v], time[which(id == uniqId[v])],
process)$Gammai))
    # if(any(is.infinite(Vi))) {
    # return(NA)
    # }
    Wi <- lapply(1:m, function(v) return(ginv(Vi[[v]])))
    V <- Reduce(adiag, Vi)
    W <- Reduce(adiag, Wi)
    Z <- Reduce(adiag, Zi)
    Gamma <- Reduce(adiag, Gammai)
  }

  #-----------------------------------------------------------------------------
  # INVERSE of MATRIX C

  WX <- W %*% X
  WN1 <- eigenMapMatMult(W, N1)
  WN2 <- eigenMapMatMult(W, N2)

  lambda1 = 1/theta[1]
  lambda2 = 1/theta[2]

  # The 1st row of matrix C
  C11 <- crossprod(X, WX)
  C12 <- crossprod(X, WN1)
  C13 <- crossprod(X, WN2)
  CRow1 <- cbind(C11, C12, C13)
  # The 2nd row of matrix C
```

```
  C21 <- crossprod(N1, WX)
  C22 <- crossprod(N1, WN1) + lambda1 * K
  C23 <- crossprod(N1, WN2)
  CRow2 <- cbind(C21, C22, C23)
  # The 2nd row of matrix C
  C31 <- crossprod(N2, WX)
  C32 <- crossprod(N2, WN1)
  C33 <- crossprod(N2, WN2) + lambda2 * K
  CRow3 <- cbind(C31, C32, C33)
  # Inverse of coefficient matrix
  C <- rbind(CRow1, CRow2, CRow3)
  invC <- ginv(C)

  WY <- W %*% Y
  temp <- rbind(crossprod(X, WY), crossprod(N1, WY), crossprod(N2, WY))

  #---------------------------------------------------------------------------
  # ESTIMATES FOR beta, f1 AND f2, TO BE USED IN SCORE IN FISHER-SCORING
  res <- invC %*% temp

  nX <- ncol(X) # X is after transformation of fixed
  betaHat <- res[1:nX]
  f1Hat <- matrix(res[(nX+1):(nX+r)])
  f2Hat <- matrix(res[(nX+r+1):(nX+2*r)])

  #---------------------------------------------------------------------------
  # Likelihood function values using new parameter estimates
  meanComp <- X %*% betaHat + N1 %*% f1Hat + N2 %*% f2Hat
  resLik <- Y - meanComp
  py <- W %*% resLik
  lik <- (-log(det(V)) - t(resLik) %*% py - lambda1 * t(f1Hat) %*% K %*% f1Hat - lambda2 * t(f2Hat) %*% K %*%
f2Hat)/2
  #---------------------------------------------------------------------------
  # Estimations of b, U

  n <- sum(niVec)
  temp <- D %*% t(Z)
  b <- eigenMapMatMult3(temp, W, resLik)

  U <- eigenMapMatMult3(Gamma, W, resLik)

  #---------------------------------------------------------------------------
  # Residuals and fitted values

  fitted <- meanComp + Z %*% b + U
  residual <- Y - fitted

  estimates <- list("betaHat" = betaHat, "f1Hat" = f1Hat, "f2Hat" = f2Hat, "WN1" = WN1, "WN2" = WN2, "C22" = C22,
"C33" = C33, "invC" = invC, "V" = V, "W" = W, "py" = py, "lik" = lik, "b" = b, "U" = U, "fitted" = fitted, "residual"
= residual)
  return(estimates)
}

#==================================================================================
# FISHER-SCORING
#==================================================================================
#----------------------------------------------------------------------------
# HELPER FUNCTIONS that RETURN PARTIAL DERIVATIVES OF COVARIANCE FUNCTIONS

# OU PROCESS
ouPv1 <- function(x, y, theta2, theta3) { # THETA2
  ouP1 <- (-1/theta2 - abs(x-y)) * ouVar(x, y, theta2, theta3)
  return(ouP1)
}
ouPv2 <- function(x, y, theta2, theta3) { # THETA3
  ouP2 <- (theta3/theta2) * exp(-theta2*abs(x-y))
  return(ouP2)
}

# NOU PROCESS
nouPv1 <- function(x, y, rho, a0, a1, a2){ # rho
  pv1 <- (abs(x-y)/rho) * nouVar(x, y, rho, a0, a1, a2)
  return(pv1)
}
nouPv2 <- function(x, y, rho, a0, a1, a2){ # a0
  pv2 <- nouVar(x, y, rho, a0, a1, a2)
  return(pv2)
}
nouPv3 <- function(x, y, rho, a0, a1, a2){ # a1
  pv3 <- ((s(28/30,x) + s(28/30,y))/2) * nouVar(x, y, rho, a0, a1, a2)
  return(pv3)
}
nouPv4 <- function(x, y, rho, a0, a1, a2){ # a2
  pv4 <- ((s(56/30,x) + s(56/30,y))/2) * nouVar(x, y, rho, a0, a1, a2)
  return(pv4)
```

```
}

# Wiener process
wPv <- function(x, y) { # sigma
  pv <- min(x, y)
  return(pv)
}

#---------------------------------------------------------------------------
# A HELPER FUNCTION that RETURNS PVi, TO BE USED IN findPar()
PVi <- function(theta, ni, ti, process) {

  Zi <- cbind(rep(c(1, 0), ni), rep(c(0, 1), ni))
  tZi <- t(Zi)

  pV_phi1i <- Zi %*% matrix(c(1, 0, 0, 0), 2, 2) %*% tZi
  pV_phi2i <- Zi %*% matrix(c(0, 1, 1, 0), 2, 2) %*% tZi
  pV_phi3i <- Zi %*% matrix(c(0, 0, 0, 1), 2, 2) %*% tZi

  pV_sigma1i <- diag(rep(c(1, 0), ni))
  pV_sigma2i <- diag(rep(c(0, 1), ni))

  zeroMat <- matrix(0, ni, ni)
  if (process == "OU") {
    parV_theta12i <- outer(ti, ti, ouPv1, theta[8], theta[9])
    parV_theta13i <- outer(ti, ti, ouPv2, theta[8], theta[9])
    pv_theta12i <- GammaiFn(parV_theta12i, zeroMat, ni)
    pv_theta13i <- GammaiFn(parV_theta13i, zeroMat, ni)

    parV_theta22i <- outer(ti, ti, ouPv1, theta[10], theta[11])
    parV_theta23i <- outer(ti, ti, ouPv2, theta[10], theta[11])
    pv_theta22i <- GammaiFn(zeroMat, parV_theta22i, ni)
    pv_theta23i <- GammaiFn(zeroMat, parV_theta23i, ni)

    pvlist <- list("pV_phi1i" = pV_phi1i, "pV_phi2i" = pV_phi2i, "pV_phi3i" = pV_phi3i, "pV_sigma1i" = pV_sigma1i,
"pV_sigma2i" = pV_sigma2i, "pV_theta12i" = pv_theta12i, "pV_theta13i" = pv_theta13i, "pV_theta22i" = pv_theta22i,
"pV_theta23i" = pv_theta23i)
    return(pvlist)
  } else if (process == "NOU") {
    rho1 <- outer(ti, ti, nouPv1, theta[8], theta[9], theta[10], theta[11])
    a10 <- outer(ti, ti, nouPv2, theta[8], theta[9], theta[10], theta[11])
    a11 <- outer(ti, ti, nouPv3, theta[8], theta[9], theta[10], theta[11])
    a12 <- outer(ti, ti, nouPv4, theta[8], theta[9], theta[10], theta[11])
    pv_1rho <- GammaiFn(rho1, zeroMat, ni)
    pv_1a0 <- GammaiFn(a10, zeroMat, ni)
    pv_1a1 <- GammaiFn(a11, zeroMat, ni)
    pv_1a2 <- GammaiFn(a12, zeroMat, ni)

    rho2 <- outer(ti, ti, nouPv1, theta[12], theta[13], theta[14], theta[15])
    a20 <- outer(ti, ti, nouPv2, theta[12], theta[13], theta[14], theta[15])
    a21 <- outer(ti, ti, nouPv3, theta[12], theta[13], theta[14], theta[15])
    a22 <- outer(ti, ti, nouPv4, theta[12], theta[13], theta[14], theta[15])
    pv_2rho <- GammaiFn(zeroMat, rho2, ni)
    pv_2a0 <- GammaiFn(zeroMat, a20, ni)
    pv_2a1 <- GammaiFn(zeroMat, a21, ni)
    pv_2a2 <- GammaiFn(zeroMat, a22, ni)

    pvlist <- list("pV_phi1i" = pV_phi1i, "pV_phi2i" = pV_phi2i, "pV_phi3i" = pV_phi3i, "pV_sigma1i" = pV_sigma1i,
"pV_sigma2i" = pV_sigma2i, "pv_1rho" = pv_1rho, "pv_1a0" = pv_1a0, "pv_1a1" = pv_1a1, "pv_1a2" = pv_1a2, "pv_2rho" =
pv_2rho, "pv_2a0" = pv_2a0, "pv_2a1" = pv_2a1, "pv_2a2" = pv_2a2)
    return(pvlist)
  } else if (process == "Wiener") {
    sigma_w <- sapply(ti, function(y) return(sapply(ti, function(x) return(wPv(x, y)))))
    pv_w1 <- GammaiFn(sigma_w, zeroMat, ni)
    pv_w2 <- GammaiFn(zeroMat, sigma_w, ni)

    pvlist <- list("pV_phi1i" = pV_phi1i, "pV_phi2i" = pV_phi2i, "pV_phi3i" = pV_phi3i, "pV_sigma1i" = pV_sigma1i,
"pV_sigma2i" = pV_sigma2i, "pv_w1" = pv_w1, "pv_w2" = pv_w2)
    return(pvlist)
  }
}
#---------------------------------------------------------------------------
# A HELPER FUNCTION THAT RETURNS A NEW THETA AFTER ONE ITERATION OF FISHER-SCORING
findPar <- function(theta, X, Y, N1, N2, K, niVec, r, time, id, B1dou, B2dou, process, dim) {

  res <- est(theta, X, Y, N1, N2, K, niVec, r, time, id, process)
  if(any(is.na(res))) {
    return(NA)
  }
  invC <- res$invC
  W <- res$W
  py <- res$py
  tpy <- t(py)
  m <- length(unique(id))
```

```
  #--------------------------------------------------------------------------
  # P*
  chi <- cbind(X, N1, N2)
  Wchi <- eigenMapMatMult(W, chi)
  Pst <- W - eigenMapMatMult3(Wchi, invC, t(Wchi))

  #--------------------------------------------------------------------------
  # PARTIAL DERIVATIVES

  if (test1(id, time) && test2(id, time)){ # IF ALL SUBJ HAVE THE SAME OBSN TIME PT
    pv <- PVi(theta, niVec[1], time[which(id == 1)], process)
    pvlist1 <- lapply(1:(dim-2), function(v) return(do.call("adiag", replicate(m, pv[[v]], simplify=F))))
    pvlist <- c(list(B1dou, B2dou), pvlist1)
  } else {
    uniqId <- unique(id)
    pvAll <- lapply(1:m, function(v) return(PVi(theta, niVec[v], time[which(id == uniqId[v])], process)))
    pvlist1 <- lapply(1:(dim-2), function(u) return(Reduce(adiag, lapply(1:m, function(v) return(pvAll[[v]][[u]])))))
    pvlist <- c(list(B1dou, B2dou), pvlist1)
  }

  #--------------------------------------------------------------------------
  # SCORE

  # Pst * pv
  Pst_pv <- lapply(1:dim, function(v) return(eigenMapMatMult(Pst, pvlist[[v]])))

  # score
  score <- sapply(1:dim, function(v) return(eigenMapMatMult3(tpy, pvlist[[v]], py) - tr(Pst_pv[[v]])))

  #--------------------------------------------------------------------------
  # FISHER INFO

  fInfo <- sapply(1:dim, function(u) return(sapply(1:dim, function(v) return(tr(eigenMapMatMult(Pst_pv[[u]],
Pst_pv[[v]]))))))

  score <- score/2
  fInfo <- fInfo/2
  Finv <- ginv(fInfo)
  thetaNew <- theta + Finv %*% score    # Parameter estimate after one iteration
  return(list("theta" = thetaNew, "sdTheta" = sqrt(diag(Finv))))
}

#----------------------------------------------------------------------------
# A FUNCTION that RETURNS THETA, using FISHER-SCORING ALGORITHM
fisher.scoring <- function(theta, X, Y, N1, N2, K, niVec, r, time, id, B1dou, B2dou, tol, cap, process, dim) {

  res <- est(theta, X, Y, N1, N2, K, niVec, r, time, id, process)
  if(any(is.na(res))) {
    return(NA)
  }
  theta1 <- theta
  lik1 <- res$lik

  iternum <- 0
  diff <- 1

  while (diff > tol) {
    iternum <- iternum + 1

    theta0 <- theta1
    lik0 <- lik1

    # Update theta0 -> theta1
    theta1 <- findPar(theta0, X, Y, N1, N2, K, niVec, r, time, id, B1dou, B2dou, process, dim)$theta
    if(any(is.na(theta1))) {
      return(NA)
    }

    #------------------------------------------------------------------------
    # Check whether theta1 is in parameter space
    if (process == "OU") {
      while (any(theta1[c(1:3, 5:11)] < 0)){
        theta1 <- (theta1 + theta0)/2
      }
    } else if (process == "NOU") { # NOU case
      while (any(theta1[c(1:3, 5:8, 12)] < 0)){
        theta1 <- (theta1 + theta0)/2
      }
    } else if (process == "Wiener") {
      while (any(theta1[c(1:3, 5:9)] < 0)) {
        theta1 <- (theta1 + theta0)/2
      }
    }

    #------------------------------------------------------------------------
```

```r
    # Find lik1 using updated theta1
    res <- est(theta1, X, Y, N1, N2, K, niVec, r, time, id, process)
    if(any(is.na(res))) {
      return(NA)
    }
    lik1 <- res$lik

    #--------------------------------------------------------------------------
    # Check whether the lik is increasing.
    while(lik1 <= lik0 || is.infinite(lik1)) {
      theta1 <- (theta1 + theta0)/2
      res <- est(theta1, X, Y, N1, N2, K, niVec, r, time, id, process)
      if(any(is.na(res))) {
        return(NA)
      }
      lik1 <- res$lik
    }
    diff <- abs(lik1 - lik0)/abs(lik1)

    #--------------------------------------------------------------------------
    # Obtain standard deviation of theta using final theta
    sdThetaFinal <- findPar(theta1, X, Y, N1, N2, K, niVec, r, time, id, B1dou, B2dou, process, dim)$sdTheta

    #--------------------------------------------------------------------------
    # Abort this iteration if iternum > cap, and set the parameters to be zero
    if (iternum > cap) {
      theta1 <- matrix(0, dim)
      return(list("theta" = theta1, "capConv" = iternum))
    }
    print(iternum)
    print (diff)
    print(theta1)
  }
  return(list("theta" = theta1, "capConv" = iternum, "sdTheta" = sdThetaFinal))
}

#==========================================================================================
# MAIN FUNCTION - OUTPUTS ESTIMATES, BIASES AND COVARIANCES
#==========================================================================================
# data = NAME OF THE DATASET TO BE USED
# response = UNIVARIATE LONGITUDINAL RESPONSE, as.matrix(Y)
# fixed = FIXED EFFECT COVARIATE MATRIX, X<-cbind(X1, ..., Xp)
# random = RANDOM EFFECT COVARIATE MATRIX,
# the random effect covariance matrix is assumed to be unstructured.
# process = name of the Gaussian process, a character string.
# OU - Ornstein-Uhlenbeck process
# NOU - Nonhomogeneous OU process with log(v(t)) = a0 + a1*t + a2*t^2
# time = OBSERVATION TIME POINTS OF OBSERVATIONS
# id = IDENTIFICATION OF SUBJECTS
# tol = tolerance level for parameters.
# cap = maximum runs of iterations for FISHER-SCORING
# THE PROGRAM ONLY ALLOWS FOR RANDOM INTERCEPT
results <- function(data, response, fixed, random = cbind(1, 1), process, time, id, tol, cap) {

  #--------------------------------------------------------------------------
  # Determine dimention of parameter theta
  q <- ncol(random)
  # random effect cov matx unstructured; 4 for smoothing param(2) & measurement error(2);
  dim <- q*(q+1)/2 + 4
  if (process == "OU") {
    dim <- dim + 4
  } else if (process == "NOU") {
    dim <- dim + 8
  } else if (process == "Wiener") {
    dim <- dim + 2
  }

  #--------------------------------------------------------------------------
  # Initialize parameter theta
  theta <- matrix(0, dim)
  s <- 1 # keep track of index position of theta
  theta[s:2] <- 1 # smoothing parameter
  s <- 3
  if (q > 0) {
    for (i in 1:q){
      for (j in 1:i) {
        if (i == j) {
          theta[s] <- 1
        } else {
          theta[s] <- -0.5
        }
        s <- s + 1
      }
    }
  }
```

```r
theta[s:(s + 1)] <- 1 # measurement error
s <- s + 2
if (process == "OU") {
  # Simulation initializations - theta is initialized to ensure that rho = 0.9, var = 1
  theta[s] <- -log(0.9) # 2.340287
  theta[s+1] <- sqrt(-log(0.9)*2) # 2.163463
  theta[s+2] <- -log(0.9)
  theta[s+3] <- sqrt(-log(0.9)*2)
  # Test whether OU == NOU
  # theta[s:(s+3)] <- rep(c(-log(0.5), sqrt(-2*log(0.5)*exp(-0.3266))), 2)
} else if (process == "NOU") {
  # Simulation initializations, using estimates from liu analysis
  # theta[s:(s+7)] <- rep(c(0.1, -5, 1.5, -0.1), 2) # doesn't work on 2c
  theta[s:(s+7)] <- rep(c(0.1, -3, -5, -2), 2) # from univFns_2c simulation
  # Liu analysis initializations
  # theta[s:(s+7)] <- c(0.4, -1, 0.1, -0.1, 0.1, -2, 1, -0.1)
  # theta[s:(s+7)] <- c(0.2, -0.44, 0.3, -0.2, 0.15, -1.6, 0.3, -0.1)
  # Test whether OU == NOU
  # theta[s:(s+7)] <- rep(c(0.5, -0.3266, 0, 0), 2)
} else if (process == "Wiener") {
  theta[s:(s+1)] <- rep(0.05, 2)
}

#--------------------------------------------------------------------------
# Obtain values from helper functions
helper <- helperFn(data, time, id, fixed, response)
niVec <- helper$niVec
K <- helper$K
N1 <- helper$N1
N2 <- helper$N2
r <- helper$r
B1dou <- helper$B1dou
B2dou <- helper$B2dou
tprime <- helper$tprime

X <- helper$X
Y <- helper$Y

fs <- fisher.scoring(theta, X, Y, N1, N2, K, niVec, r, time=tprime, id, B1dou, B2dou, tol, cap, process, dim)

# variance matrix is invertible in est() function
if(any(is.na(fs))) {
  return(NA)
}

newTheta <- fs$theta

# fisher-scoring non-convergent
if (identical(newTheta, matrix(0, dim))) {
  return (fs)
}

#--------------------------------------------------------------------------
# Compute AICc, using newTheta
res <- est(newTheta, X, Y, N1, N2, K, niVec, r, time=tprime, id, process)

loglik <- res$lik
nX <- ncol(X)
numPar <- dim + nX
numObv <- nrow(Y)/2
AICc <- -2*loglik + 2*(numPar)*(numObv / (numObv - numPar - 1))
AIC <- -2*loglik + 2*(numPar)

#--------------------------------------------------------------------------
# Compute Covariance & Bias, using newTheta

WN1 <- res$WN1
WN2 <- res$WN2
V <- res$V
W <- res$W

tX <- t(X)
tN1 <- t(N1)
tN2 <- t(N2)

K1 <- (1/theta[1]) * K
K2 <- (1/theta[2]) * K

# WEIGHT MATRICES, W1 & W2
W1 <- W - eigenMapMatMult3(WN1, ginv(res$C22), t(WN1))
W2 <- W - eigenMapMatMult3(WN2, ginv(res$C33), t(WN2))

# WEIGHT MATRICES, Wx, Wf1 & Wf2
W1N2 <- eigenMapMatMult(W1, N2)
W2X <- eigenMapMatMult(W2, X)
```

```
  W1X <- eigenMapMatMult(W1, X)
  WxInv <- ginv(eigenMapMatMult3(tN2, W1, N2) + K2)
  Wf1Inv <- ginv(eigenMapMatMult3(tX, W2, X))
  Wf2Inv <- ginv(eigenMapMatMult3(tX, W1, X))
  Wx <- W1 - eigenMapMatMult3(W1N2, WxInv, t(W1N2))
  Wf1 <- W2 - eigenMapMatMult3(W2X, Wf1Inv, t(W2X))
  Wf2 <- W1 - eigenMapMatMult3(W1X, Wf2Inv, t(W1X))

  #--------------------------------------------------------------------------
  # COVARIANCE - beta
  betaInv <- ginv(eigenMapMatMult3(tX, Wx, X))
  hatB <- eigenMapMatMult3(betaInv, tX, Wx)
  covBeta <- eigenMapMatMult3(hatB, V, t(hatB))

  # COVARIANCE - f1
  tN1Wf1 <- eigenMapMatMult(tN1, Wf1)
  f1Inv <- ginv(eigenMapMatMult(tN1Wf1, N1) + K1)
  hatF1 <- eigenMapMatMult(f1Inv, tN1Wf1)
  covF1 <- eigenMapMatMult3(hatF1, V, t(hatF1))

  # COVARIANCE - f2
  tN2Wf2 <- eigenMapMatMult(tN2, Wf2)
  f2Inv <- ginv(eigenMapMatMult(tN2Wf2, N2) + K2)
  hatF2 <- eigenMapMatMult(f2Inv, tN2Wf2)
  covF2 <- eigenMapMatMult3(hatF2, V, t(hatF2))

  #--------------------------------------------------------------------------
  # BIAS FOR beta & f, FOR SIMULATION
  if (test1(id, time) && test2(id, time)) {

    # BIAS - beta
    f1 <- sine(niVec[1], time[which(id == 1)])
    f2 <- cosine(niVec[1], time[which(id == 1)])
    N1f1 <- eigenMapMatMult(N1, f1)
    N2f2 <- eigenMapMatMult(N2, f2)
    Nf <- N1f1 + N2f2
    biasBeta <- eigenMapMatMult(hatB, Nf)

    # BIAS - f1
    temp1 <- eigenMapMatMult(tN1Wf1, N2f2) - eigenMapMatMult(K1, f1)
    biasF1 <- eigenMapMatMult(f1Inv, temp1)

    # BIAS - f2
    temp2 <- eigenMapMatMult(tN2Wf2, N1f1) - eigenMapMatMult(K2, f2)
    biasF2 <- eigenMapMatMult(f2Inv, temp2)

    newlist <- list("theta" = newTheta, "betaHat" = res$betaHat, "f1Hat" = res$f1Hat, "f2Hat" = res$f2Hat, "sdBeta" =
sqrt(diag(covBeta)), "sdF1" = sqrt(diag(covF1)), "sdF2" = sqrt(diag(covF2)), "biasBeta" = biasBeta, "biasF1" =
biasF1, "biasF2" = biasF2, "conv" = fs$conv, "capConv" = fs$capConv, "loglik" = loglik, "AICc" = AICc, "AIC" = AIC,
"sdTheta" = fs$sdTheta, "b" = res$b, "U" = res$U, "fitted" = res$fitted, "trueRes" = res$trueRes, "diagV" = diag(V))
    return(newlist)
  }

  newlist <- list("theta" = newTheta, "betaHat" = res$betaHat, "f1Hat" = res$f1Hat, "f2Hat" = res$f2Hat, "sdBeta" =
sqrt(diag(covBeta)), "sdF1" = sqrt(diag(covF1)), "sdF2" = sqrt(diag(covF2)), "conv" = fs$conv, "capConv" =
fs$capConv, "loglik" = loglik, "AICc" = AICc, "AIC" = AIC, "sdTheta" = fs$sdTheta, "b" = res$b, "U" = res$U, "fitted"
= res$fitted, "trueRes" = res$trueRes, "diagV" = diag(V))
  return(newlist)
}
```