STAT547 Homework 5

Xingche Guo

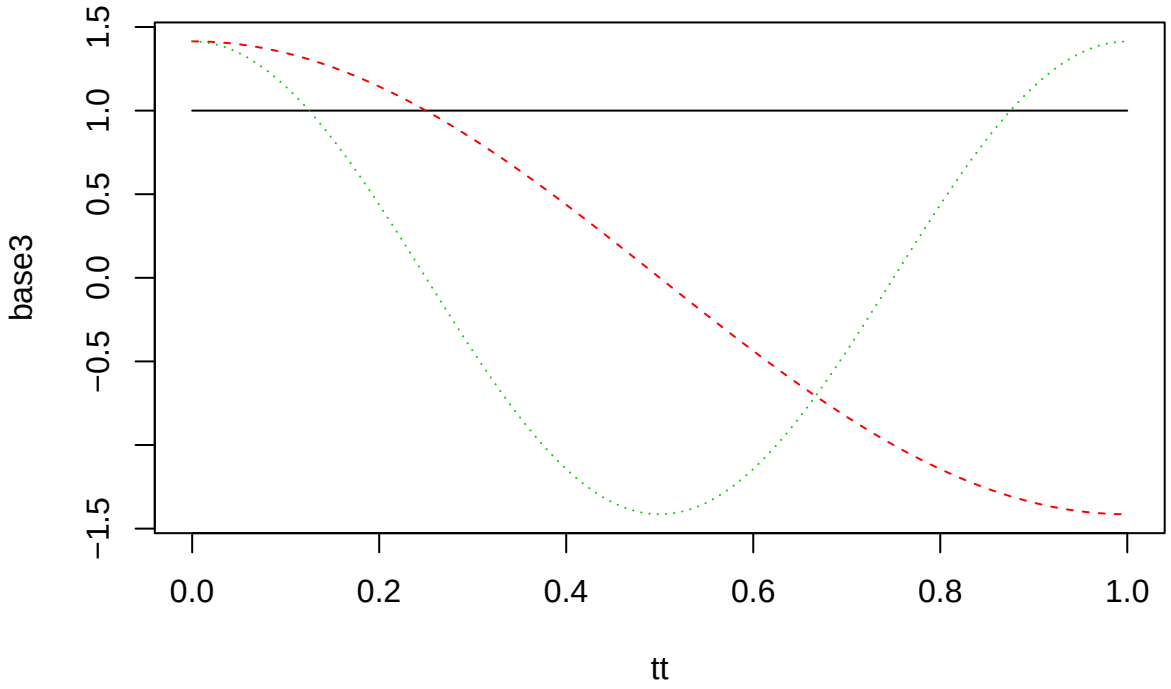
11/9/2019

Problem 1

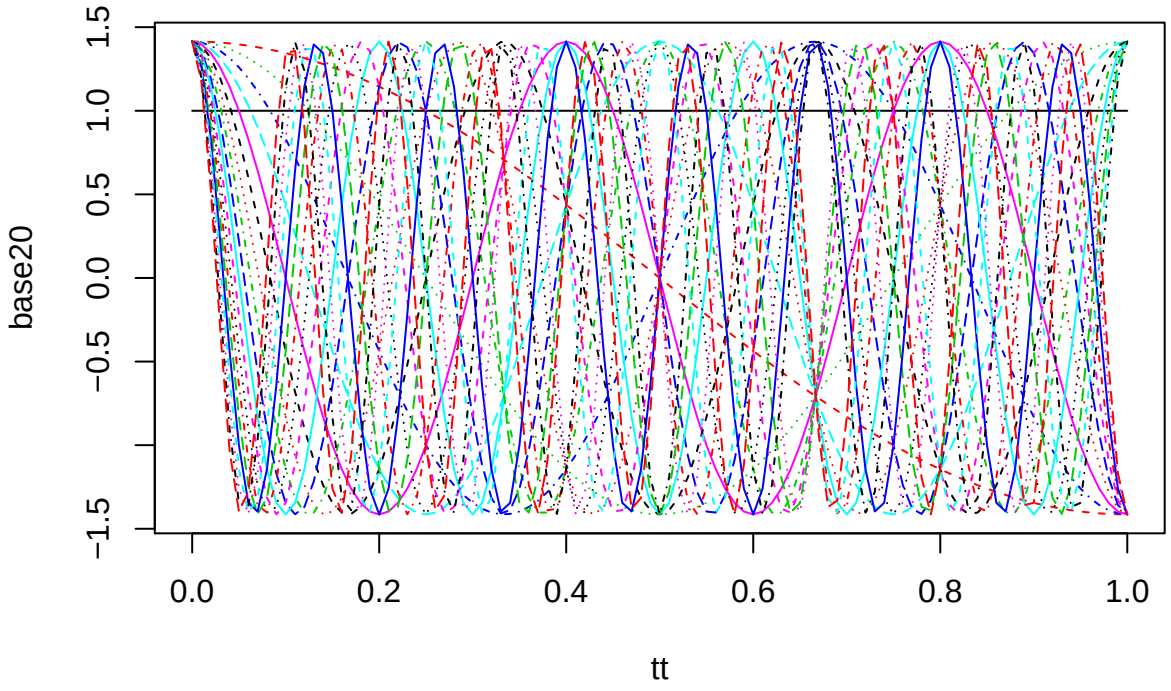
```
library(fdapace)
library(ggplot2)
set.seed(123123)

############## functions ###############
## plot sparse (raw) functional data
plot.spar.fdata <- function(Lt, Ly){
  n <- length(Lt)
  x <- unlist(Lt)
  y <- unlist(Ly)
  count <- unlist(lapply(Lt, length))
  label <- as.factor(rep(1:n, times = count))
  DD <- data.frame(x = x, y = y, label = label)
  p <- ggplot(data = DD, aes(x = x, y = y, group = label)) +
    geom_line(size = 0.5, alpha = 0.8) +
    geom_point(colour = "blue", alpha = 0.25) +
    xlab("t") +
    ylab("y") +
    theme_bw()
  print(p)
}

## evaluate 3 Methods for selecting K (for dense data).
den.model.eval.K <- function(fdata){
  input <- MakeFPCAInputs(tVec = fdata$pts, yVec = fdata$Y)
  fit1 <- FPCA(Ly = input$Ly, Lt = input$Lt,
               optns = list(methodSelectK = "FVE",
                            methodXi = "IN"))
  fit2 <- FPCA(Ly = input$Ly, Lt = input$Lt,
               optns = list(methodSelectK = "AIC",
                            methodXi = "IN"))
  fit3 <- FPCA(Ly = input$Ly, Lt = input$Lt,
               optns = list(methodSelectK = "BIC",
                            methodXi = "IN"))
  K1 <- fit1$selectK
  K2 <- fit2$selectK
  K3 <- fit3$selectK
  cat("FVE selecting: K=", K1, "; ",
      "AIC selecting: K=", K2, "; ",
      "BIC selecting: K=", K3, ".\n")
}

## evaluate 3 Methods for selecting K (for sparse data).
spar.model.eval.K <- function(fdata){
```

```
  fit1 <- FPCA(Ly = fdata$Ly, Lt = fdata$Lt,
               optns = list(methodSelectK = "FVE",
                            methodXi = "CE"))
  fit2 <- FPCA(Ly = fdata$Ly, Lt = fdata$Lt,
               optns = list(methodSelectK = "AIC",
                            methodXi = "CE"))
  fit3 <- FPCA(Ly = fdata$Ly, Lt = fdata$Lt,
               optns = list(methodSelectK = "BIC",
                            methodXi = "CE"))
  K1 <- fit1$selectK
  K2 <- fit2$selectK
  K3 <- fit3$selectK
  cat("FVE selecting: K=", K1, "; ",
      "AIC selecting: K=", K2, "; ",
      "BIC selecting: K=", K3, ".\n")
}

## evaluate 2 Methods for computing FPCs (for dense data)
den.model.eval.xi <- function(fdata){
  input <- MakeFPCAInputs(tVec = fdata$pts, yVec = fdata$Y)
  fit1 <- FPCA(Ly = input$Ly, Lt = input$Lt,
               optns = list(methodSelectK = "FVE",
                            methodXi = "IN"))
  fit2 <- FPCA(Ly = input$Ly, Lt = input$Lt,
               optns = list(methodSelectK = "FVE",
                            methodXi = "CE"))
  par(mfrow = c(2,3))
  plot(fit1$xiEst[,1], fit2$xiEst[,1], xlab = "Integration method",
       ylab = "BLUP", main = "1st FPC")
  plot(fit1$xiEst[,2], fit2$xiEst[,2], xlab = "Integration method",
       ylab = "BLUP", main = "2nd FPC")
  plot(fit1$xiEst[,3], fit2$xiEst[,3], xlab = "Integration method",
       ylab = "BLUP", main = "3rd FPC")
  matplot(fit1$workGrid, fit1$phi[,1:3], type = "l", xlab = "t", ylab = "First 3 Phi",
          main = "Integration method")
  matplot(fit2$workGrid, fit2$phi[,1:3], type = "l",  xlab = "t", ylab = "First 3 Phi",
          main = "BLUP")
  par(mfrow = c(1,1))
}

## evaluate 2 Methods for computing FPCs (for sparse data)
spar.model.eval.xi <- function(fdata){
  fit <- FPCA(Ly = fdata$Ly, Lt = fdata$Lt,
               optns = list(methodSelectK = "AIC",
                            methodXi = "CE"))

  par(mfrow = c(2,2))
  plot(fdata$xi[,1], fit$xiEst[,1], xlab = "TRUE",
       ylab = "BLUP", main = "1st FPC")
  plot(fdata$x[,2], fit$xiEst[,2], xlab = "TRUE",
       ylab = "BLUP", main = "2nd FPC")
  plot(fdata$x[,3], fit$xiEst[,3], xlab = "TRUE",
       ylab = "BLUP", main = "3rd FPC")
```

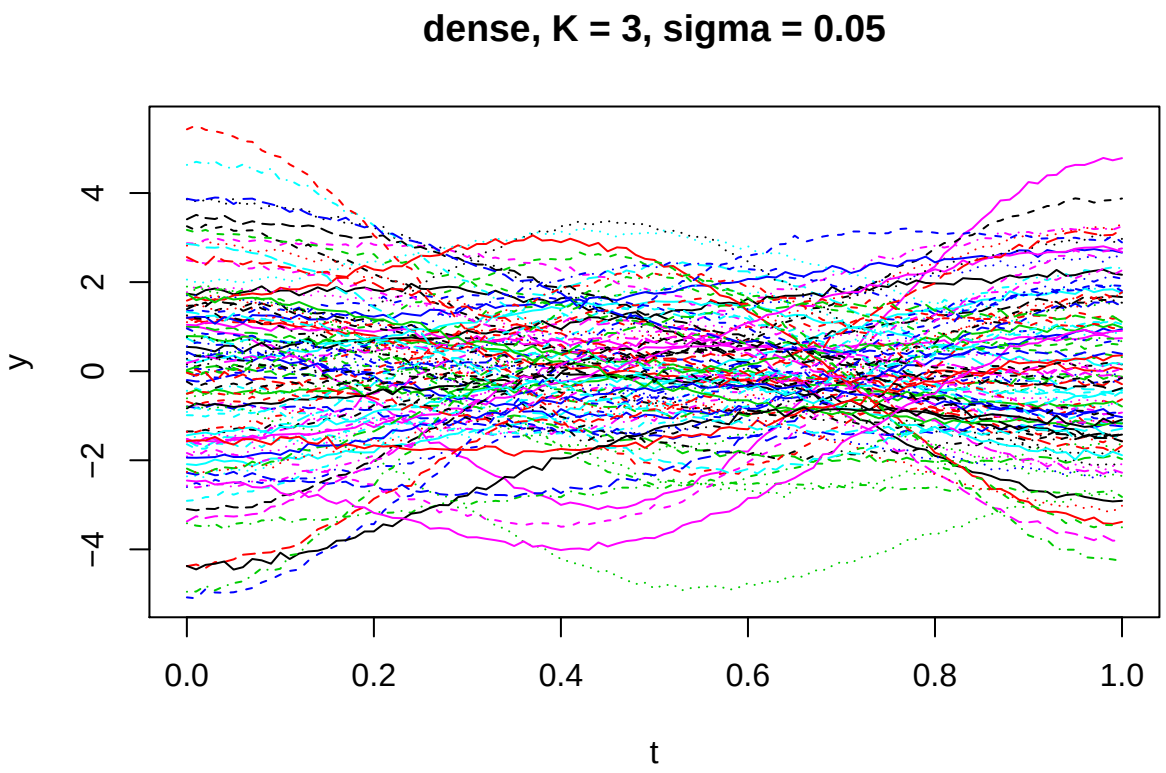
```
  par(mfrow = c(1,1))
}

########### check basis ##################
tt <- seq(0, 1, length.out = 101)
base3 <- CreateBasis(K = 3, pts = tt)
base20 <- CreateBasis(K = 20, pts = tt)
matplot(tt, base3, type = "l")
```



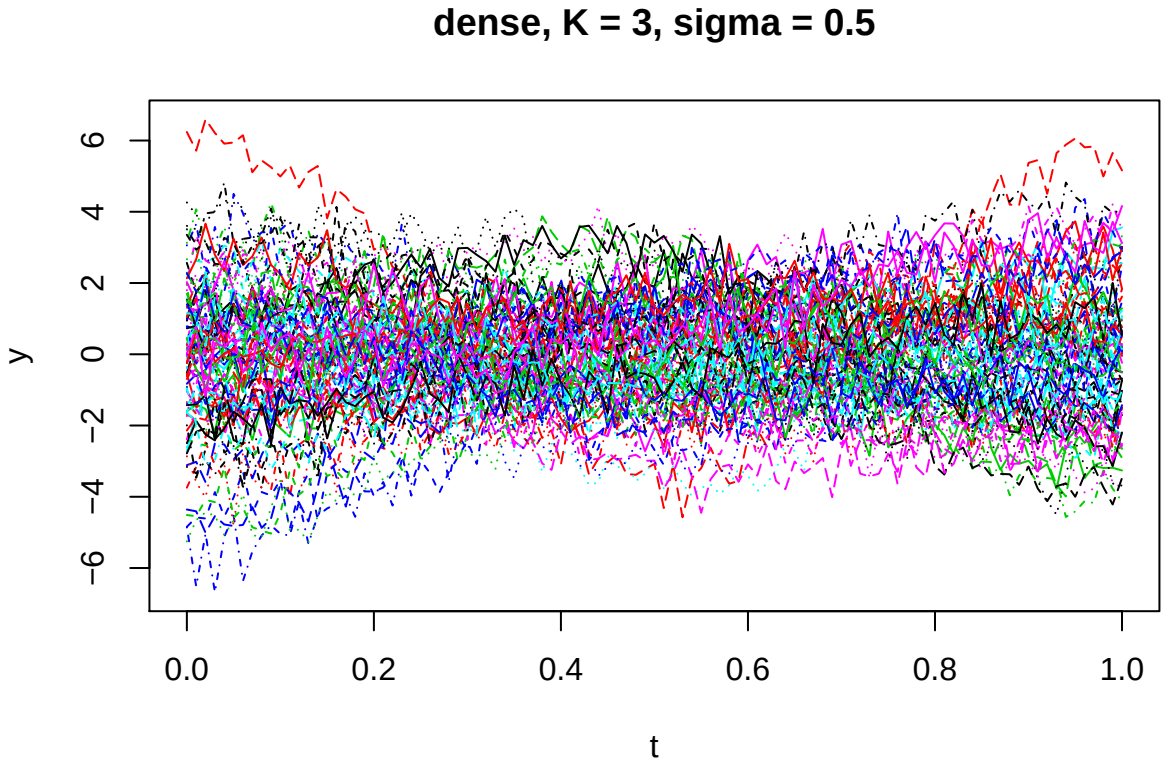

```
matplot(tt, base20, type = "l")
```



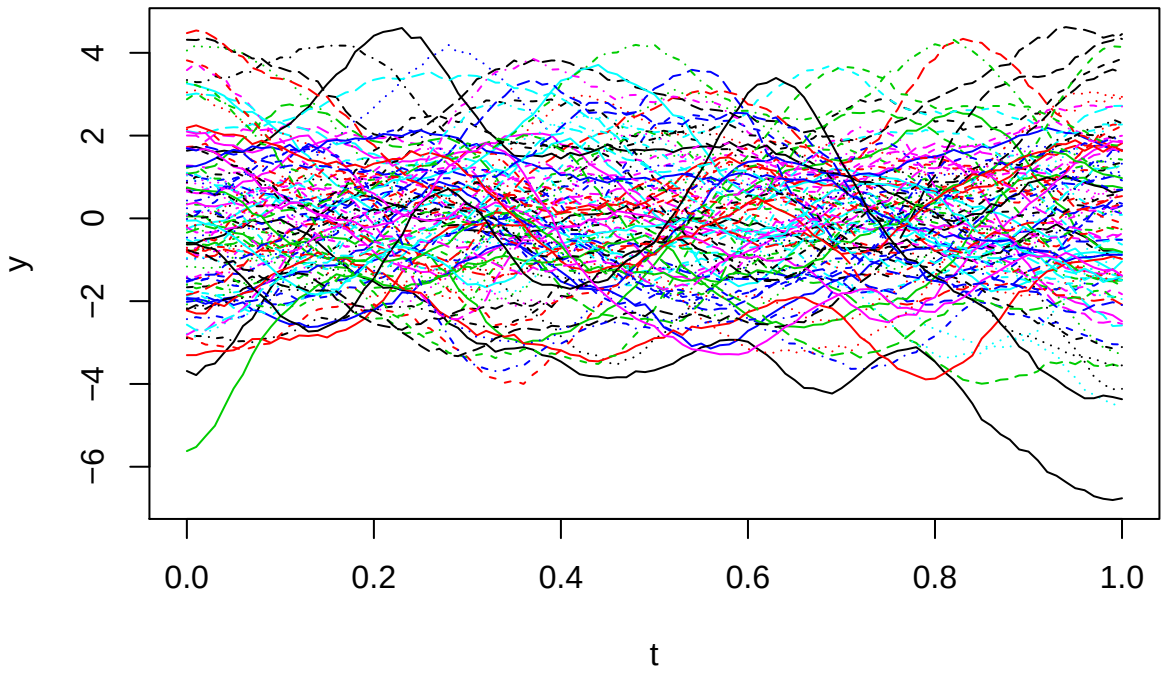
```
########### generate dense data ###############
## K = 3, sigma = 0.05
fdata_den_3_1 <- MakeGPFunctionalData(n = 100, M = 101, K = 3, sigma = 0.05,
                                      lambda = c(1, 0.8, 0.6))
## K = 3, sigma = 0.5
fdata_den_3_2 <- MakeGPFunctionalData(n = 100, M = 101, K = 3, sigma = 0.5,
                                      lambda = c(1, 0.8, 0.6))
## K = 20, sigma = 0.05
fdata_den_20_1 <- MakeGPFunctionalData(n = 100, M = 101, K = 20, sigma = 0.05,
                                      lambda = exp(-(1/2)*(0:19)) )
## K = 20, sigma = 0.5
fdata_den_20_2 <- MakeGPFunctionalData(n = 100, M = 101, K = 20, sigma = 0.5,
                                      lambda = exp(-(1/2)*(0:19)) )


matplot(tt, t(fdata_den_3_1$Yn), type = "l", xlab = "t", ylab = "y",
        main = "dense, K = 3, sigma = 0.05")
```



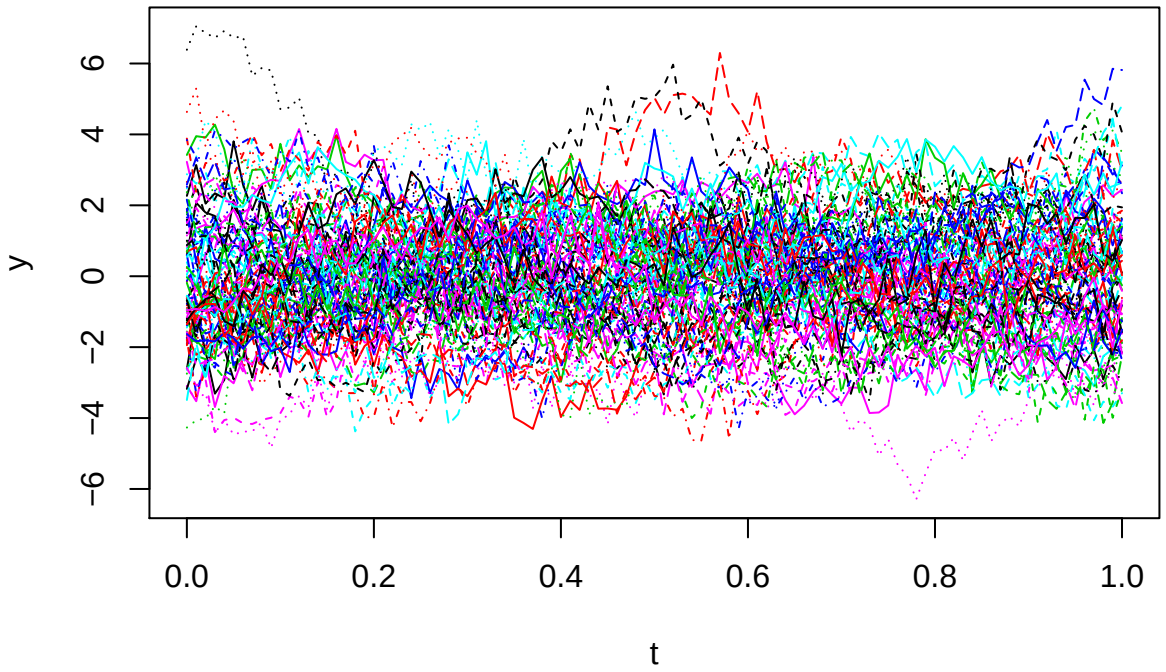

```
matplot(tt, t(fdata_den_3_2$Yn), type = "l", xlab = "t", ylab = "y",
        main = "dense, K = 3, sigma = 0.5")
```



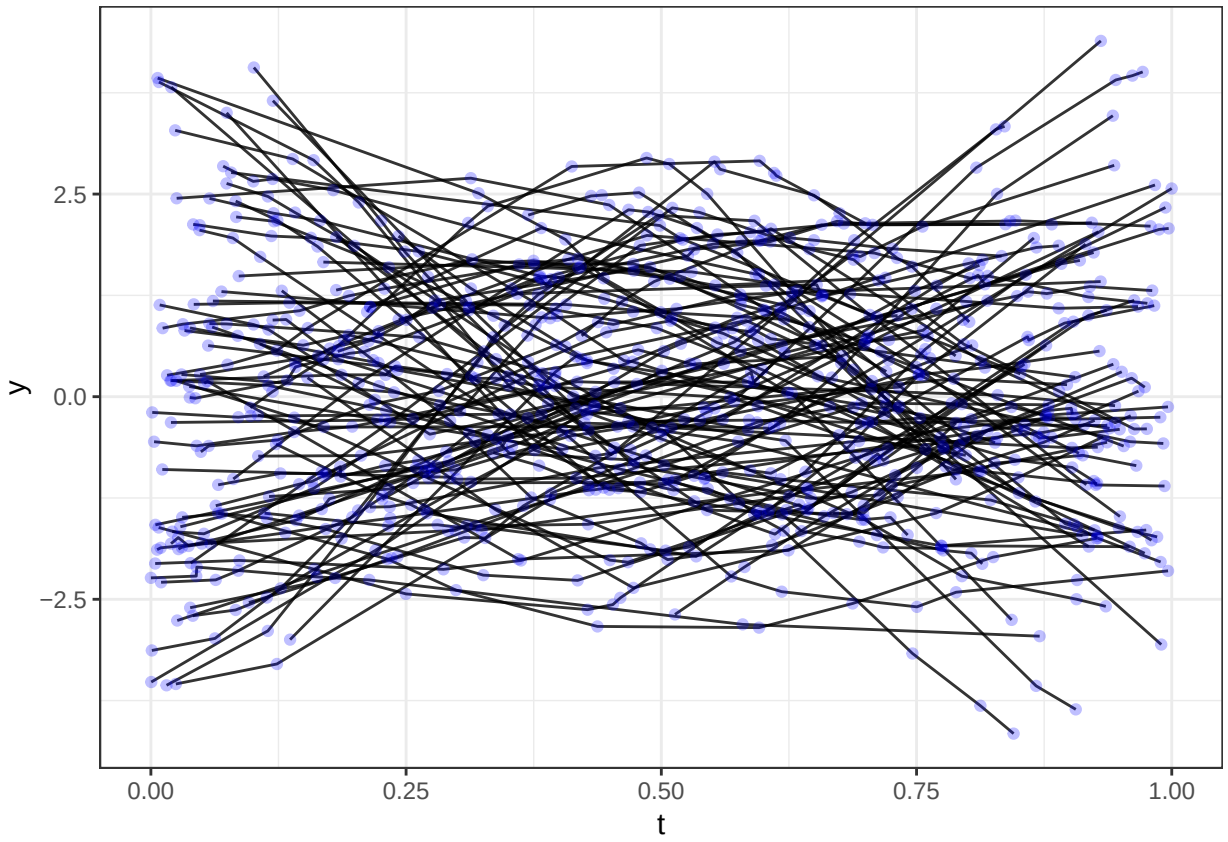

```
matplot(tt, t(fdata_den_20_1$Yn), type = "l", xlab = "t", ylab = "y",
        main = "dense, K = 20, sigma = 0.05")
```



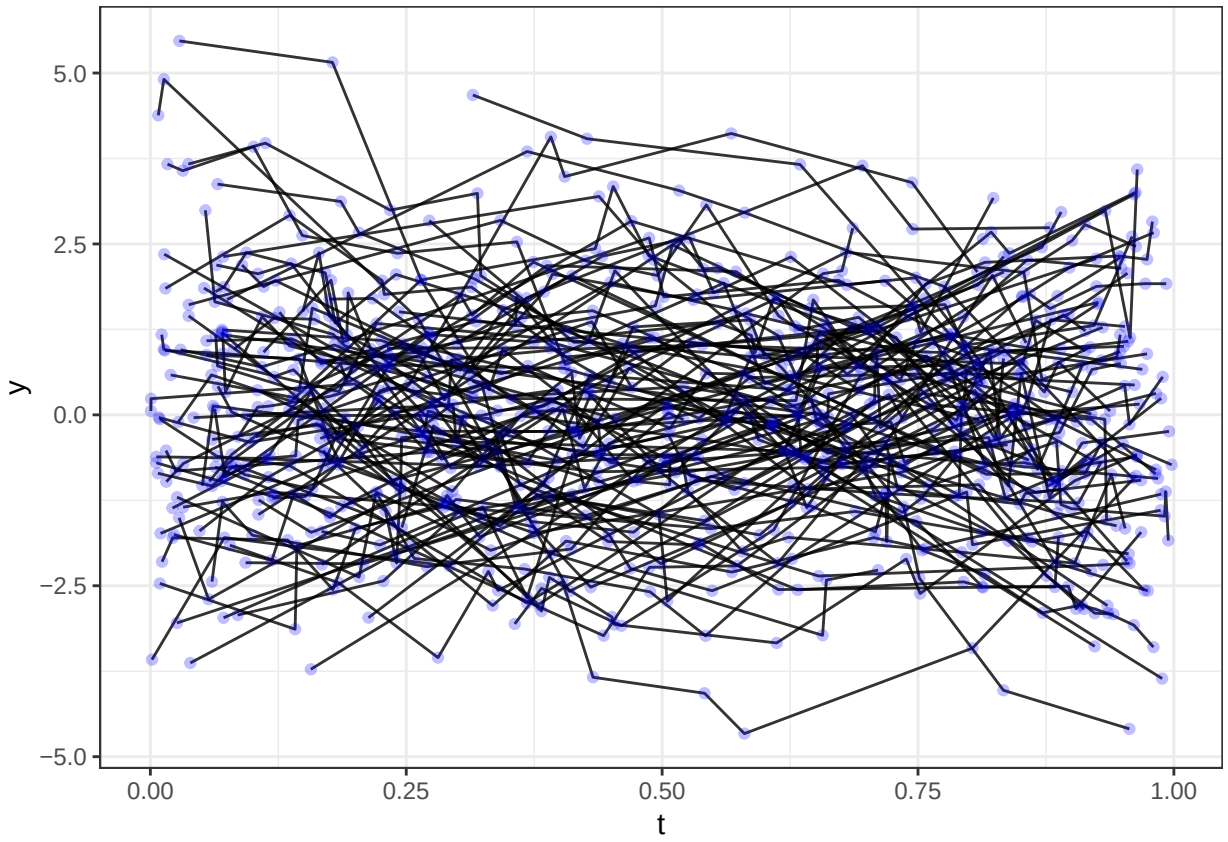

```
matplot(tt, t(fdata_den_20_2$Yn), type = "l", xlab = "t", ylab = "y",
        main = "dense, K = 20, sigma = 0.5")
```



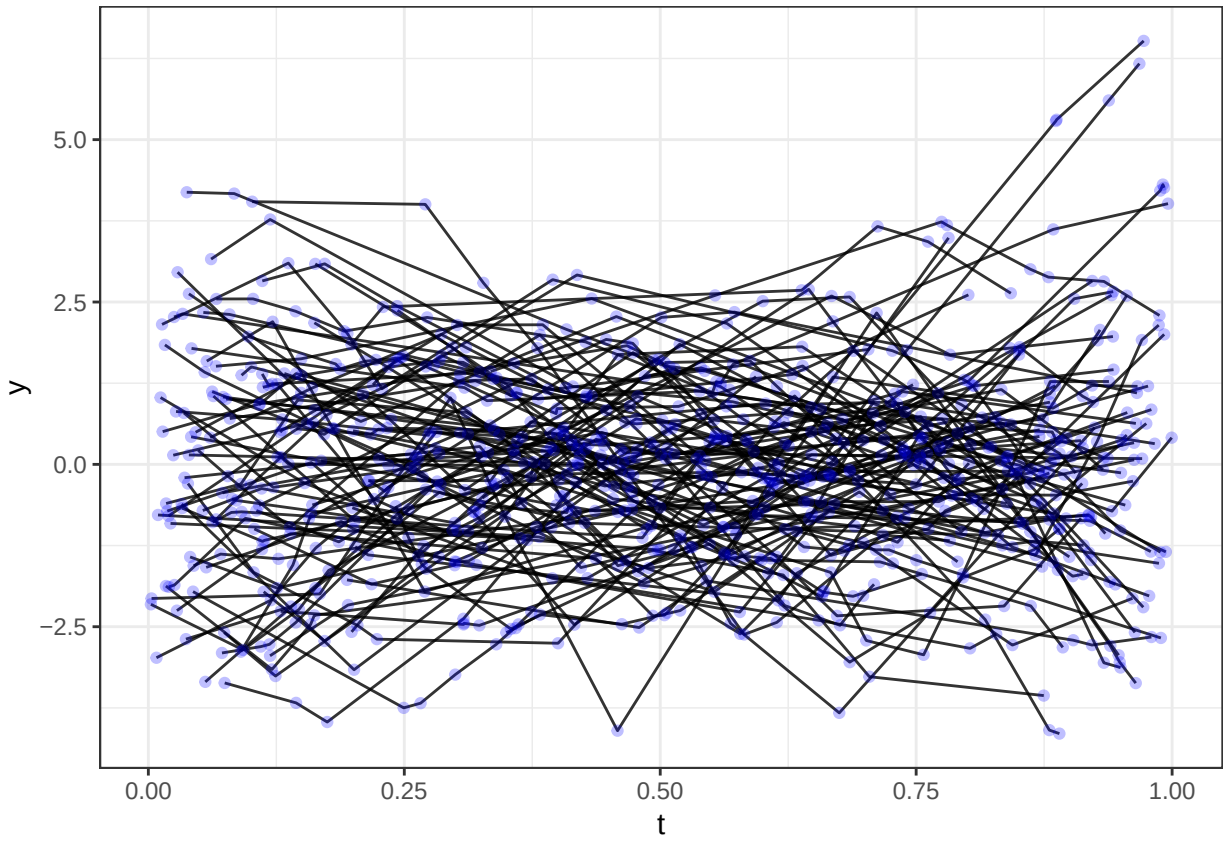

```
############ generate sparse data ################
## K = 3, sigma = 0.05
fdata_spar_3_1 <- MakeSparseGP(n = 100, sparsity = 6:12, K = 3,
                               lambda = c(1, 0.8, 0.6), sigma = 0.05)
## K = 3, sigma = 0.5
fdata_spar_3_2 <- MakeSparseGP(n = 100, sparsity = 6:12, K = 3,
                               lambda = c(1, 0.8, 0.6), sigma = 0.5)
## K = 20, sigma = 0.05
fdata_spar_20_1 <- MakeSparseGP(n = 100, sparsity = 6:12, K = 20,
                               lambda =  exp(-(1/2)*(0:19)), sigma = 0.05)
## K = 20, sigma = 0.5
fdata_spar_20_2 <- MakeSparseGP(n = 100, sparsity = 6:12, K = 20,
                               lambda =  exp(-(1/2)*(0:19)), sigma = 0.5)



plot.spar.fdata(fdata_spar_3_1$Lt, fdata_spar_3_1$Ly)
```



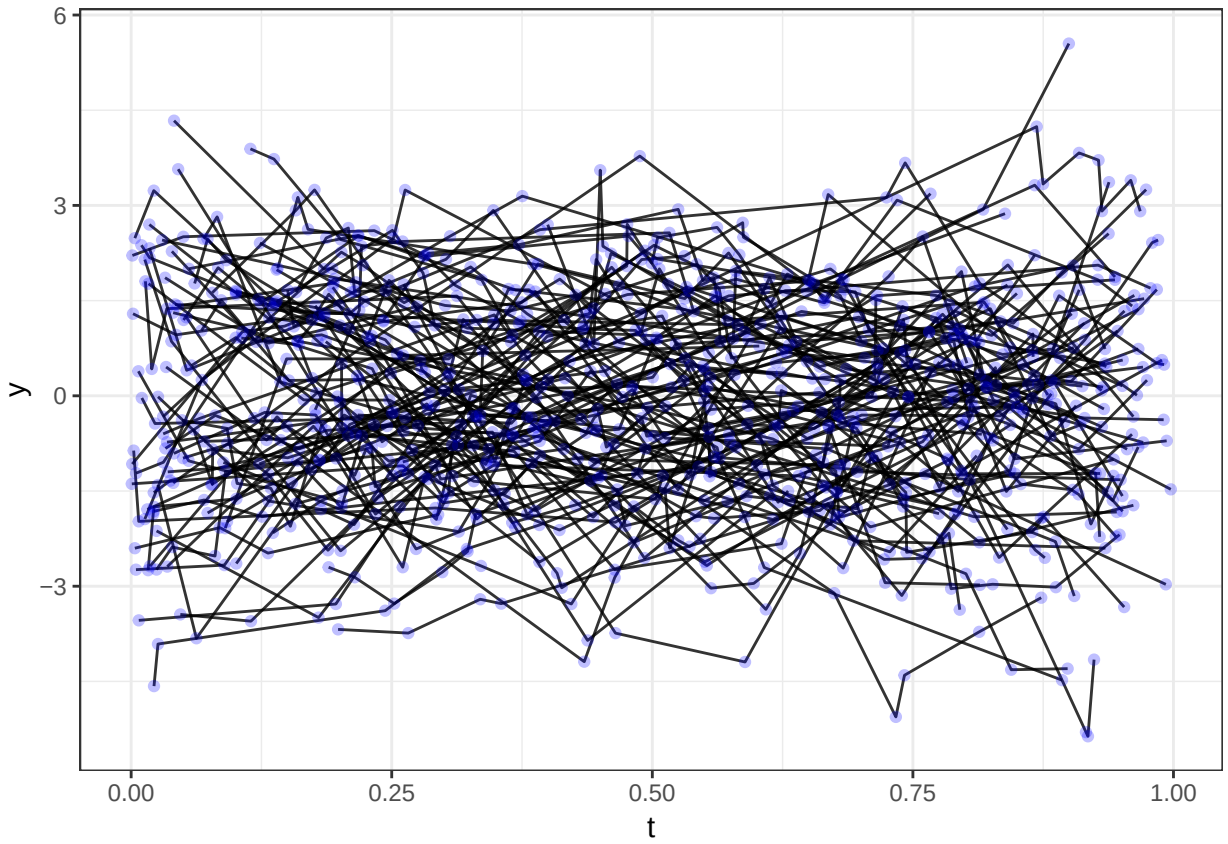

```
plot.spar.fdata(fdata_spar_3_2$Lt, fdata_spar_3_2$Ly)
```




```
plot.spar.fdata(fdata_spar_20_1$Lt, fdata_spar_20_1$Ly)
```




```
plot.spar.fdata(fdata_spar_20_2$Lt, fdata_spar_20_2$Ly)
```




```
################## FPCA for dense functional data ###################
input_den_3_1 <- MakeFPCAInputs(tVec = tt, yVec = fdata_den_3_1$Y)
fit_den_3_1 <- FPCA(Ly = input_den_3_1$Ly, Lt = input_den_3_1$Lt,
                    optns = list(methodSelectK = 'FVE',
                                 methodXi = "IN"))



###### evaluate K selection methods ############

## dense, K = 3, sigma = 0.05
den.model.eval.K(fdata_den_3_1)
```

```
## FVE selecting: K= 3 ;  AIC selecting: K= 3 ;  BIC selecting: K= 3 .
```

```
## dense, K = 3, sigma = 0.5
den.model.eval.K(fdata_den_3_2)
```

```
## FVE selecting: K= 3 ;  AIC selecting: K= 3 ;  BIC selecting: K= 3 .
```

```
## dense, K = 20, sigma = 0.05
den.model.eval.K(fdata_den_20_1)
```

```
## FVE selecting: K= 18 ;  AIC selecting: K= 18 ;  BIC selecting: K= 18 .
```

```
## dense, K = 20, sigma = 0.5
den.model.eval.K(fdata_den_20_2)
```

```
## FVE selecting: K= 18 ;  AIC selecting: K= 18 ;  BIC selecting: K= 18 .
```

```
## sparse, K = 3, sigma = 0.05
spar.model.eval.K(fdata_spar_3_1)
```

```
## FVE selecting: K= 6 ;  AIC selecting: K= 4 ;  BIC selecting: K= 4 .
```

```
## sparse, K = 3, sigma = 0.5
spar.model.eval.K(fdata_spar_3_2)
```

```
## FVE selecting: K= 6 ;  AIC selecting: K= 3 ;  BIC selecting: K= 3 .
```

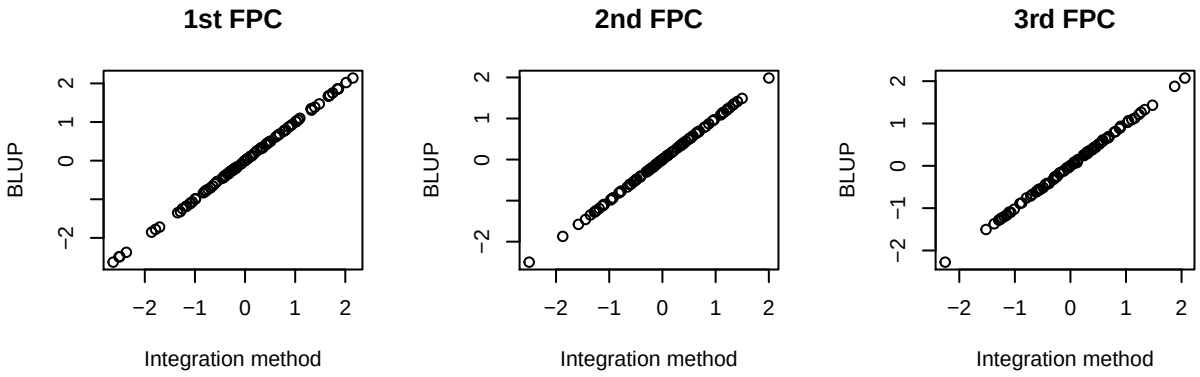
```
## sparse, K = 20, sigma = 0.05
spar.model.eval.K(fdata_spar_20_1)
```

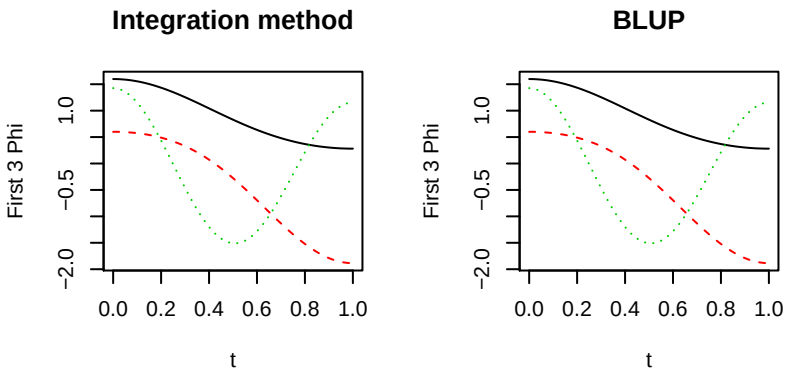
```
## FVE selecting: K= 8 ;  AIC selecting: K= 6 ;  BIC selecting: K= 5 .
```

```
## sparse, K = 20, sigma = 0.5
spar.model.eval.K(fdata_spar_20_2)
```

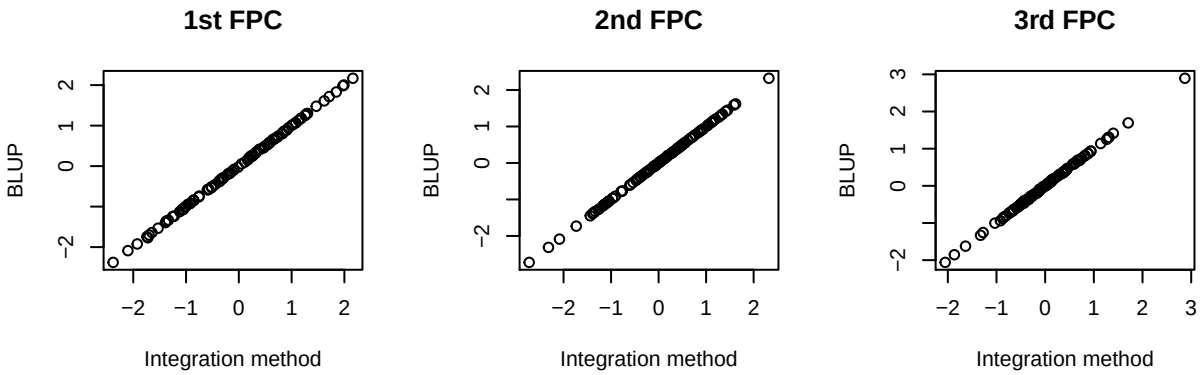
```
## FVE selecting: K= 8 ;  AIC selecting: K= 7 ;  BIC selecting: K= 7 .
```

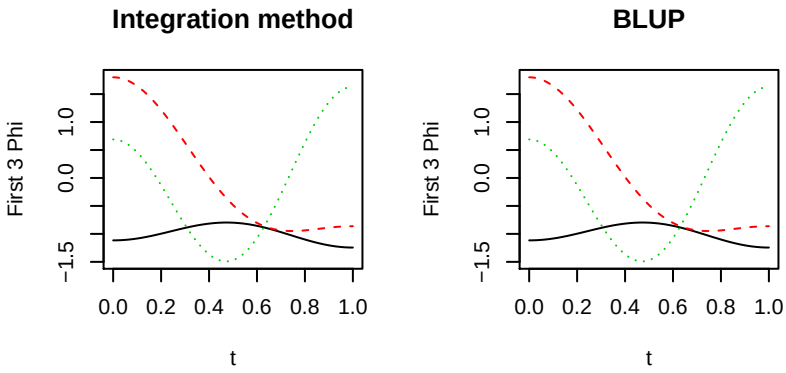
```
###### integration method vs BLUP ############

## dense, K = 3, sigma = 0.05
den.model.eval.xi(fdata_den_3_1)
```



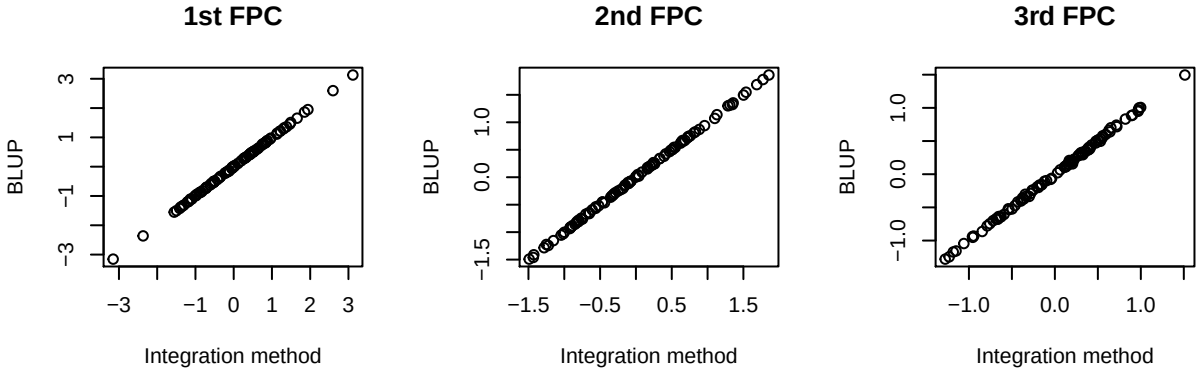


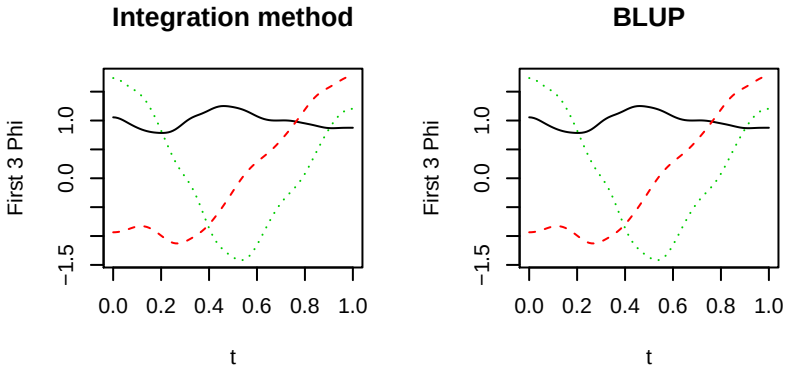

```
## dense, K = 3, sigma = 0.5
den.model.eval.xi(fdata_den_3_2)
```



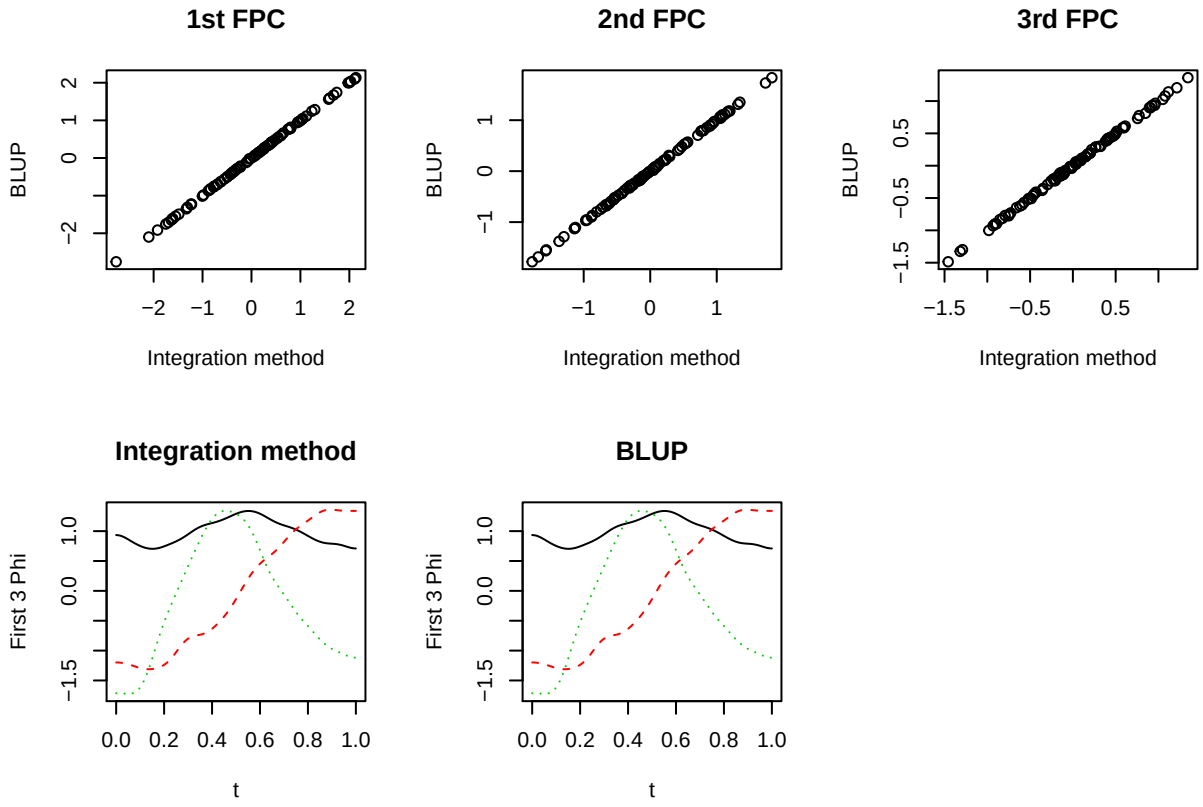



```
## dense, K = 20, sigma = 0.05
den.model.eval.xi(fdata_den_20_1)
```

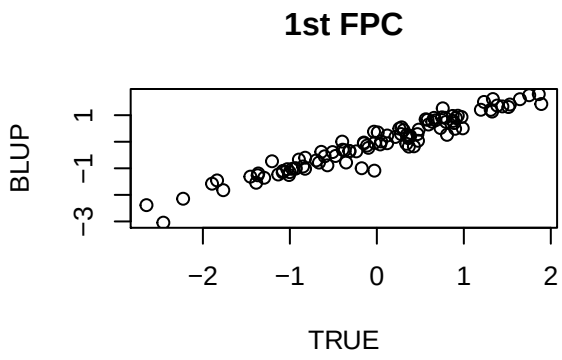


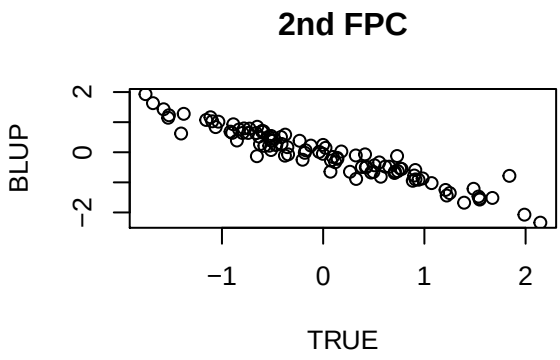


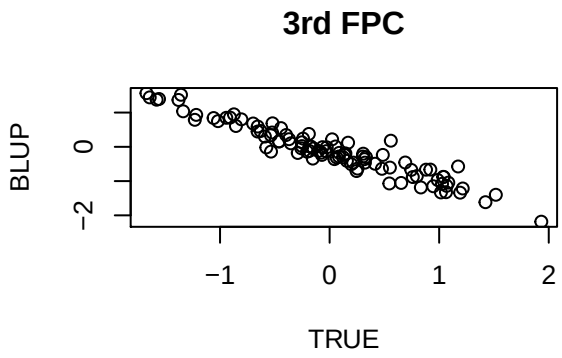
```
## dense, K = 20, sigma = 0.5
den.model.eval.xi(fdata_den_20_2)
```



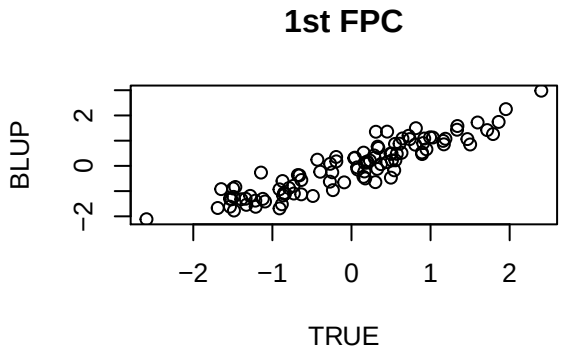

```
## sparse, K = 3, sigma = 0.05
spar.model.eval.xi(fdata_spar_3_1)
```

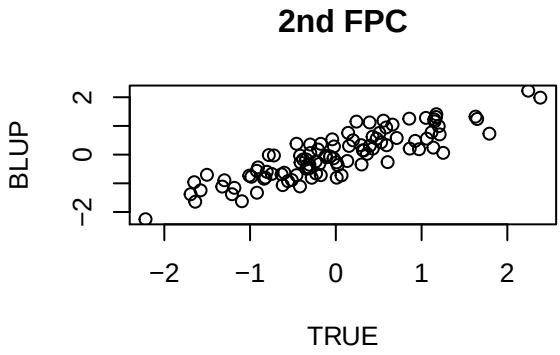


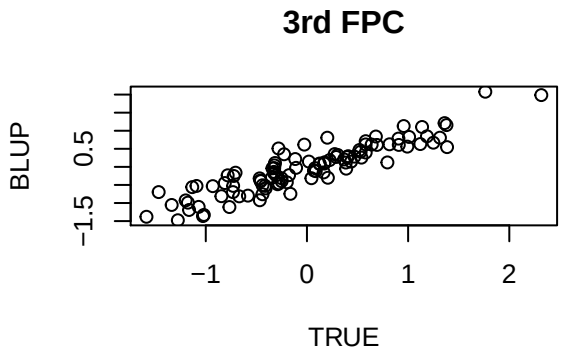




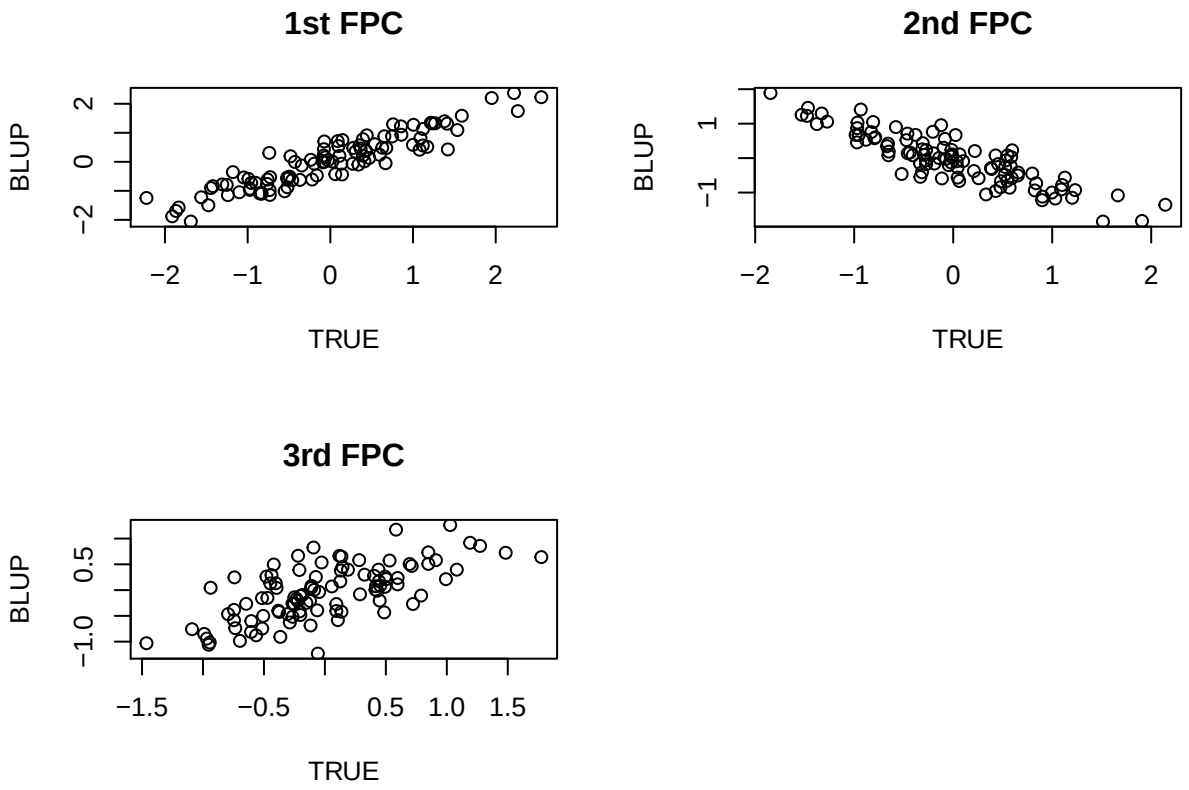

```
## sparse, K = 3, sigma = 0.5
spar.model.eval.xi(fdata_spar_3_2)
```



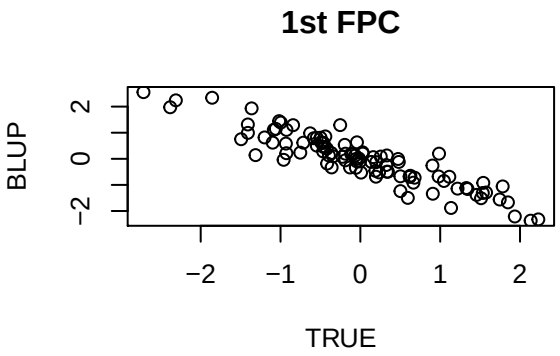


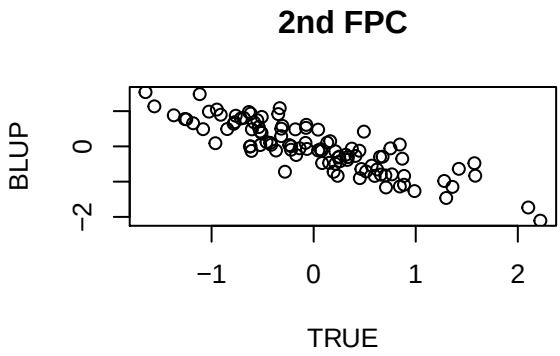


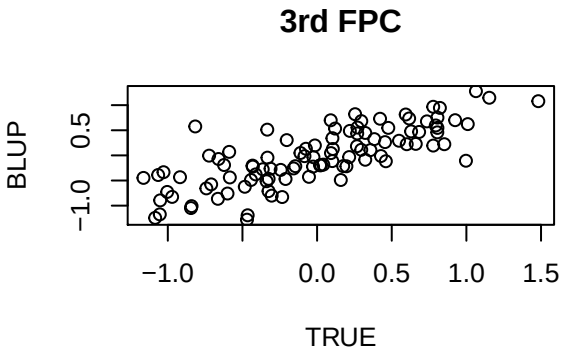
```
## sparse, K = 20, sigma = 0.05
spar.model.eval.xi(fdata_spar_20_1)
```




```
## sparse, K = 20, sigma = 0.5
spar.model.eval.xi(fdata_spar_20_2)
```







Problem 2

mFPCA funcitons

```
# Note: These functions are used to estimate bivariate FPCA (sparse).
# The methodology applied here are based on paper: Chiou, Chen and Yang (2014).
# I also import some functions in R package: fdapace, locfit.
# The parameters (bindwidths, K) can only be manually adjusted.

library(fdapace)
library(locfit)
```

```
## locfit 1.5-9.1    2013-03-22
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## create mFPCA object that can be directed used in mFPCA function.
Make_mFPCA_Inputs <- function(idVec, tVec, yMat){
```

```
  label <- unique(idVec)
  n_sub <- length(label)
  count <- rep(0, n_sub)
  for (i in 1:n_sub){
    count[i] <- sum(idVec == label[i])
  }
  weight <- rep(1/count, times = count)
  out <- data.frame(id = idVec, t = tVec, weight, yMat)
  return(out)
}


## estimate mean funcitons (D: output of Make_mFPCA_Inputs).
Mean_Curve_Est <- function(D, deg = 1, h_Mu = 0.1, ngrid = 100){
  N <- nrow(D)
  p <- ncol(D) - 3
  t <- D$t
  Y <- D[,-(1:3)]
  muhat_grid <- matrix(0, ngrid, p)
  muhat <- matrix(0, N, p)
  for (i in 1:p){
    fit <- locfit(Y[,i] ~ lp(t, deg = deg, h = h_Mu), weights = D$weight,
                  ev = lfgrid(mg = ngrid) )
    muhat_grid[,i] <- predict(fit)
    muhat[,i] <- predict(fit, t)
  }
  return(list(muhat = muhat, muhat_grid = muhat_grid))
}


## create raw covariance data (D: output of Make_mFPCA_Inputs).
Raw_Cov_data <- function(D, muhat){
  D[,-(1:3)] <-  D[,-(1:3)] - muhat
  p <- ncol(muhat)
  label <- unique(D$id)
  n_sub <- length(label)
  rcov <- NULL
  for (i in 1:n_sub){
    d <- D[ D$id==label[i], ]
    TT <- expand.grid(t1=d$t, t2=d$t)
    w <- d$weight[1]
    if ( w ==1 ){
      weight <- 1/2
    } else {
      weight <- rep( (w^2)/(1-w), nrow(TT) )
    }
    id <- rep(label[i], nrow(TT))
    Raw <- apply(d[,-(1:3)], 2, function(x){
      c(tcrossprod(x))
    })
    if (nrow(TT) == 1){
      dd <- c(id, as.numeric(TT), weight, as.numeric(Raw))
    } else {
```

```
      dd <- cbind(id, TT, weight, Raw)
    }
    rcov <- rbind(rcov, dd)
  }
  rcov <- as.data.frame(rcov)
  return(rcov)
}


## estimate covariance surface (rcov: output of Raw_Cov_data).
Cov_Surf_Est <- function(rcov, t, h_Cov = 0.2, ngrid = 100){
  p <- ncol(rcov) - 4
  rcov <- rcov %>% filter(t1 != t2)
  t1 <- rcov$t1
  t2 <- rcov$t2
  weight <- rcov$weight
  covArray <- array(0, c(ngrid, ngrid, p))
  covArray_raw <- array(0, c(ngrid, ngrid, p))
  covDiag <- matrix(0, length(t), p)
  covDiag_grid <- matrix(0, ngrid, p)
  for (i in 1:p){
    yi <- rcov[,4+i]
    fit <- locfit(yi ~ lp(t1, t2, deg=1, h=h_Cov),
                  ev=lfgrid(mg=ngrid), weights = weight)
    covMat <- matrix(predict(fit), ngrid, ngrid)
    eig_cov <- eigen(covMat)
    eig_value <- eig_cov$values
    eig_vector <- eig_cov$vectors
    eff_ind <- which(eig_value > 0)
    covMat <- eig_vector[,eff_ind] %*%  (t(eig_vector[,eff_ind]) *  eig_value[eff_ind])
    covDiag[,i] <- pmax( predict(fit, data.frame(t1=t, t2=t)), 1e-4)
    covDiag_grid[,i] <- diag(covMat)
    V <- diag(diag(covMat)^(-1/2))
    covArray[,,i] <- V %*% covMat %*% V
    covArray_raw[,,i] <- covMat
  }
  return(list(covArray = covArray, covDiag = covDiag, covDiag_grid = covDiag_grid,
              covArray_raw = covArray_raw))
}


## data transformation (D: output of Make_mFPCA_Inputs).
Trans_data <- function(D, muhat, covDiag){
  D[,-(1:3)] <-  (D[,-(1:3)] - muhat) / ((covDiag)^(1/2))
  return(D)
}


## create raw cross-covariance data (D: output of Trans_data).
Raw_Cross_Cov_data <- function(D){
  p <- ncol(D) - 3
  label <- unique(D$id)
  n_sub <- length(label)
```

```
  rcrosscov <- NULL
  for (i in 1:n_sub){
    d <- D[ D$id==label[i], ]
    TT <- expand.grid(t1=d$t, t2=d$t)
    w <- d$weight[1]
    if ( w ==1 ){
      weight <- 1/2
    } else {
      weight <- rep( (w^2)/(1-w), nrow(TT) )
    }
    id <- rep(label[i], nrow(TT))
    Raw <- matrix(0, nrow(TT), p*(p-1)/2 )
    flag <- 0
    for (k in 2:p){
      for (l in 1:(k-1)){
        flag <- flag + 1
        Raw[,flag] <- c( d[,3+k] %*% t(d[,3+l]) )
      }
    }
    if (nrow(TT) == 1){
      dd <- c(id, as.numeric(TT), weight, as.numeric(Raw))
    } else {
      dd <- cbind(id, TT, weight, Raw)
    }
    rcrosscov <- rbind(rcrosscov, dd)
  }
  rcrosscov <- as.data.frame(rcrosscov)
  return(rcrosscov)
}


## estimate cross-covariance surface (rcrosscov: output of Raw_Cross_Cov_data).
Cross_Cov_Surf_Est <- function(rcrosscov, h_Cov = 0.2, ngrid = 100){
  q <- ncol(rcrosscov) - 4
  rcrosscov <- rcrosscov %>% filter(t1 != t2)
  t1 <- rcrosscov$t1
  t2 <- rcrosscov$t2
  weight <- rcrosscov$weight
  crosscovArray <- array( 0, c(ngrid, ngrid, q) )
  flag <- 0
  for (i in 1:q){
      flag <- flag + 1
      y_kl <- rcrosscov[,4+i]
      fit <- locfit(y_kl ~ lp(t1, t2, deg=1, h=h_Cov),
                    ev=lfgrid(mg=ngrid), weights = weight)
      crosscovMat <- matrix(predict(fit), ngrid, ngrid)
      crosscovArray[,,i] <- crosscovMat
    }

  return(crosscovArray)
}
```

```
## estimate sigma_k^2 (D: output of Make_mFPCA_Inputs).
Sigma_Est <- function(D, muhat, covArray_raw, h_W = 0.2){
  D[,-(1:3)] <-  D[,-(1:3)] - muhat
  p <- ncol(D) - 3
  tt <- D$t
  weight <- D$weight
  ngrid <- dim(covArray_raw)[1]
  eff_index <- seq( round(ngrid/4), round(3*ngrid/4), 1 )
  sigma2 <- rep(0, p)
  for (i in 1:p){
    yi <- (D[,3+i])^2
    fit <- locfit(yi ~ lp(tt, deg=1, h=h_W),
                  ev=lfgrid(mg=ngrid), weights = weight)
    errVec <- predict(fit)
    covVec <- diag(covArray_raw[,,i])
    sigmaVec <- errVec - covVec
    sigma2[i] <- mean( sigmaVec[eff_index] )
  }
  return(sigma2)
}


## compute first L eigenvalues and eigenfunctions by discretizing the
## covariance and cross-covariance functions
Get_Eigen <- function(L = 3, covArray, crosscovArray){
  p <- dim(covArray)[3]
  ngrid <- dim(covArray)[1]
  Cov_joint <- matrix(0, p*ngrid, p*ngrid)
  Cov_joint[1:ngrid, 1:ngrid] <- covArray[,,1]
  flag <- 0
  for (k in 2:p){
    Cov_joint[ngrid*(k-1)+(1:ngrid), ngrid*(k-1)+(1:ngrid)] <- covArray[,,k]
    for (l in 1:(k-1)){
      flag <- flag + 1
      Cov_joint[ngrid*(k-1)+(1:ngrid), ngrid*(l-1)+(1:ngrid)] <- crosscovArray[,,flag]
      Cov_joint[ngrid*(l-1)+(1:ngrid), ngrid*(k-1)+(1:ngrid)] <- t(crosscovArray[,,flag])
    }
  }
  eigenCov <- eigen(Cov_joint)
  L1 <- which(eigenCov$values<0)[1]
  L <- min(L, L1-1)
  eigenval <- eigenCov$values[1:L] / ngrid
  eigenfunc <- eigenCov$vectors[,1:L, drop = FALSE] * sqrt(ngrid)
  Cov_joint_pd <- eigenfunc %*% diag(eigenval, ncol = length(eigenval)) %*% t(eigenfunc)
  return(list(eigenval = eigenval, eigenfunc = eigenfunc,
              Cov_joint = Cov_joint,  Cov_joint_pd = Cov_joint_pd))
}


## BLUP (D: output of Trans_data, eigen_out: output of Get_Eigen).
mBLUP <- function(D, eigen_out, sigma2, covDiag, tgrid){
  p <- ncol(D) - 3
  L <- length(eigen_out$eigenval)
```

```
  label <- unique(D$id)
  n_sub <- length(label)
  Xi <- matrix(0, n_sub, L)
  rownames(Xi) <- label
  ngrid <- length(tgrid)
  for (i in 1:n_sub){
    index_i <- which(D$id==label[i])
    d <- D[ index_i, ]
    covDiag_i <- covDiag[ index_i, , drop = FALSE]
    Ui <- c( as.matrix(d[,-(1:3)]) )
    ti <- d$t
    mi <- length(ti)
    Hi_trans <- matrix(0, p*mi, L)
    Sig_Ui <- matrix(0, p*mi, p*mi)

    Hi_trans[1:mi, ] <- matrix(ConvertSupport(tgrid, ti,
      phi = eigen_out$eigenfunc[1:ngrid, ]), ncol = L)
    Sig_Ui[1:mi, 1:mi] <- ConvertSupport(tgrid, ti,
      Cov = eigen_out$Cov_joint_pd[1:ngrid, 1:ngrid]) +
      diag(sigma2[1]/covDiag_i[,1], nrow = mi)

    for (k in 2:p){
      Hi_trans[(k-1)*mi+(1:mi),] <- matrix(ConvertSupport(tgrid, ti,
        phi = eigen_out$eigenfunc[(k-1)*ngrid+(1:ngrid),]), ncol = L)
      Sig_Ui[(k-1)*mi+(1:mi), (k-1)*mi+(1:mi)] <- ConvertSupport(tgrid, ti,
        Cov = eigen_out$Cov_joint_pd[(k-1)*ngrid+(1:ngrid), (k-1)*ngrid+(1:ngrid)]) +
        diag( sigma2[k] / covDiag_i[, k], nrow = mi)
      for (l in 1:(k-1)){
        Sig_Ui[(k-1)*mi+(1:mi), (l-1)*mi+(1:mi)] <- ConvertSupport(tgrid, ti,
          Cov = eigen_out$Cov_joint_pd[(k-1)*ngrid+(1:ngrid), (l-1)*ngrid+(1:ngrid)], isCrossCov = TRUE)
        Sig_Ui[(l-1)*mi+(1:mi), (k-1)*mi+(1:mi)] <- t(Sig_Ui[(k-1)*mi+(1:mi), (l-1)*mi+(1:mi)])
      }
    }
    Hi <- t(Hi_trans) * eigen_out$eigenval
    Xi[i,] <- as.numeric( Hi %*% solve(Sig_Ui) %*% Ui )
  }
  return(Xi)
}


## estimate X_L
X_L_est <- function(muhat_grid, eigen_out, XiEst, covDiag_grid){
  ngrid <- nrow(muhat_grid)
  p <- ncol(muhat_grid)
  n_sub <- nrow(XiEst)
  Xhat <- array(0, c(n_sub, ngrid, p))
  for (i in 1:n_sub){
    Xhat[i,,] <- muhat_grid
  }
  Phi <- eigen_out$eigenfunc
  for (k in 1:p){
    Phi_k <- Phi[(k-1)*ngrid+(1:ngrid),]
    D_Phi_k <- Phi_k * sqrt(covDiag_grid[,k])
```

```
    Xhat[,,k] <- Xhat[,,k] + matrix( XiEst %*% t(D_Phi_k), ncol = ngrid)
  }
  return(Xhat)
}


## mFPCA
 mFPCA <- function(idVec, tVec, yMat, ngrid = 100, h_Mu = 0.1, h_Cov = 0.2,
                   h_W = 0.2, L = 3){
   DD1 <- Make_mFPCA_Inputs(idVec, tVec, yMat)
   tgrid <- seq(min(DD1$t), max(DD1$t), length.out = ngrid)

   ## estimate mean curves
   muEst <- Mean_Curve_Est(DD1, h_Mu = h_Mu)
   muhat <- muEst$muhat
   muhat_grid <- muEst$muhat_grid

   ## estimate covariance surfaces
   rcov <- Raw_Cov_data(DD1, muhat)
   covEst <- Cov_Surf_Est(rcov, DD1$t, h_Cov = h_Cov)
   covArray <- covEst$covArray
   covDiag <- covEst$covDiag
   covDiag_grid <- covEst$covDiag_grid
   covArray_raw <- covEst$covArray_raw

   ## transform data & estimate cross-covariance surfaces
   DD2 <- Trans_data(DD1, muhat, covDiag)
   rcrosscov <- Raw_Cross_Cov_data(DD2)
   crosscovArray <- Cross_Cov_Surf_Est(rcrosscov, h_Cov = h_Cov)
   sigma2 <- Sigma_Est(DD1, muhat, covArray_raw, h_W = h_W)

   ## eigenvalue & eigenfunction
   eigen_out <- Get_Eigen(L = L, covArray, crosscovArray)
   Cov_joint <- eigen_out$Cov_joint
   eigenfunc <- eigen_out$eigenfunc

   ## BLUP for estimating xi
   XiEst <- mBLUP(DD2, eigen_out, sigma2, covDiag, tgrid)

   ## Estimate X_L
   Xhat <- X_L_est(muhat_grid, eigen_out, XiEst, covDiag_grid)

   output <- list(Mu_hat = muhat_grid, D_hat = covDiag_grid, C_hat = eigen_out$Cov_joint_pd,
                  Phi_hat = eigenfunc, Xi_hat = XiEst, sigma2_hat = sigma2, X_hat = Xhat, tgrid = tgrid

   return(output)
 }
```

real data analysis

```
library(dplyr)
library(ggplot2)
```

```r
library(locfit)
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
library(fda)
```

```
## Loading required package: splines
```

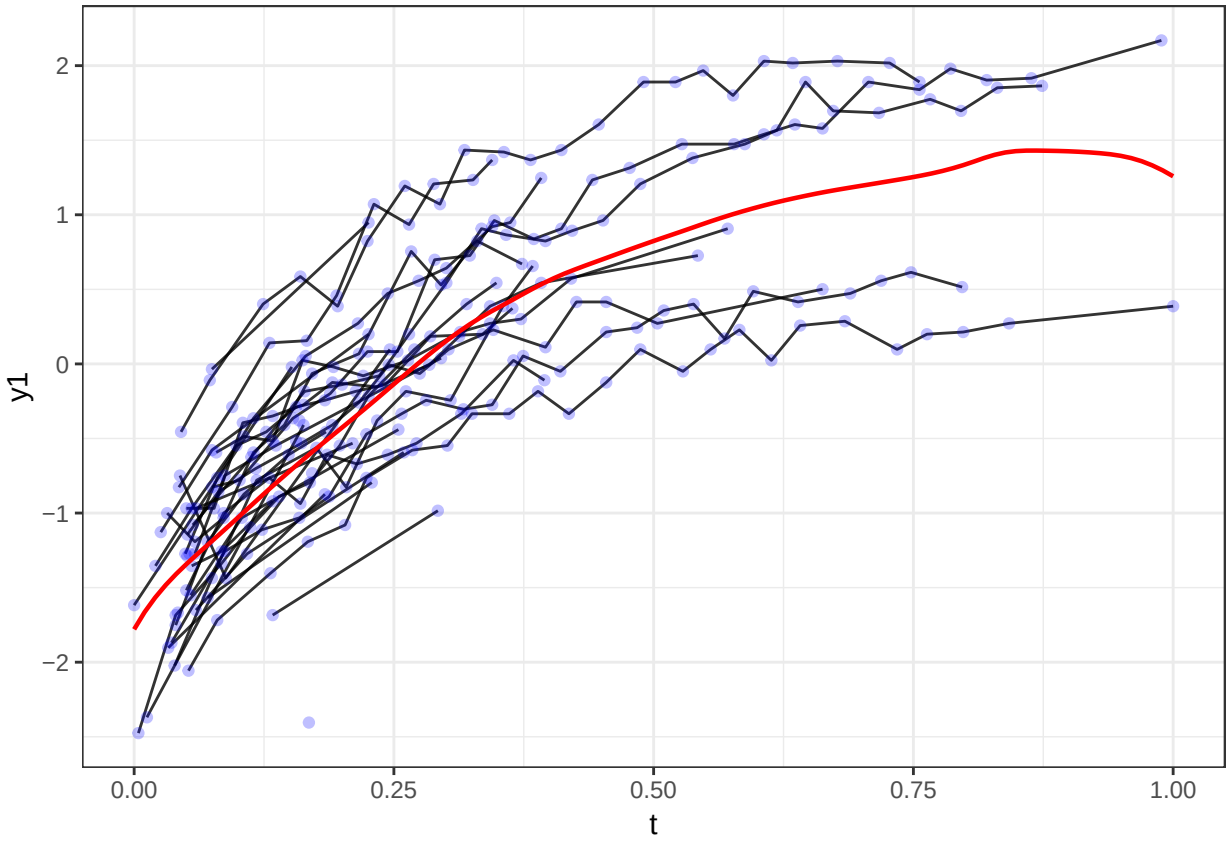
```
## Loading required package: Matrix
```

```
##
## Attaching package: 'fda'
```

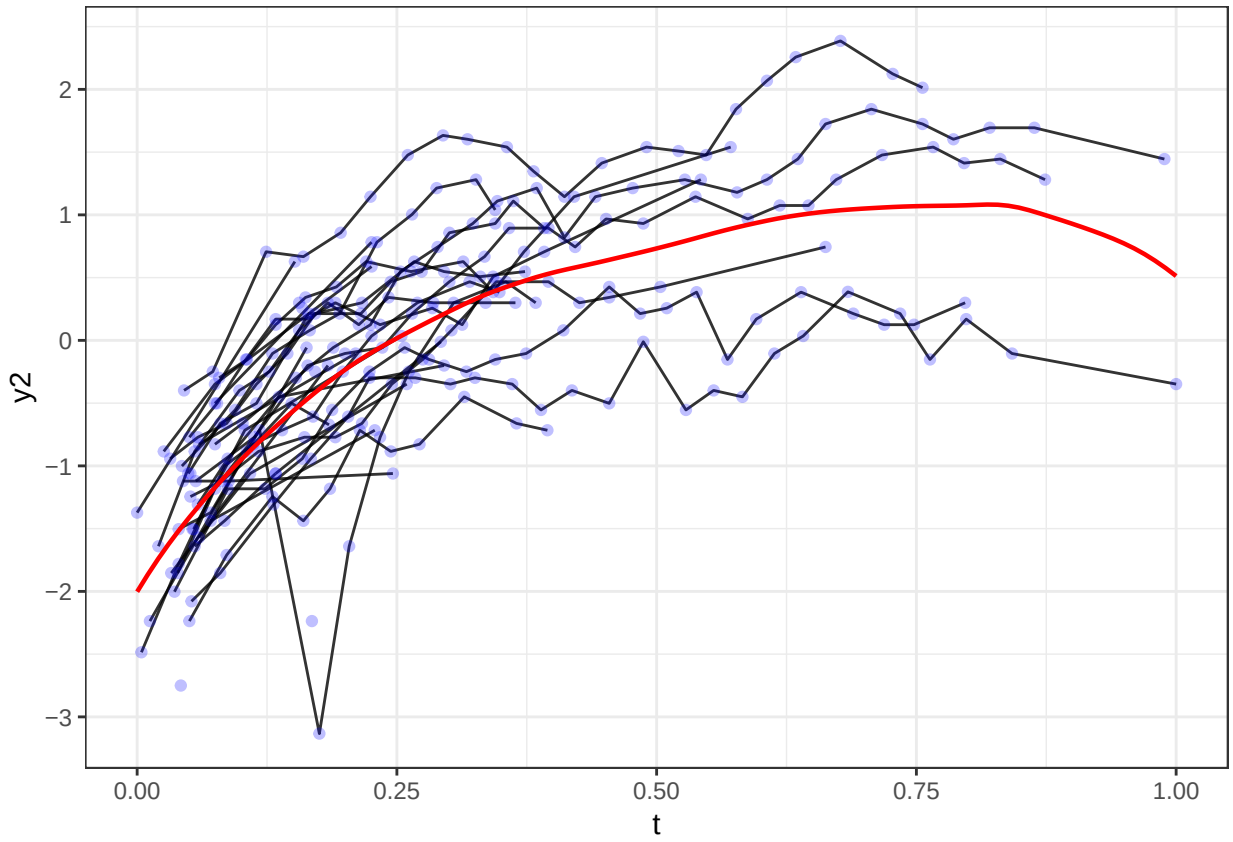
```
## The following object is masked from 'package:graphics':
##
##     matplot
```

```r
library(reshape2)
source("/Users/apple/Desktop/ISU 2019 fall/STAT547/hw/hw5/mFPCA.R")

#### read data ####
DD0 <- read.delim('http://www.statsci.org/data/oz/wallaby.txt') %>%
  select(Anim, Leng, Weight, Age) %>%
  na.omit %>%
  # filter(Age >= 0.6 * 365.24 & Age <= 2.4 * 365.24 ) %>%
  mutate(Leng = log(Leng), Weight = log(Weight)) %>%
  mutate(Leng = (Leng - mean(Leng))/sd(Leng),
         Weight = (Weight - mean(Weight))/sd(Weight),
         Age = (Age - min(Age))/diff(range(Age)))

names(DD0) <- c("label", "y1", "y2", "t")


#### run mFPCA ####
idVec <- DD0$label
tVec <- DD0$t
yMat <- DD0[,c(2,3)]

mFPCA_out <- mFPCA(idVec, tVec, yMat, ngrid = 100,
                   h_Mu = 0.2, h_Cov = 0.5, h_W = 0.5, L = 3)


#### plot mean ####
tgrid <- mFPCA_out$tgrid
```

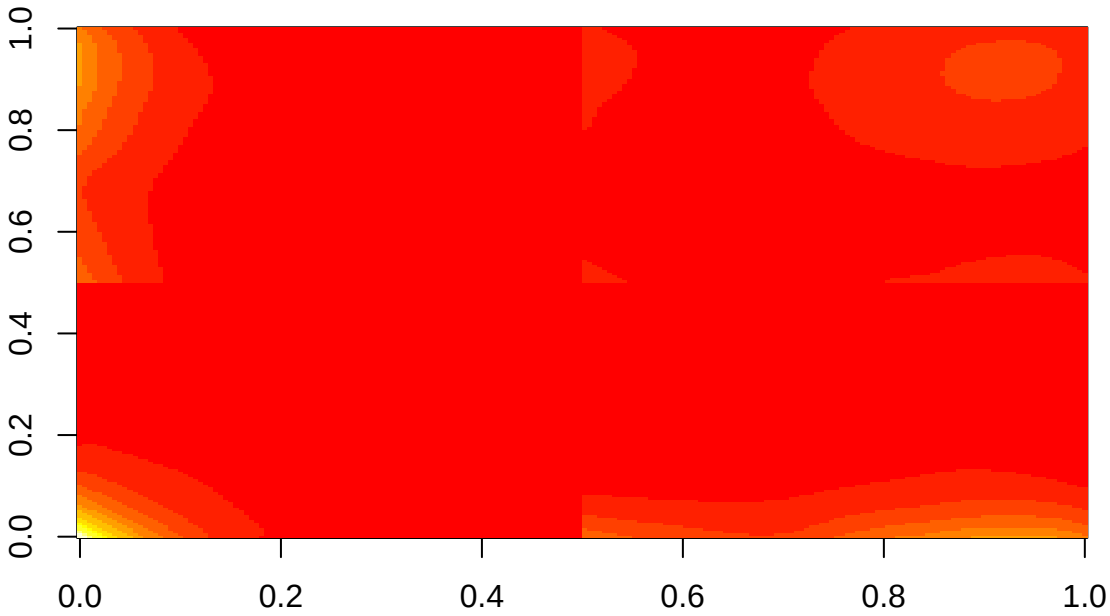
```r
muhat_grid <- mFPCA_out$Mu_hat
DD_mu <- data.frame(tgrid, muhat_grid)
names(DD_mu) <- c("tgrid", "mu1", "mu2")

# y1
ggplot(data = DD0) +
  geom_line( aes(x = t, y = y1, group = label), size = 0.5, alpha = 0.8) +
  geom_point( aes(x = t, y = y1, group = label), colour = "blue", alpha = 0.25) +
  geom_line(data = DD_mu, aes(x = tgrid, y = mu1), colour = "red", size = 0.8) +
  theme_bw()
```



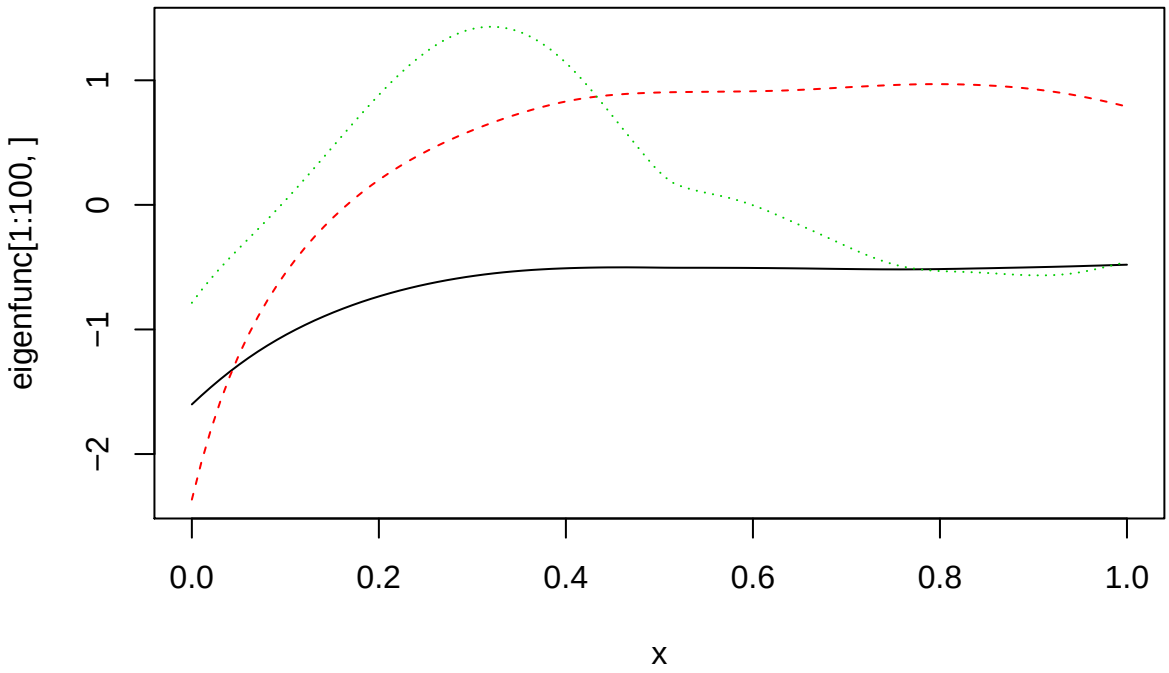

```r
# y2
ggplot(data = DD0) +
  geom_line( aes(x = t, y = y2, group = label), size = 0.5, alpha = 0.8) +
  geom_point( aes(x = t, y = y2, group = label), colour = "blue", alpha = 0.25) +
  geom_line(data = DD_mu, aes(x = tgrid, y = mu2), colour = "red", size = 0.8) +
  theme_bw()
```



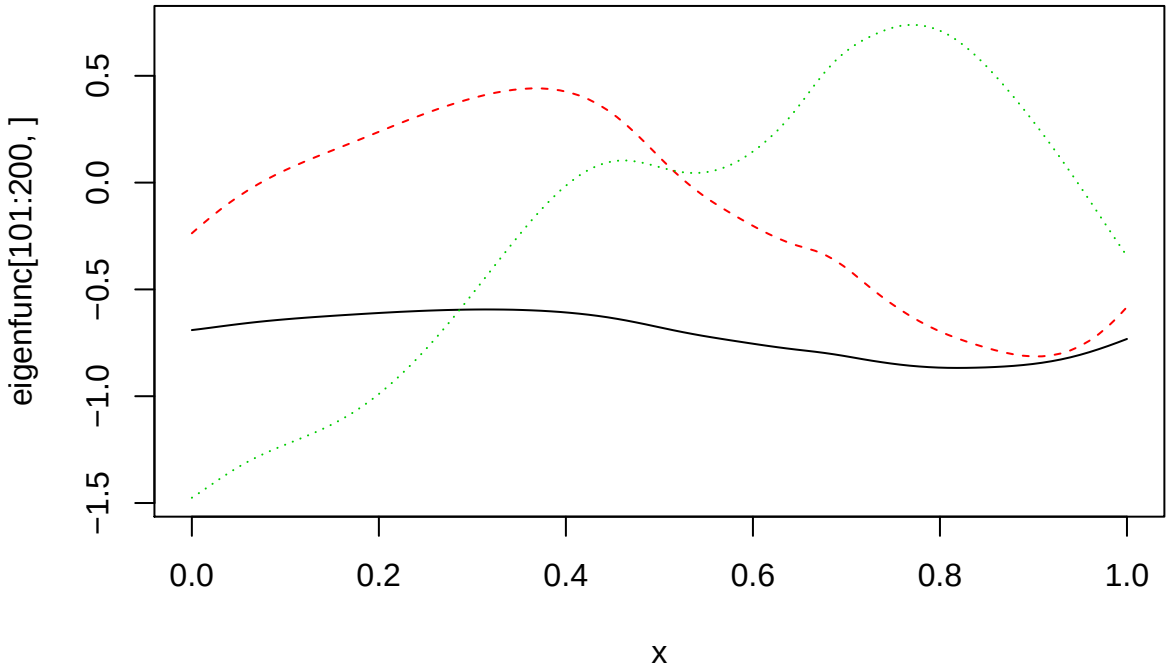

```
#### plot covariance surfaces ####
Cov_hat <- mFPCA_out$C_hat

image(Cov_hat)
```



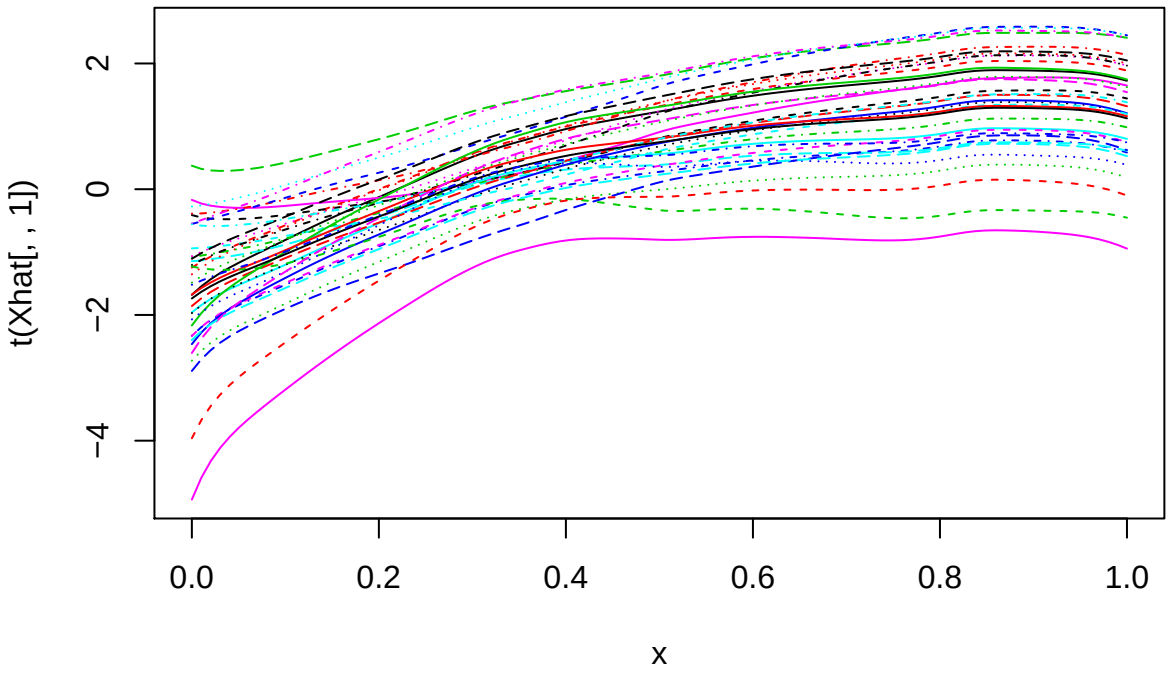

```
#### plot eigenfuncitons ####
eigenfunc <- mFPCA_out$Phi_hat
matplot(tgrid, eigenfunc[1:100, ], type = "l")
```



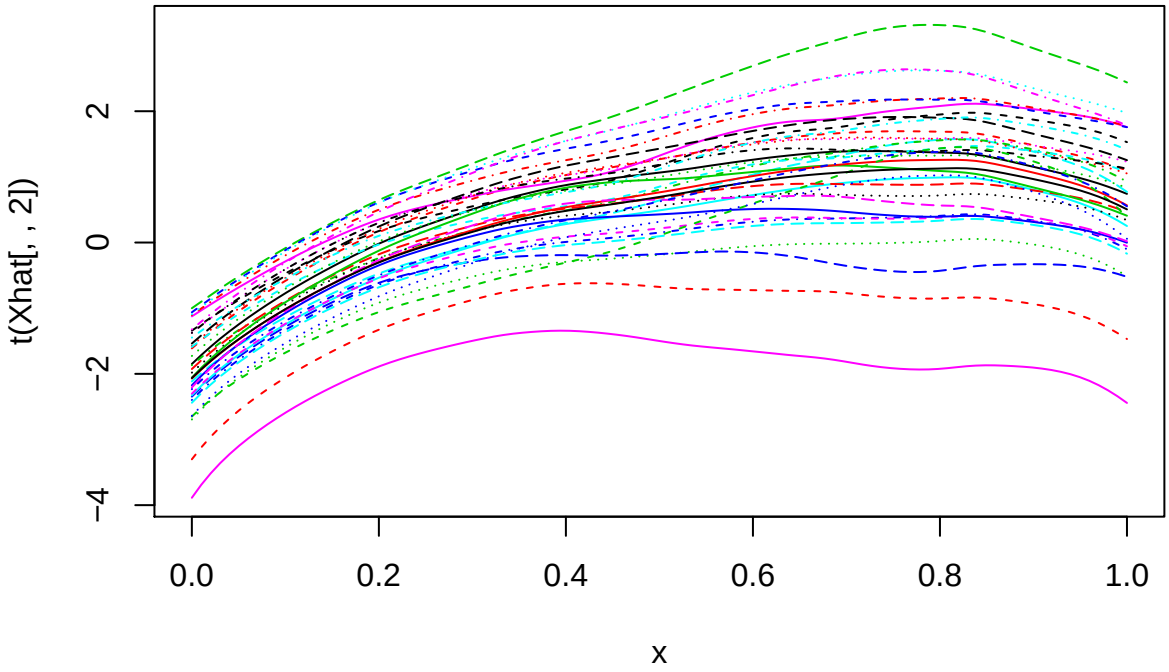

```
matplot(tgrid, eigenfunc[101:200, ], type = "l")
```




```
#### plot Xhat ####
XiEst <- mFPCA_out$Xi_hat
Xhat <- mFPCA_out$X_hat

matplot(tgrid, t(Xhat[,,1]), type = "l")
```

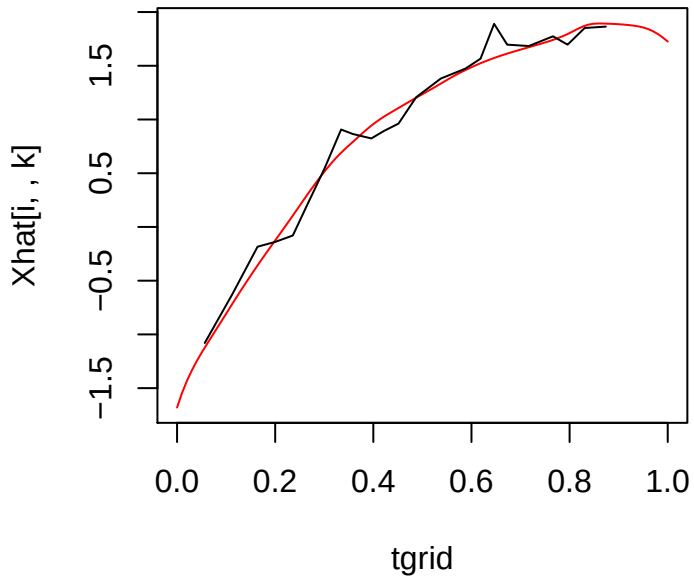



```
matplot(tgrid, t(Xhat[,,2]), type = "l")
```

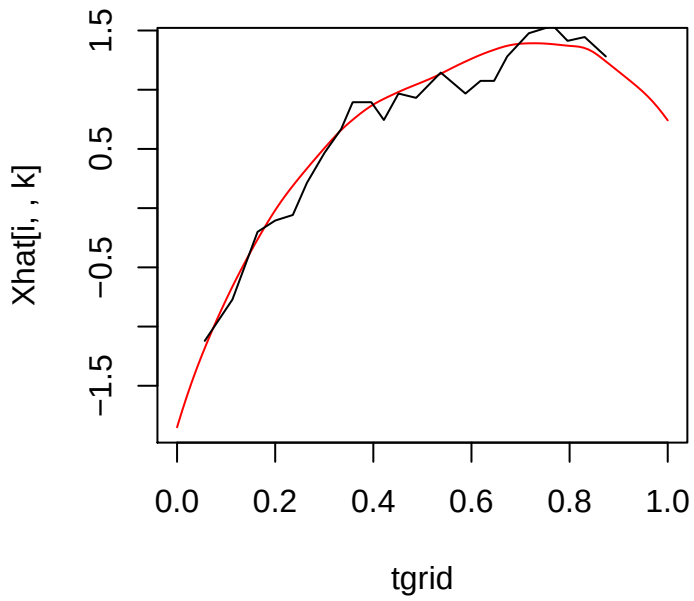



```
## estimate vs true plot
label <- unique(DD0$label)
for (i in 1:length(label)){
  for (k in 1:2){
    index_i <- which( DD0$label == label[i] )
    plot(tgrid, Xhat[i,,k], type = "l", col = "red")
    if (length(index_i) == 1){
       points(DD0[index_i,4], DD0[index_i,1+k])
    } else {
```

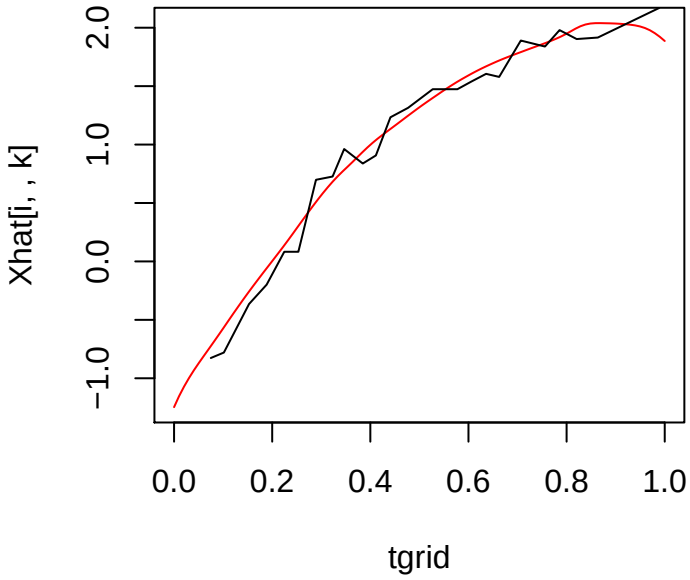
```
      lines(DD0[index_i,4], DD0[index_i,1+k])
    }
    cat("i = ", i, "; k = ", k, "\n")
  }
}
```



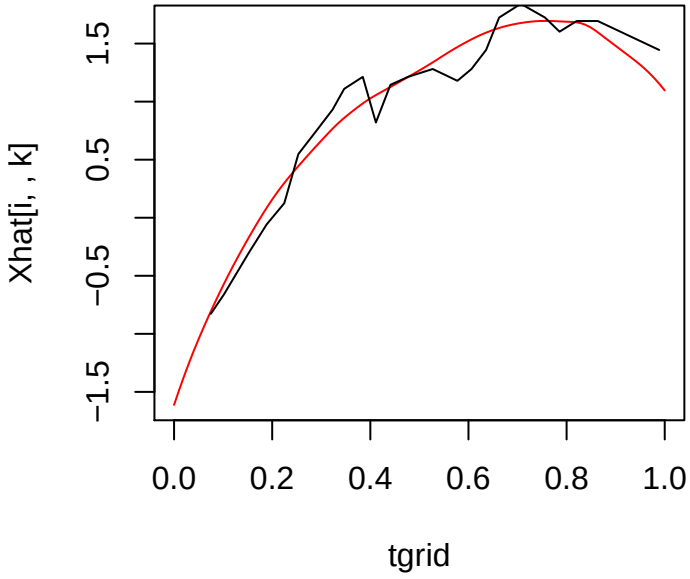

```
## i =  1 ; k =  1
```



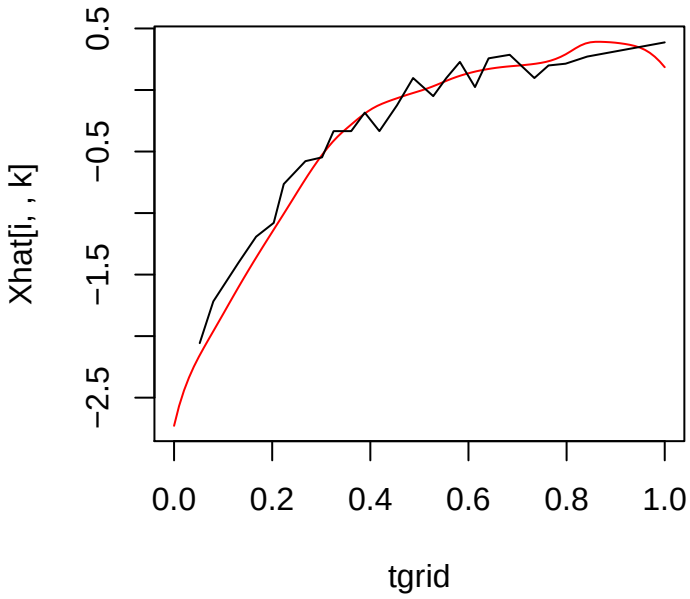

```
## i =  1 ; k =  2
```


Xhat[i, , k]
2.0
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

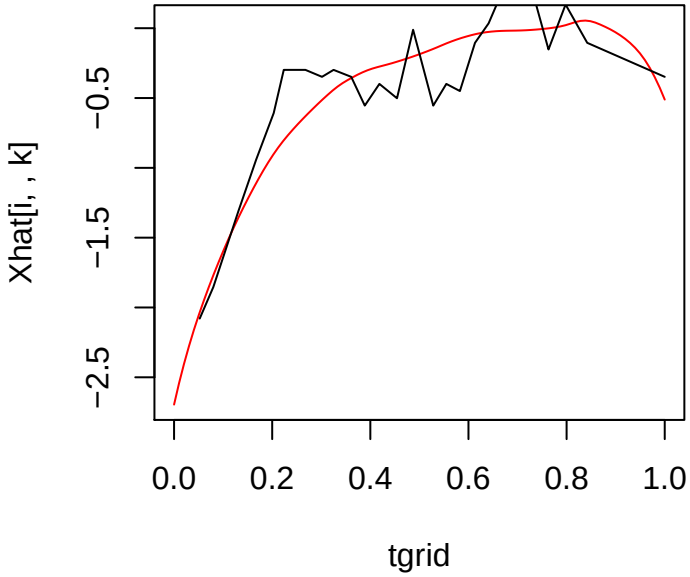
```
## i =  2 ; k =  1
```


Xhat[i, , k]
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

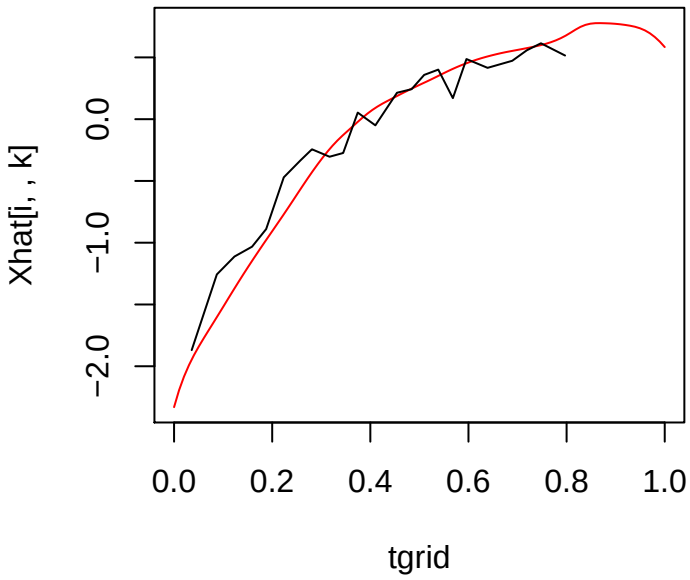
```
## i =  2 ; k =  2
```


Xhat[i, , k]
0.5
−0.5
−1.5
−2.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

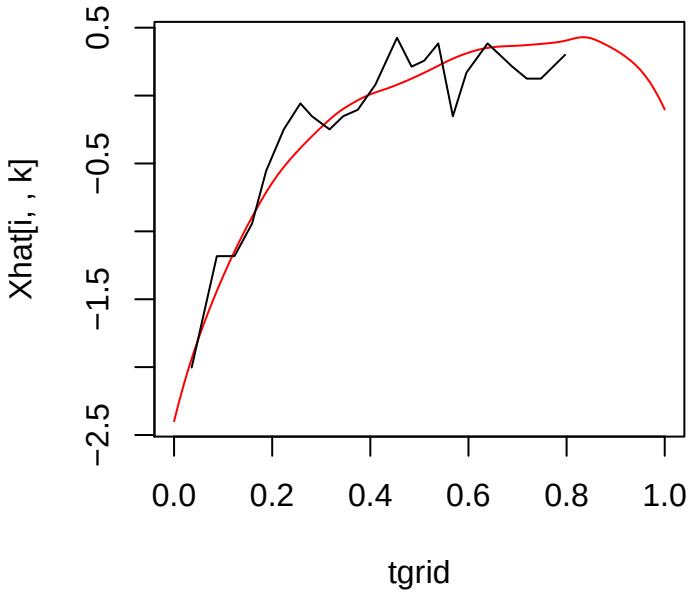
```
## i =  3 ; k =  1
```


Xhat[i, , k]
−0.5
−1.5
−2.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

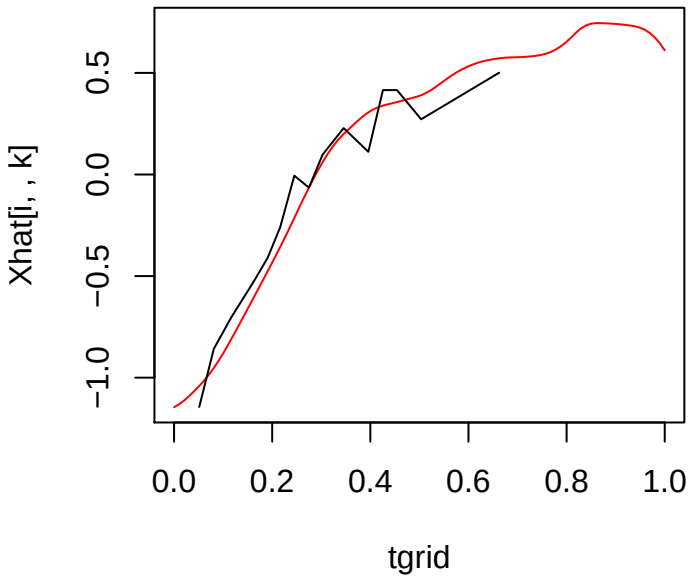
```
## i =  3 ; k =  2
```


Xhat[i, , k]
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

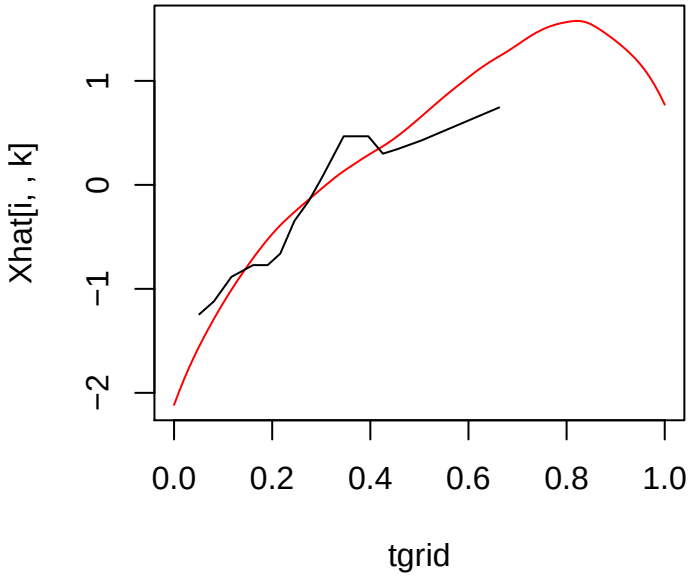
```
## i =  4 ; k =  1
```


Xhat[i, , k]
0.5
−0.5
−1.5
−2.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

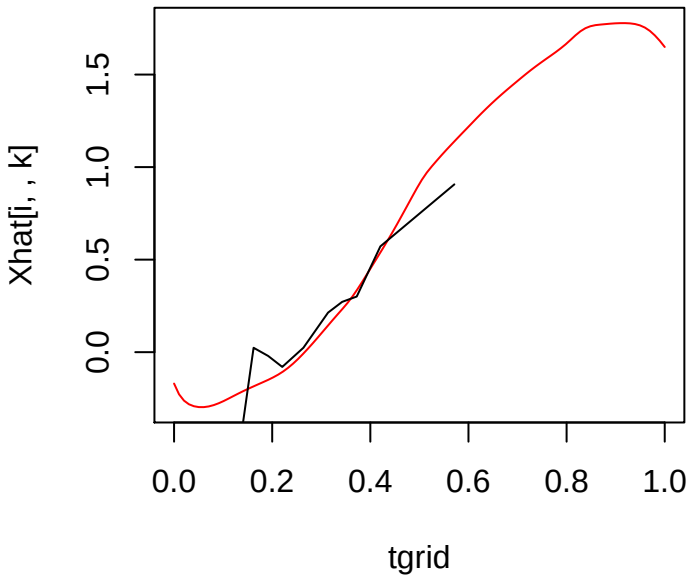
```
## i =  4 ; k =  2
```


Xhat[i, , k]
−1.0
−0.5
0.0
0.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

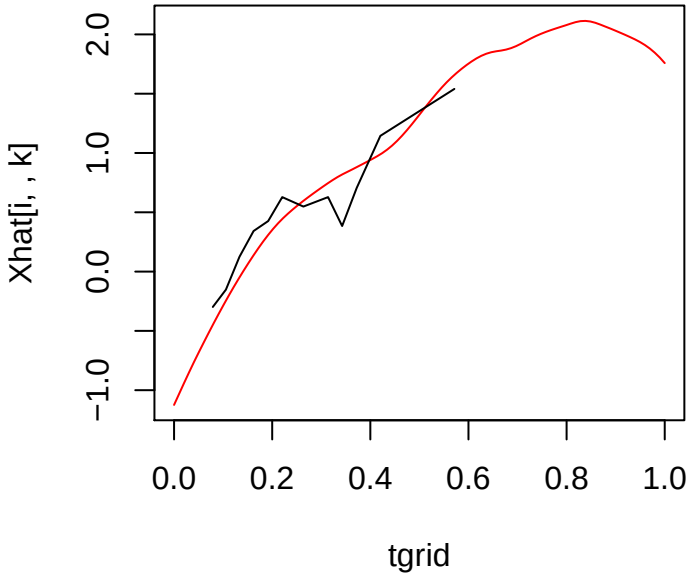
```
## i =  5 ; k =  1
```


Xhat[i, , k]
−2
−1
0
1
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

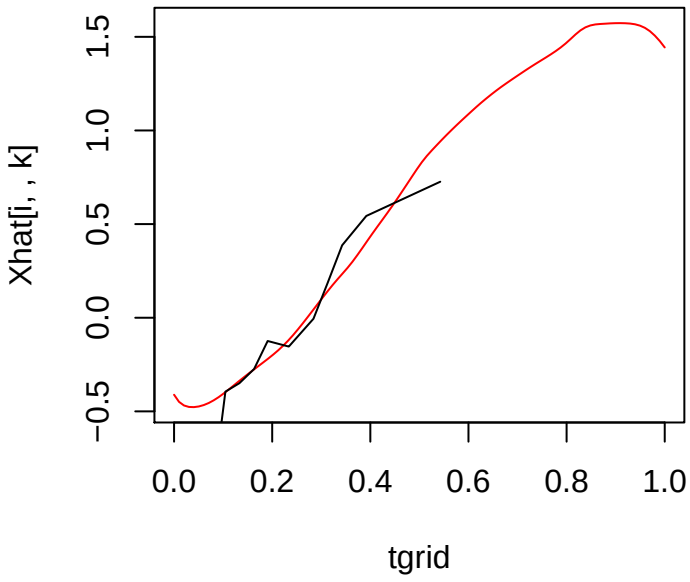
```
## i =  5 ; k =  2
```


Xhat[i, , k]
0.0
0.5
1.0
1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

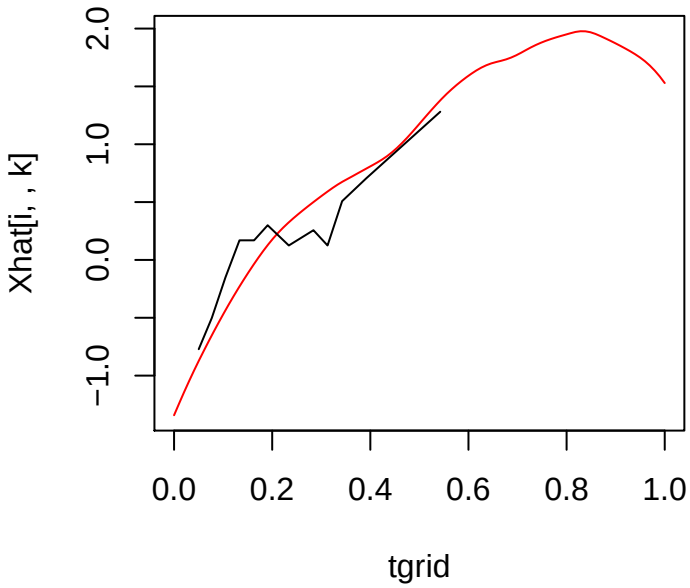
```
## i =  6 ; k =  1
```


Xhat[i, , k]
−1.0
0.0
1.0
2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

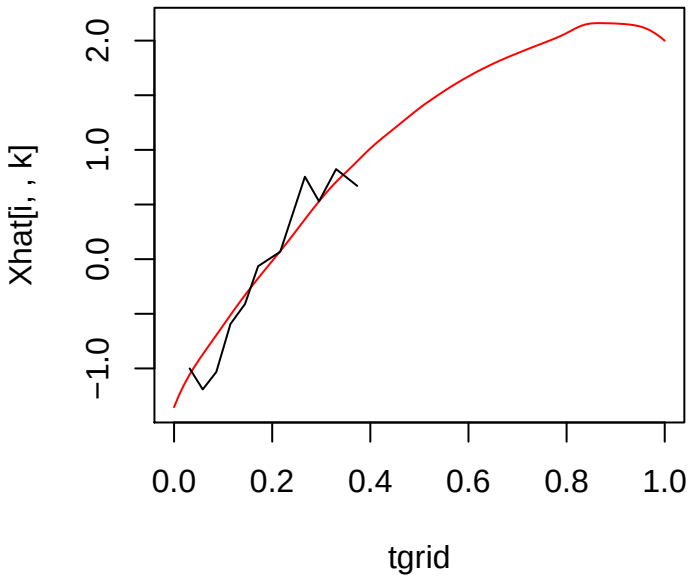
```
## i =  6 ; k =  2
```


Xhat[i, , k]
−0.5
0.0
0.5
1.0
1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

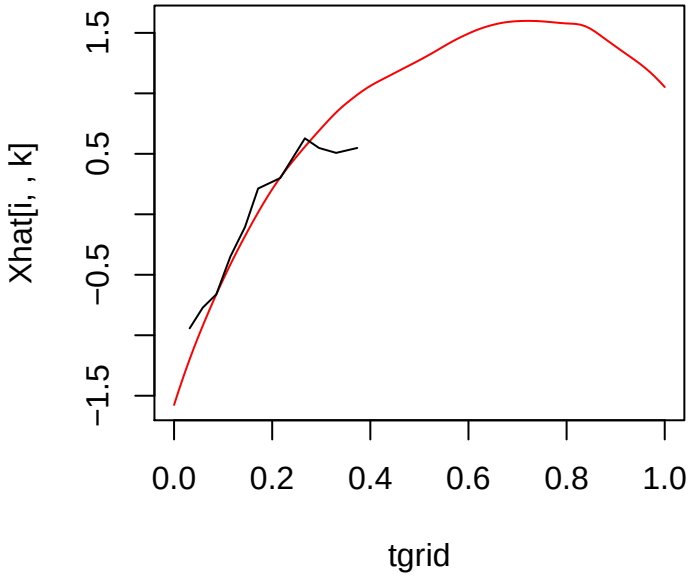
```
## i =  7 ; k =  1
```


Xhat[i, , k]
−1.0
0.0
1.0
2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

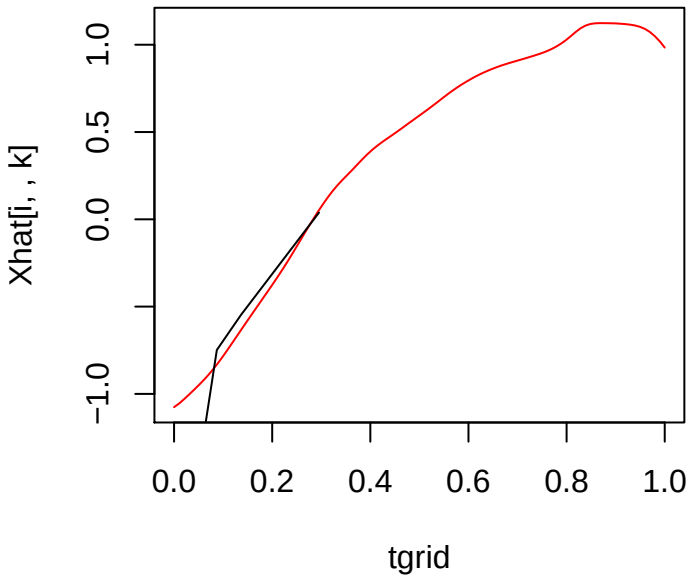
```
## i =  7 ; k =  2
```


Xhat[i, , k]
2.0
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

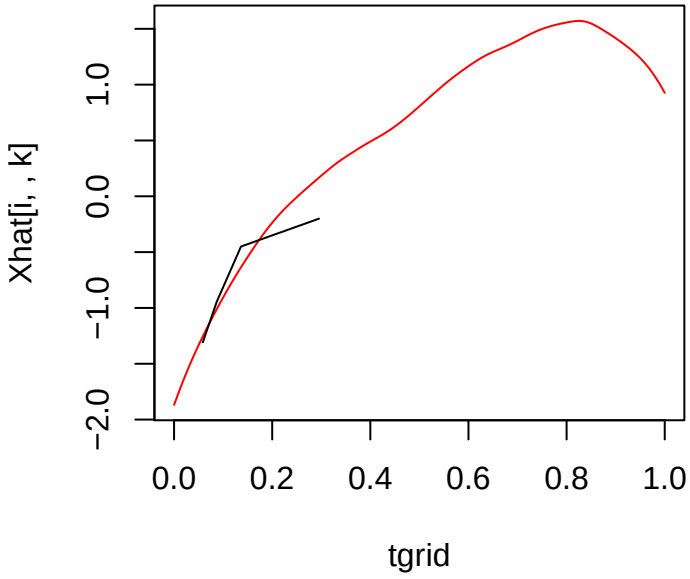
```
## i =  8 ; k =  1
```


Xhat[i, , k]
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

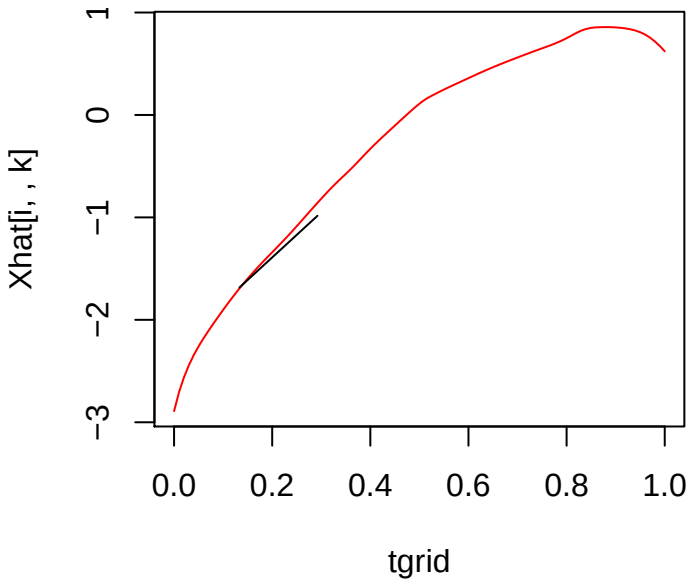
```
## i =  8 ; k =  2
```


Xhat[i, , k]
1.0
0.5
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

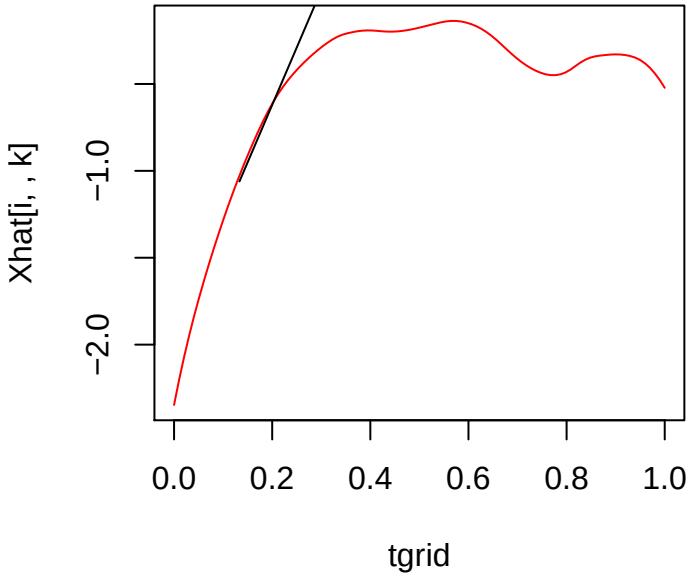
```
## i =  9 ; k =  1
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

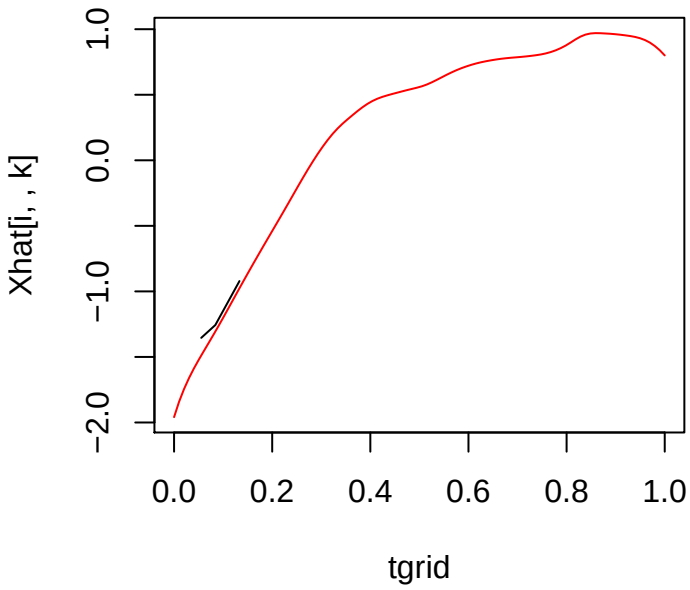
```
## i =  9 ; k =  2
```


Xhat[i, , k]
1
0
−1
−2
−3
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

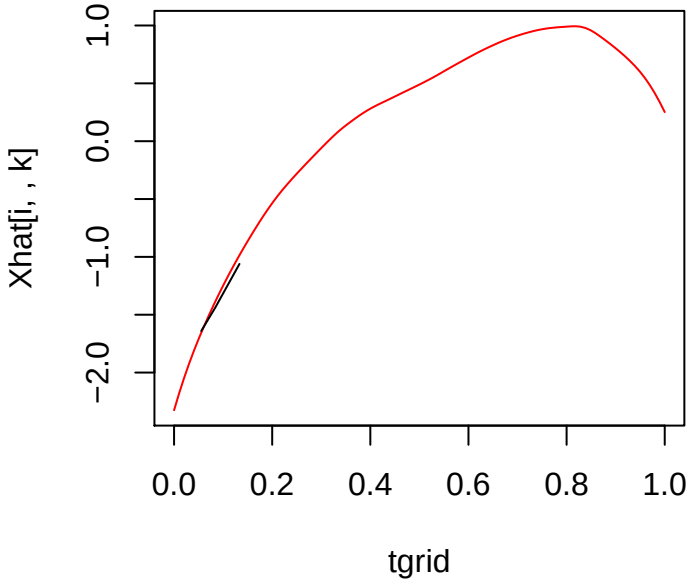
```
## i =  10 ; k =  1
```


Xhat[i, , k]
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

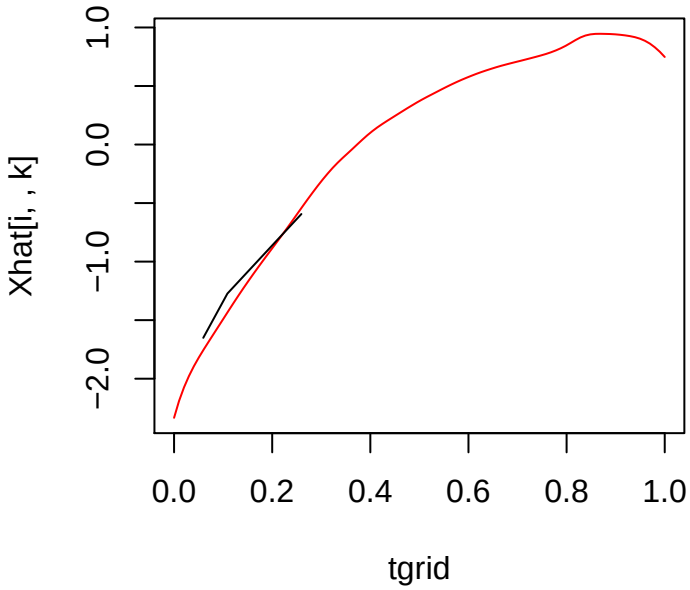
```
## i =  10 ; k =  2
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

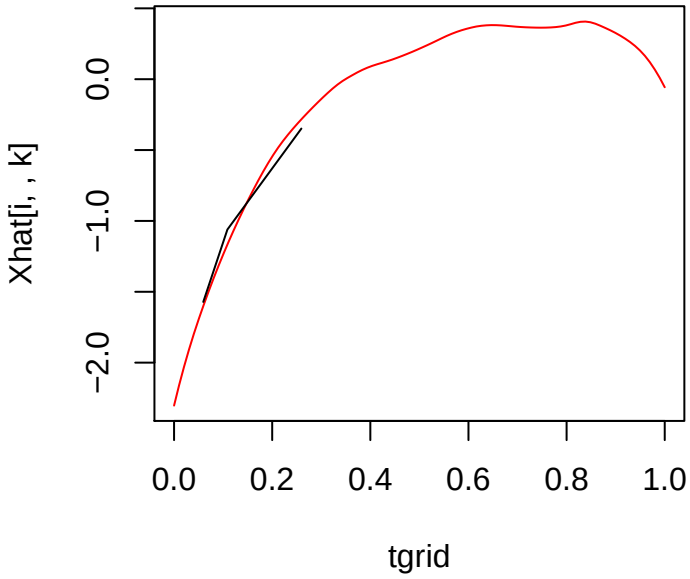
```
## i =  11 ; k =  1
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

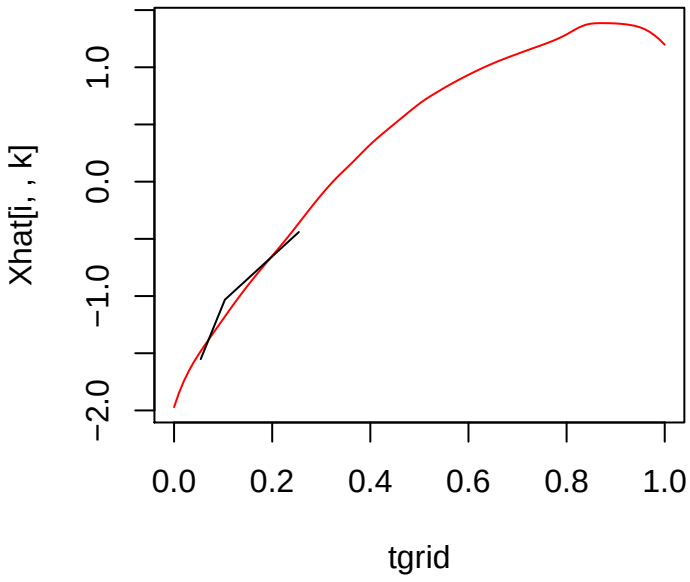
```
## i =  11 ; k =  2
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

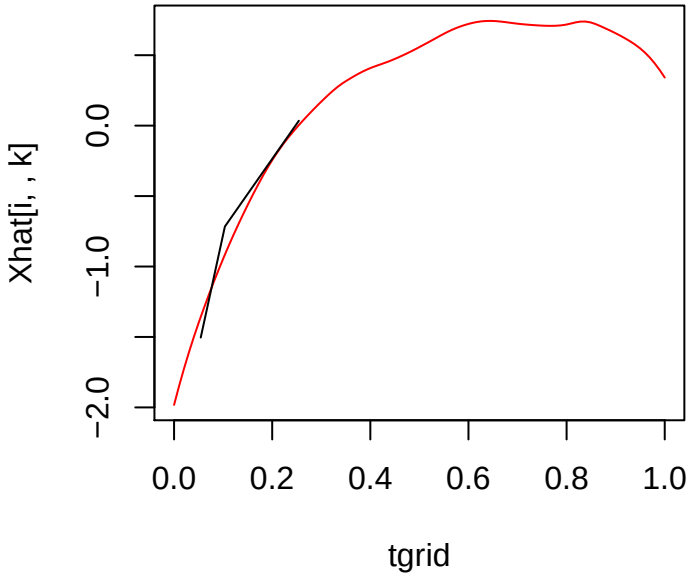
```
## i =  12 ; k =  1
```


Xhat[i, , k]
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

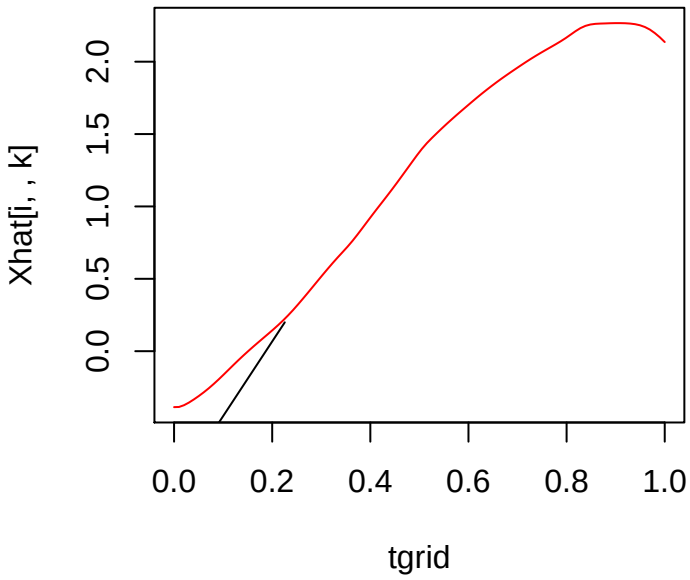
```
## i =  12 ; k =  2
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

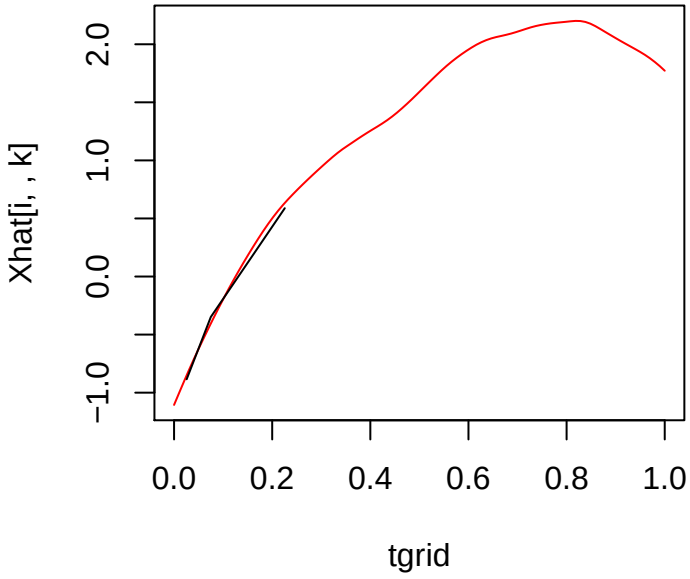
```
## i =  13 ; k =  1
```


Xhat[i, , k]
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

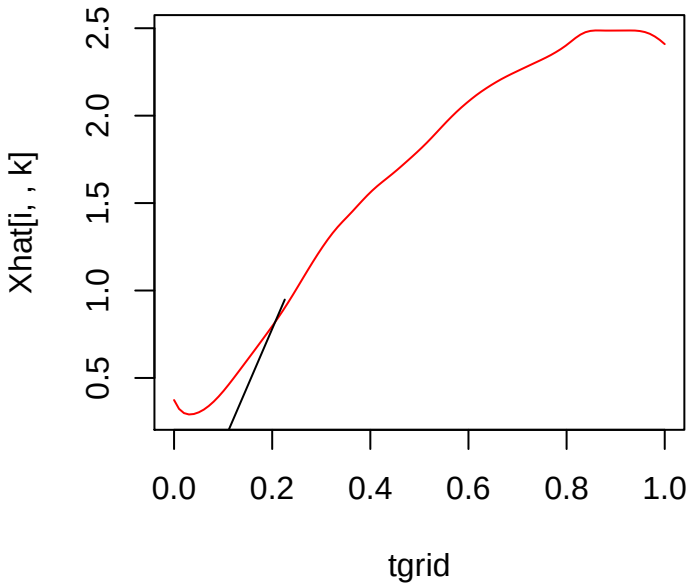
```
## i =  13 ; k =  2
```


Xhat[i, , k]
0.0
0.5
1.0
1.5
2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

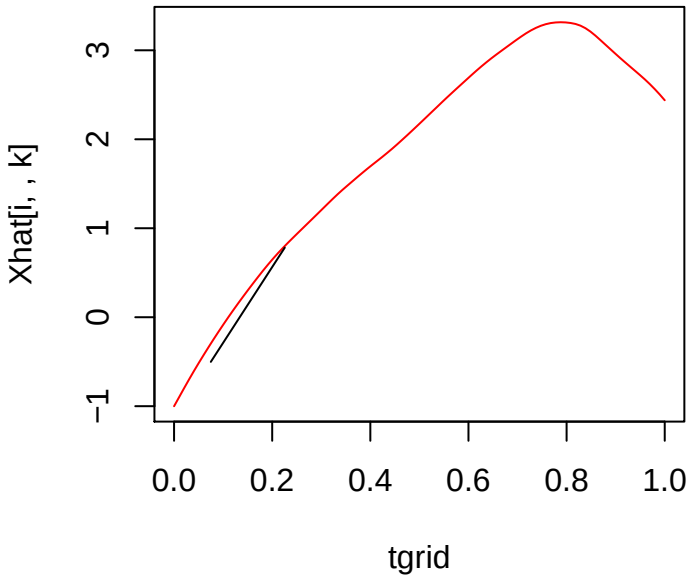
```
## i =  14 ; k =  1
```


Xhat[i, , k]
−1.0
0.0
1.0
2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

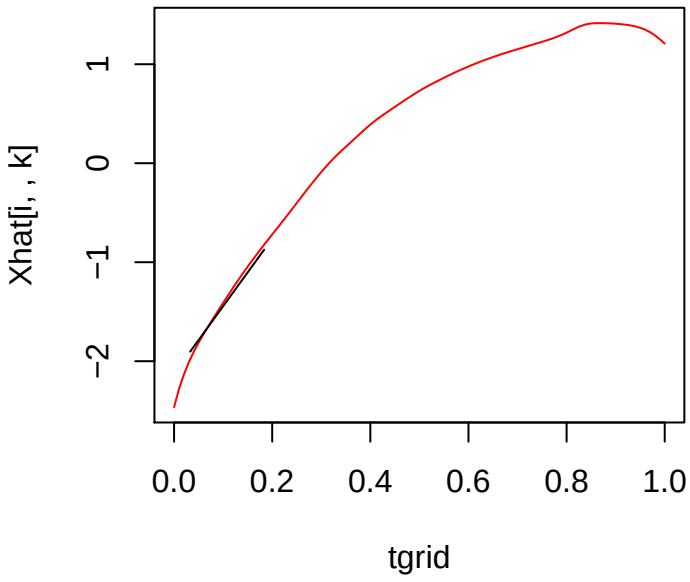
```
## i =  14 ; k =  2
```


Xhat[i, , k]
0.5
1.0
1.5
2.0
2.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

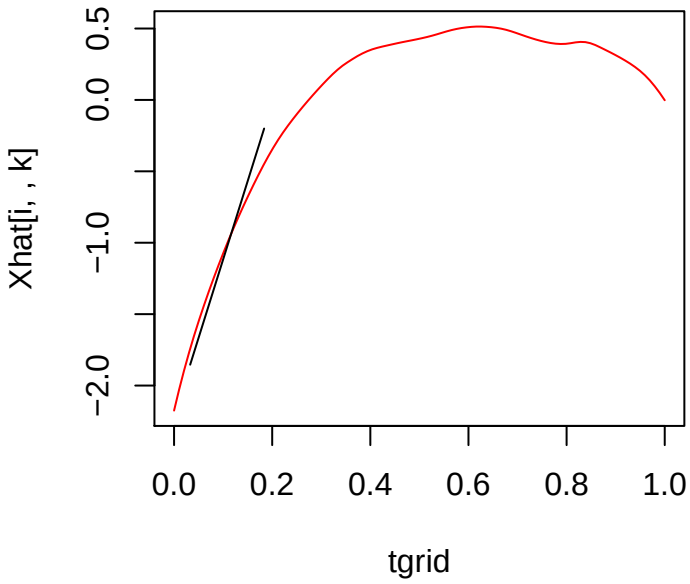
```
## i =  15 ; k =  1
```


Xhat[i, , k]
−1
0
1
2
3
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

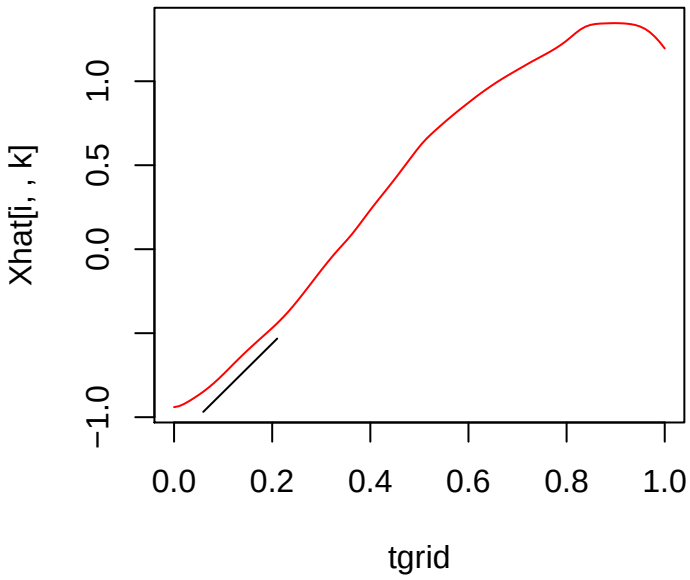
```
## i =  15 ; k =  2
```


Xhat[i, , k]
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

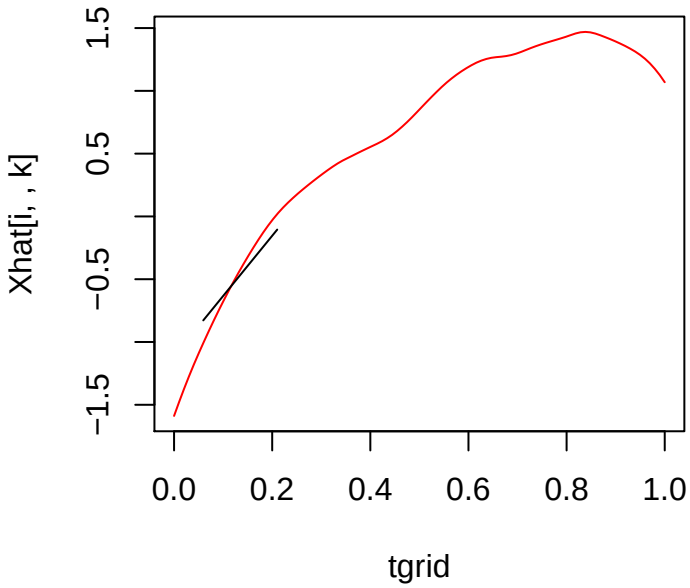
```
## i =  16 ; k =  1
```


Xhat[i, , k]
0.5
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

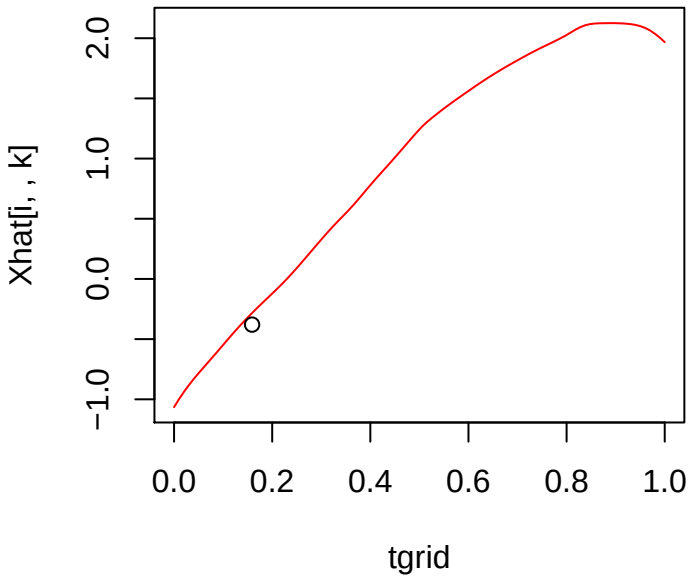
```
## i =  16 ; k =  2
```


Xhat[i, , k]
−1.0
0.0
0.5
1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

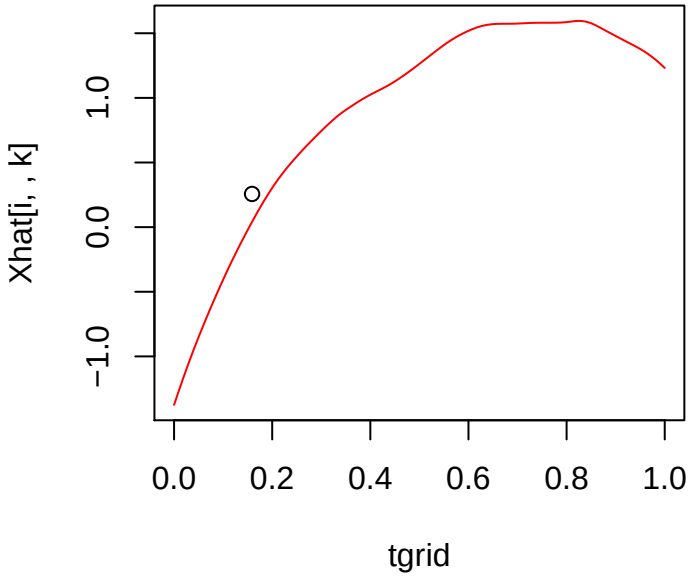
```
## i =  17 ; k =  1
```


Xhat[i, , k]
−1.5
−0.5
0.5
1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

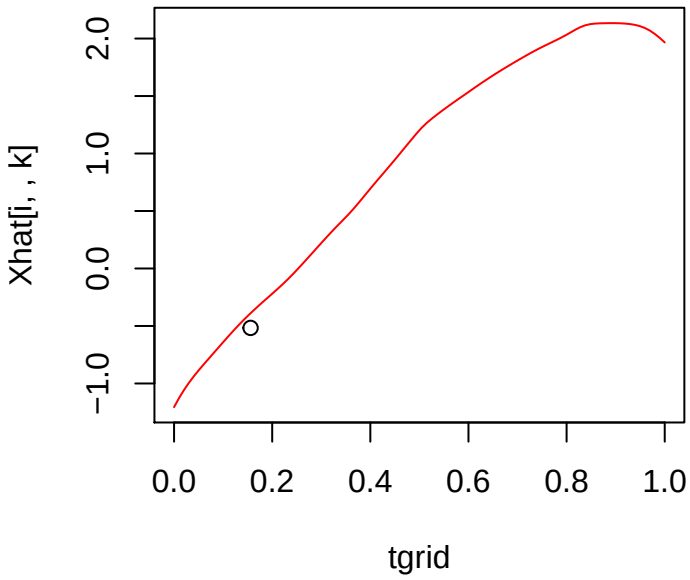
```
## i =  17 ; k =  2
```


Xhat[i, , k]
2.0
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

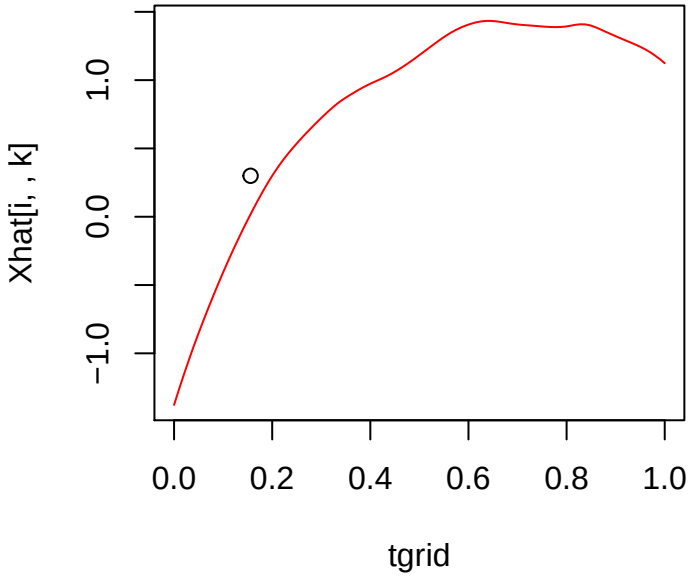
```
## i =  18 ; k =  1
```


Xhat[i, , k]
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

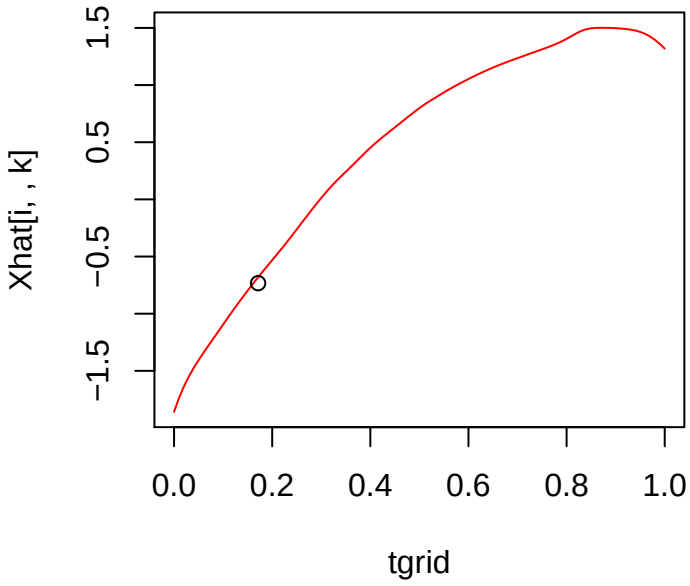
```
## i =  18 ; k =  2
```


Xhat[i, , k]
2.0
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

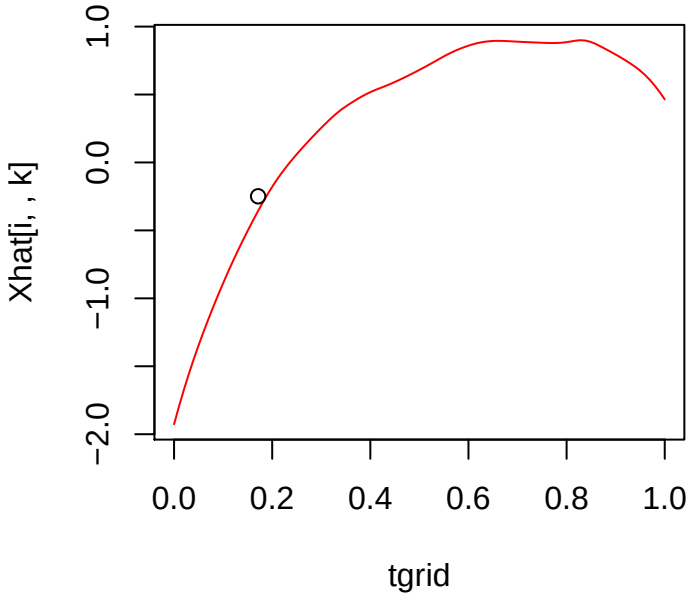
```
## i =  19 ; k =  1
```


Xhat[i, , k]
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

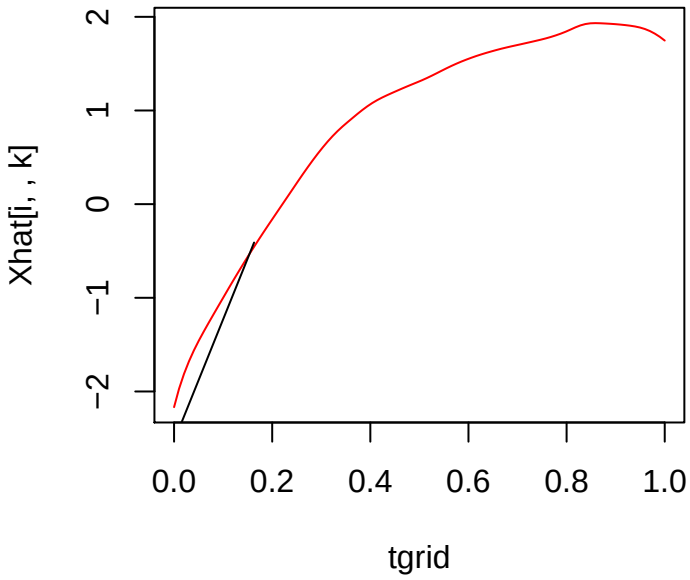
```
## i =  19 ; k =  2
```


Xhat[i, , k]
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

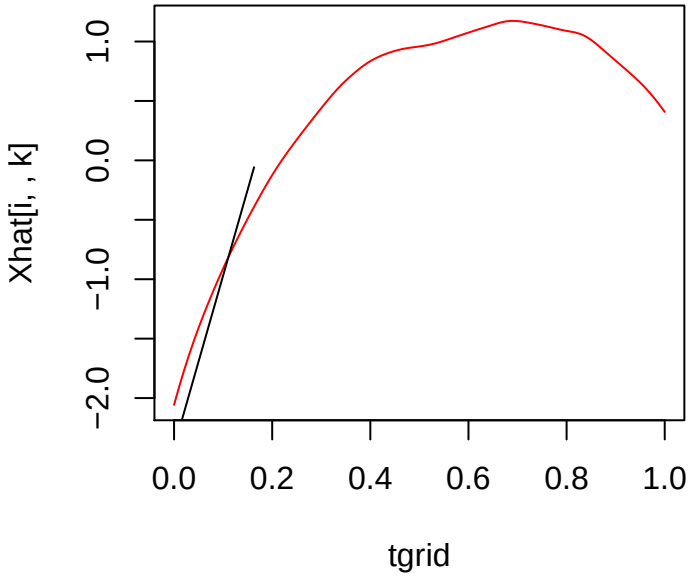
```
## i =  20 ; k =  1
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

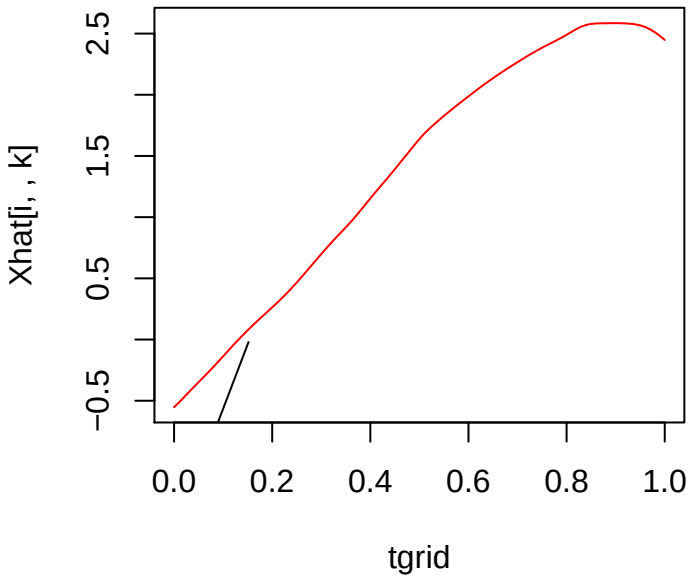
```
## i =  20 ; k =  2
```


Xhat[i, , k]
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

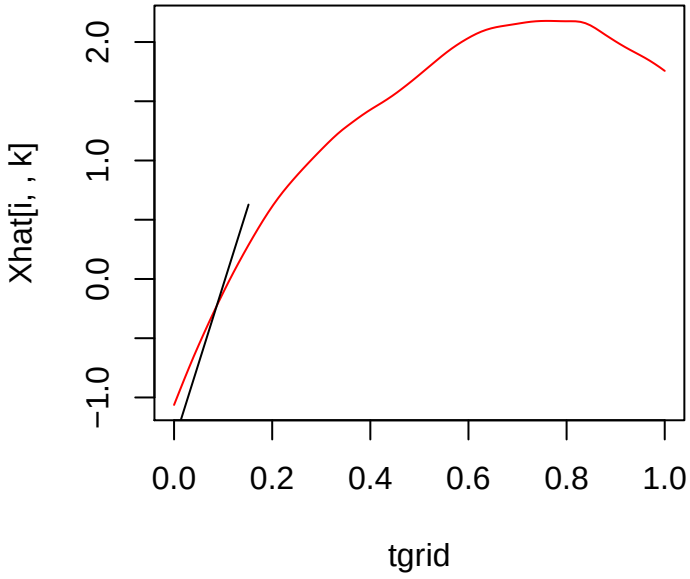
```
## i =  21 ; k =  1
```


Xhat[i, , k]
1.0
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

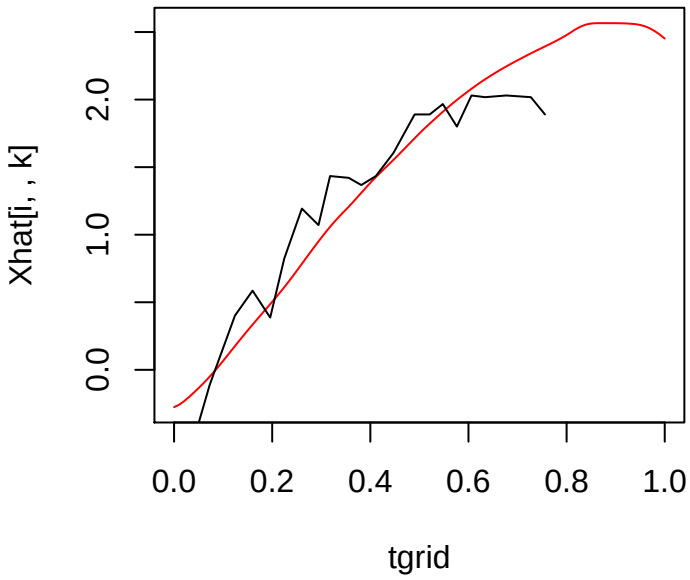
```
## i =  21 ; k =  2
```


Xhat[i, , k]
2.5
1.5
0.5
−0.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

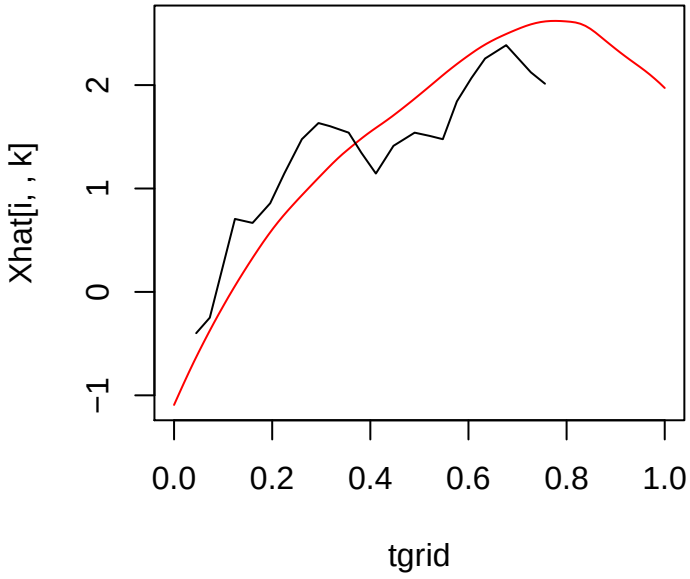
```
## i =  22 ; k =  1
```


Xhat[i, , k]
2.0
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

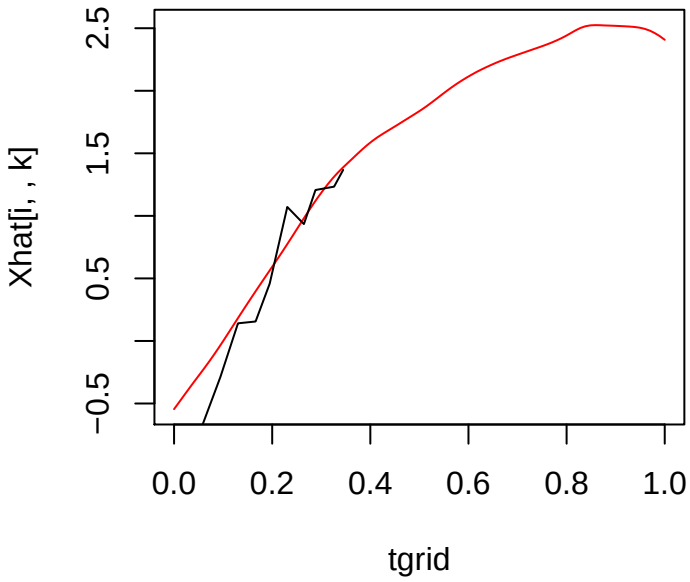
```
## i =  22 ; k =  2
```


Xhat[i, , k]
0.0
1.0
2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

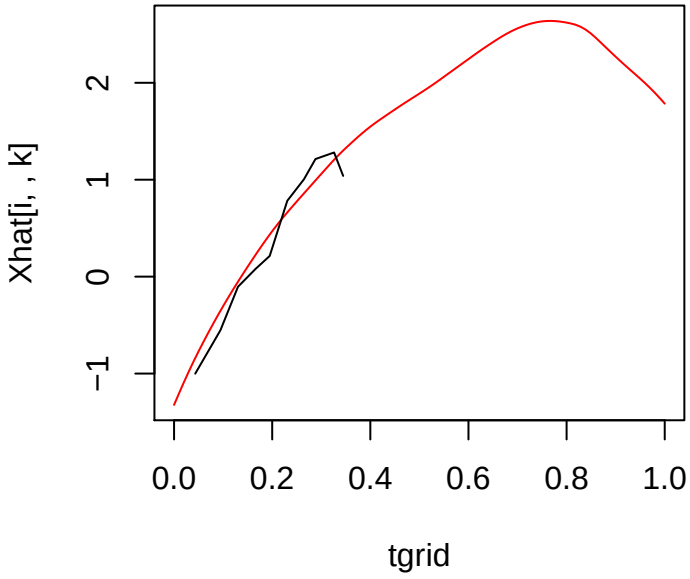
```
## i =  23 ; k =  1
```


Xhat[i, , k]
−1
0
1
2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

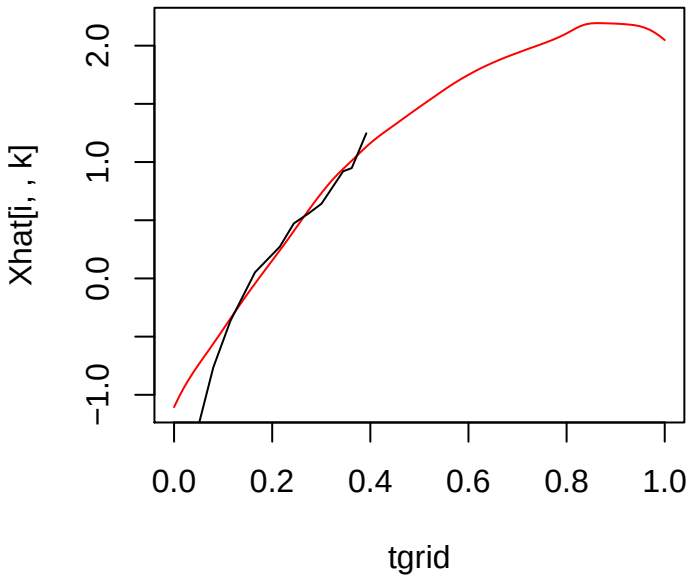
```
## i =  23 ; k =  2
```


Xhat[i, , k]
2.5
1.5
0.5
−0.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

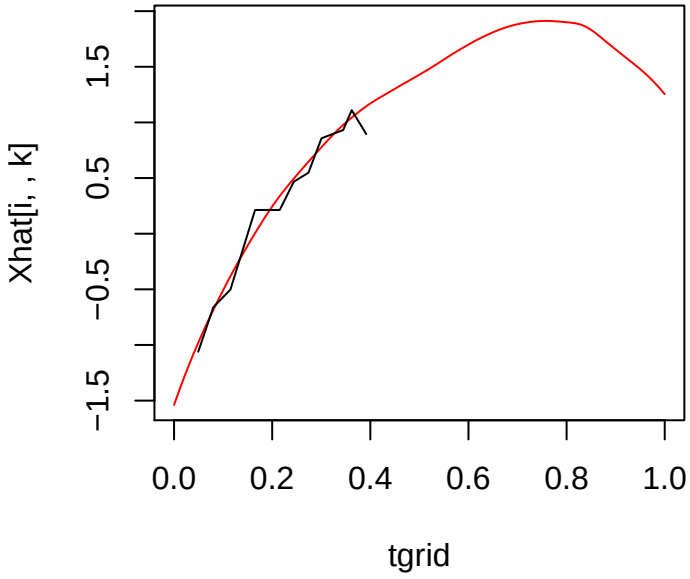
```
## i =  24 ; k =  1
```


Xhat[i, , k]
2
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

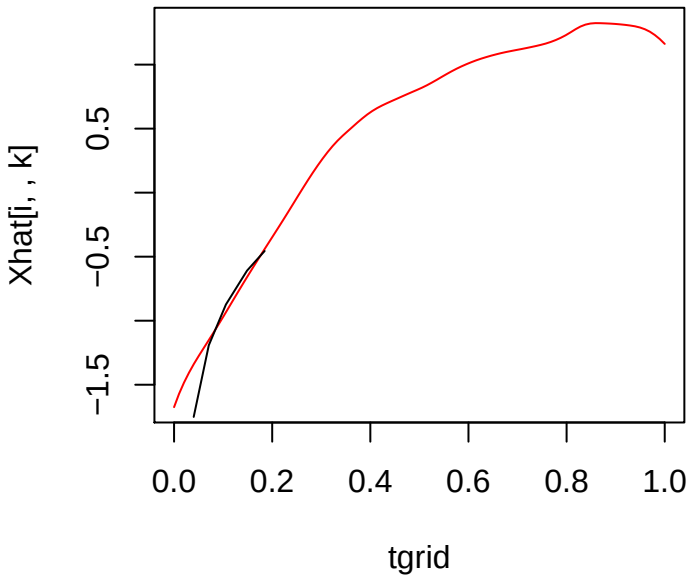
```
## i =  24 ; k =  2
```


Xhat[i, , k]
2.0
1.0
0.0
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

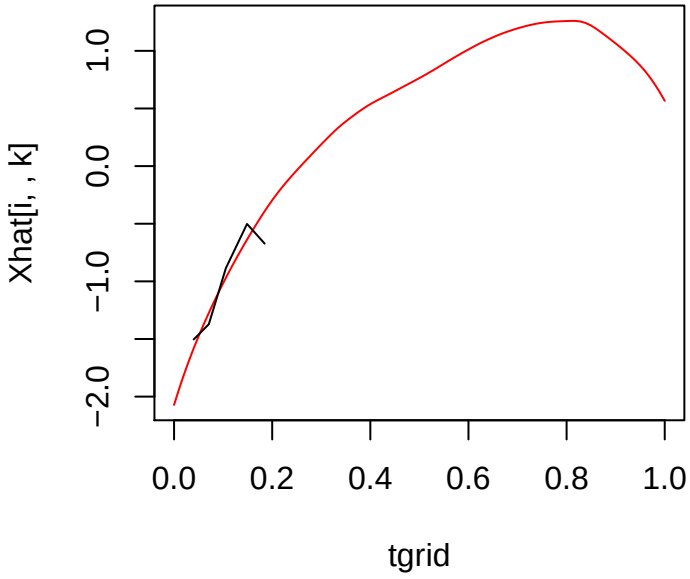
```
## i =  25 ; k =  1
```


Xhat[i, , k]
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

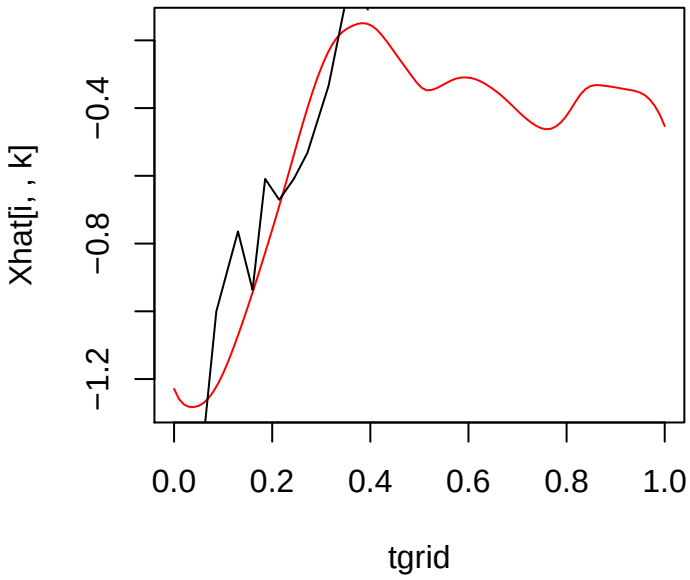
```
## i =  25 ; k =  2
```


Xhat[i, , k]
−1.5
−0.5
0.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

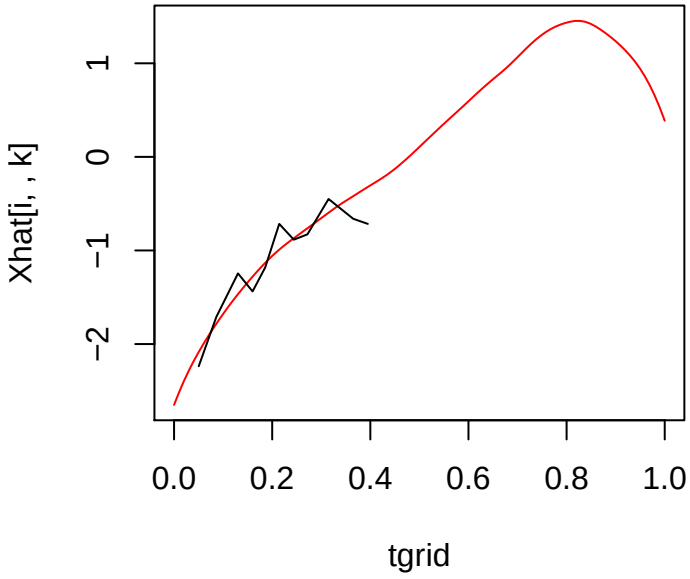
```
## i =  26 ; k =  1
```


Xhat[i, , k]
−2.0
−1.0
0.0
1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

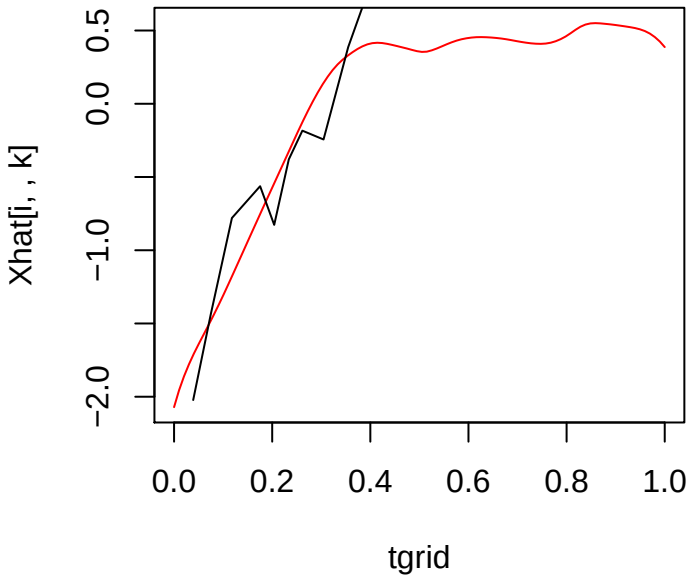
```
## i =  26 ; k =  2
```


Xhat[i, , k]
−0.4
−0.8
−1.2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

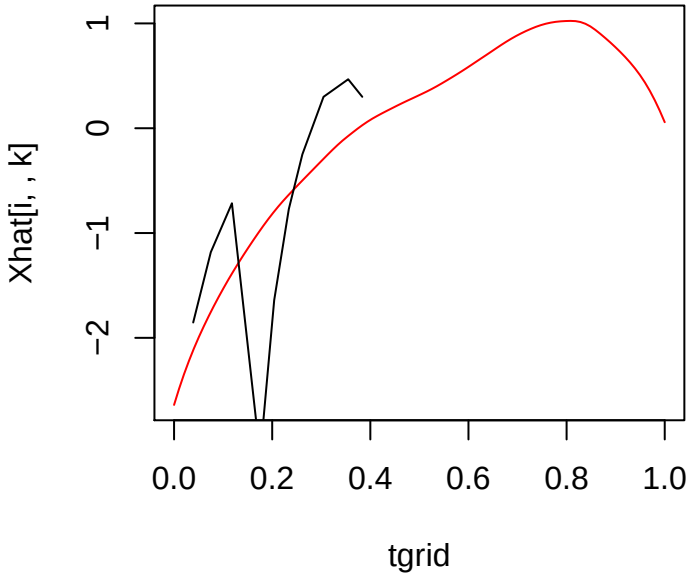
```
## i =  27 ; k =  1
```


Xhat[i, , k]
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

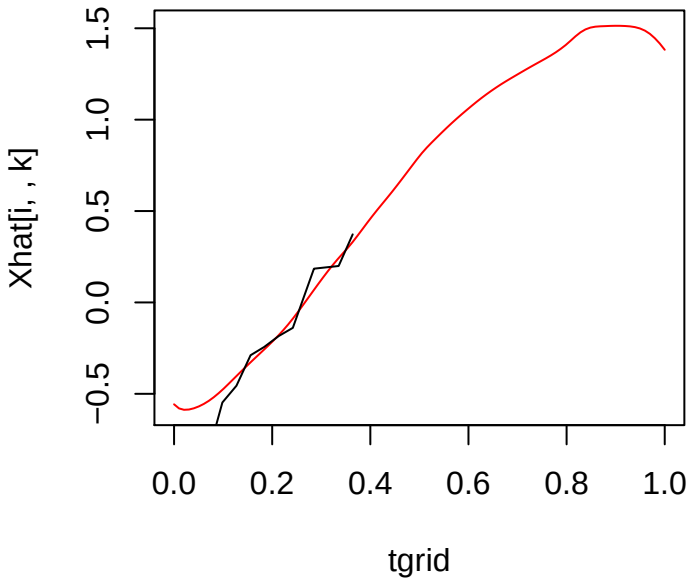
```
## i =  27 ; k =  2
```


Xhat[i, , k]
0.5
0.0
−1.0
−2.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

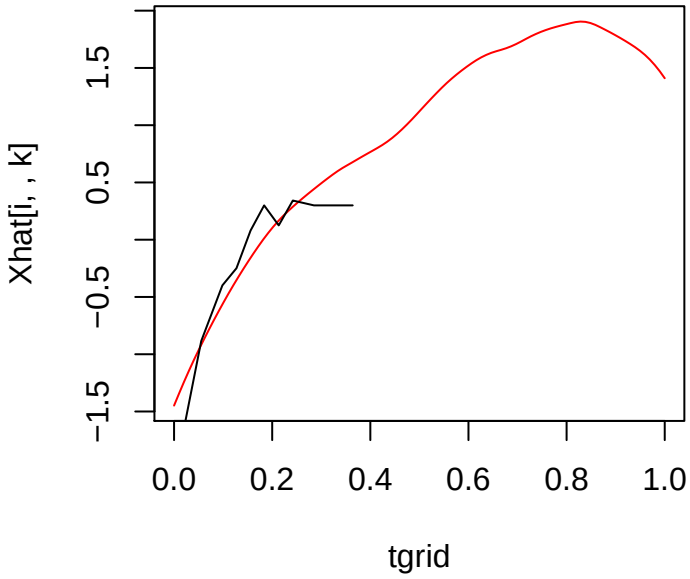
```
## i =  28 ; k =  1
```


Xhat[i, , k]
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

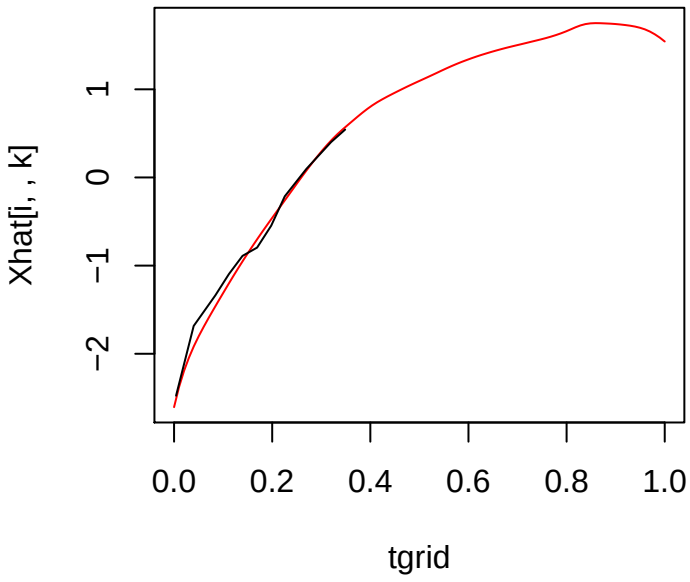
```
## i =  28 ; k =  2
```


Xhat[i, , k]
1.5
1.0
0.5
0.0
−0.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

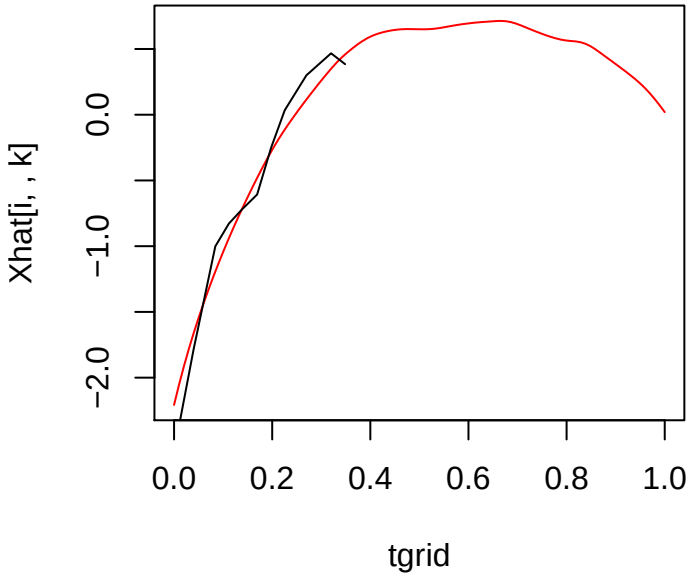
```
## i =  29 ; k =  1
```


Xhat[i, , k]
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

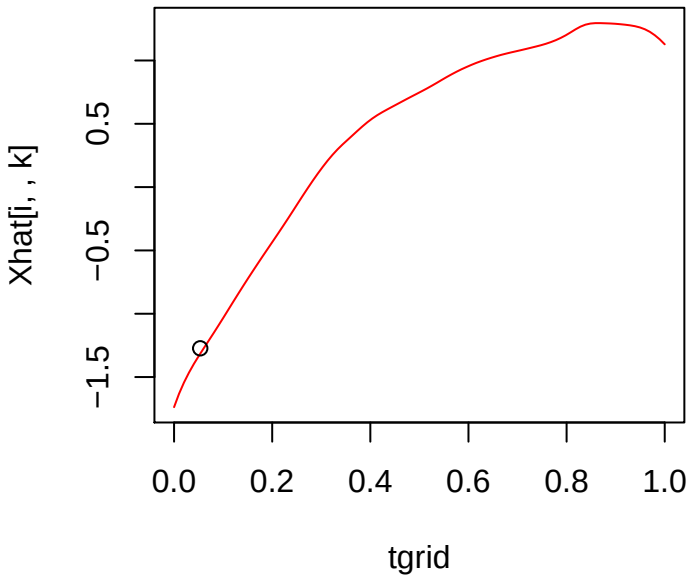
```
## i =  29 ; k =  2
```


Xhat[i, , k]
−2
−1
0
1
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

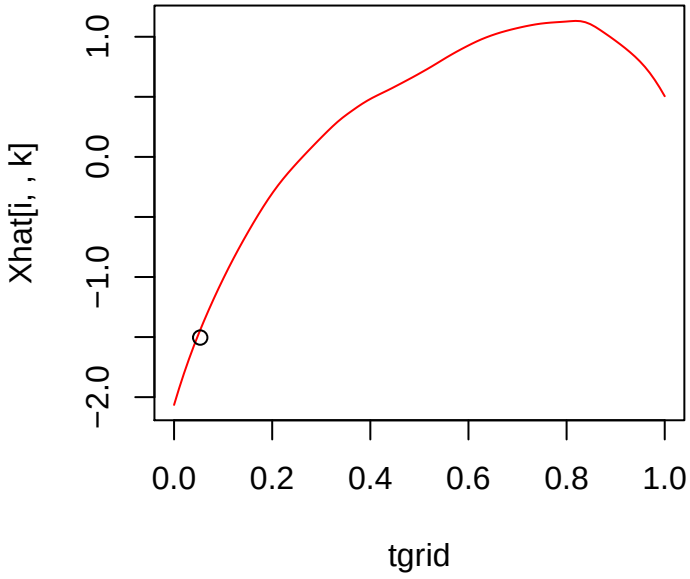
```
## i =  30 ; k =  1
```


Xhat[i, , k]
−2.0
−1.0
0.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

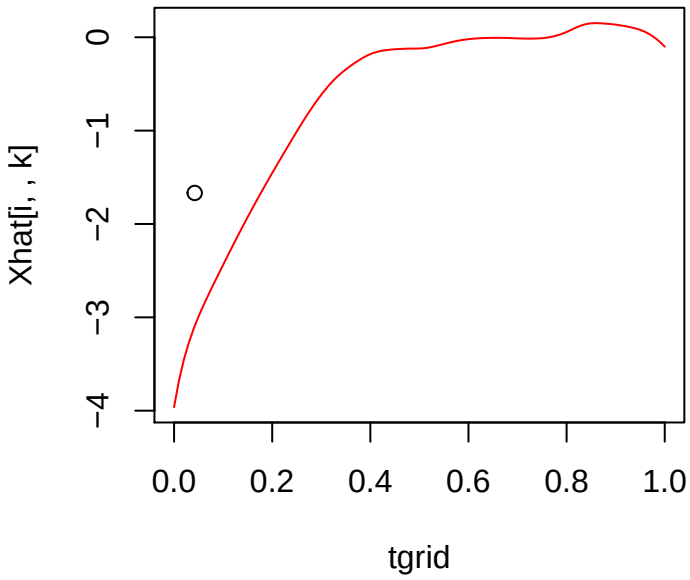
```
## i =  30 ; k =  2
```


Xhat[i, , k]
−1.5
−0.5
0.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

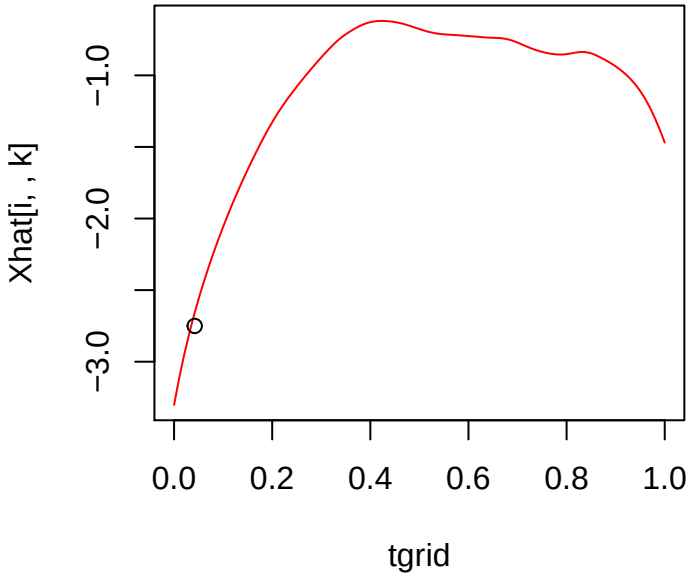
```
## i =  31 ; k =  1
```


Xhat[i, , k]
−2.0
−1.0
0.0
1.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

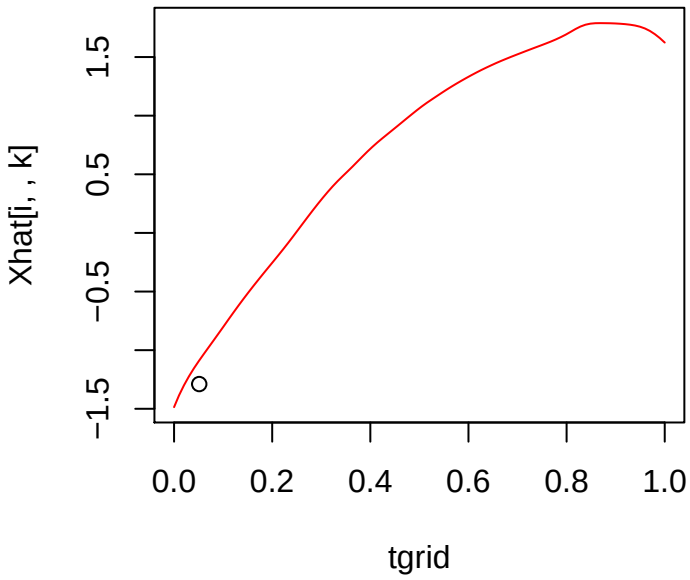
```
## i =  31 ; k =  2
```


Xhat[i, , k]
0
−1
−2
−3
−4
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

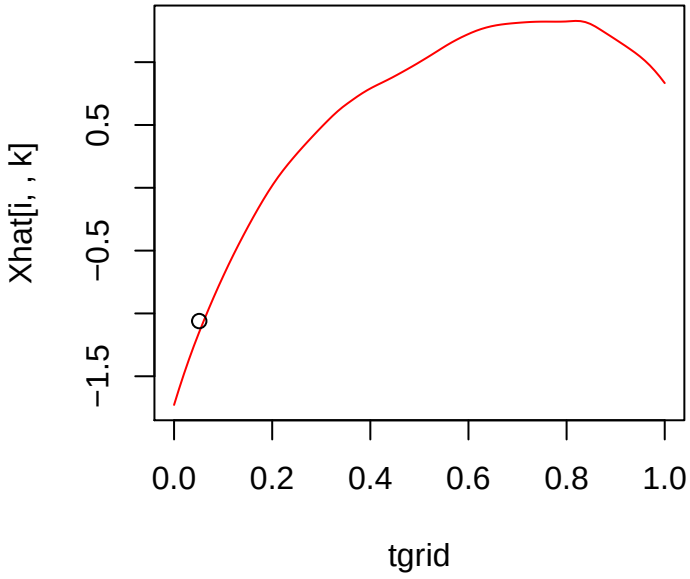
```
## i =  32 ; k =  1
```


Xhat[i, , k]
−1.0
−2.0
−3.0
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

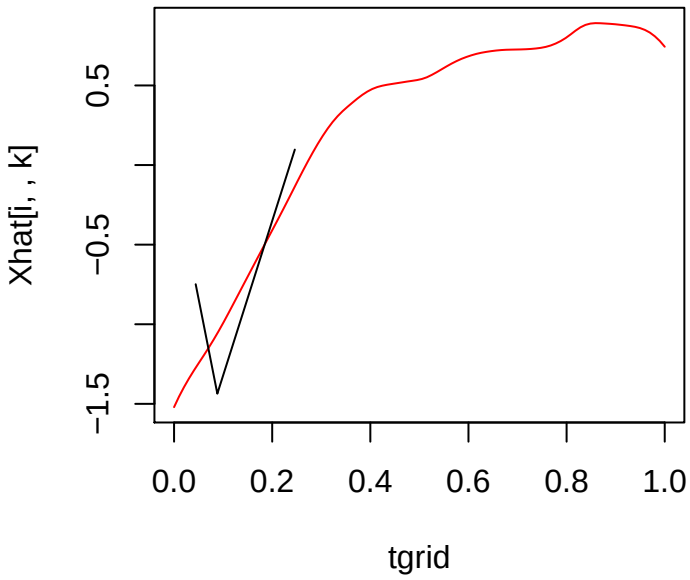
```
## i =  32 ; k =  2
```


Xhat[i, , k]
1.5
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

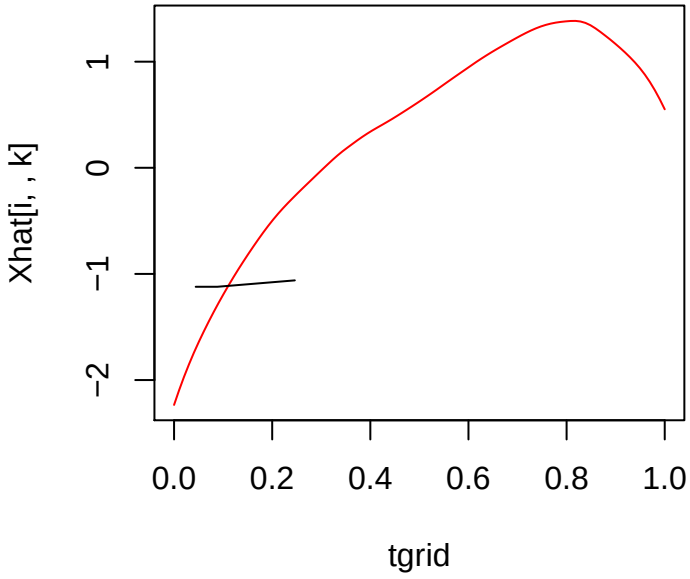
```
## i =  33 ; k =  1
```


Xhat[i, , k]
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

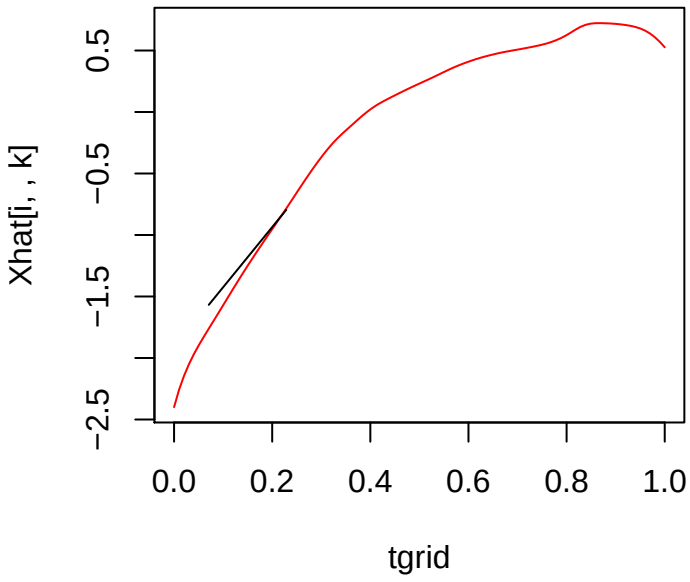
```
## i =  33 ; k =  2
```


Xhat[i, , k]
0.5
−0.5
−1.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

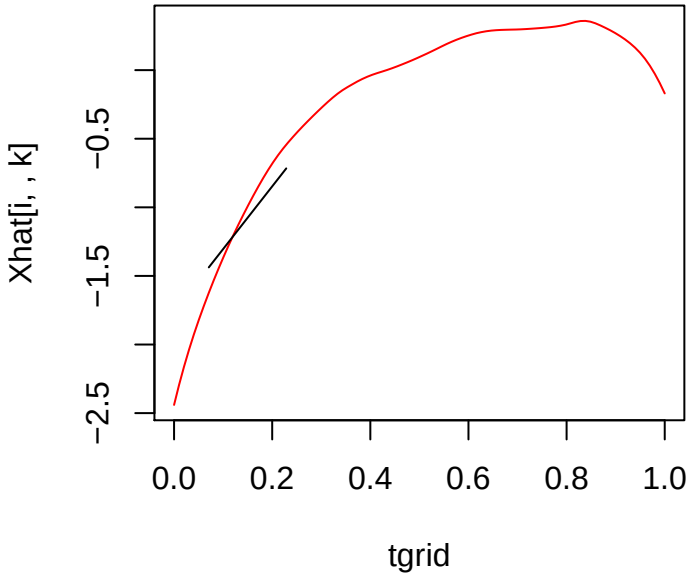
```
## i =  34 ; k =  1
```


Xhat[i, , k]
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

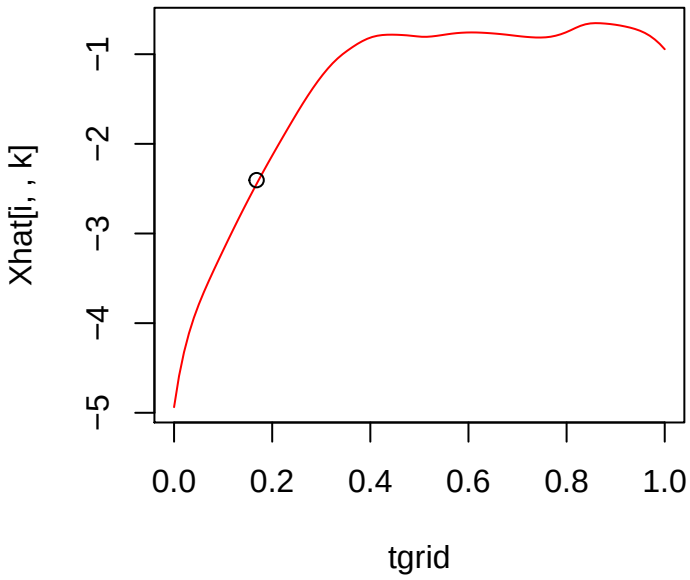
```
## i =  34 ; k =  2
```


Xhat[i, , k]
0.5
−0.5
−1.5
−2.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

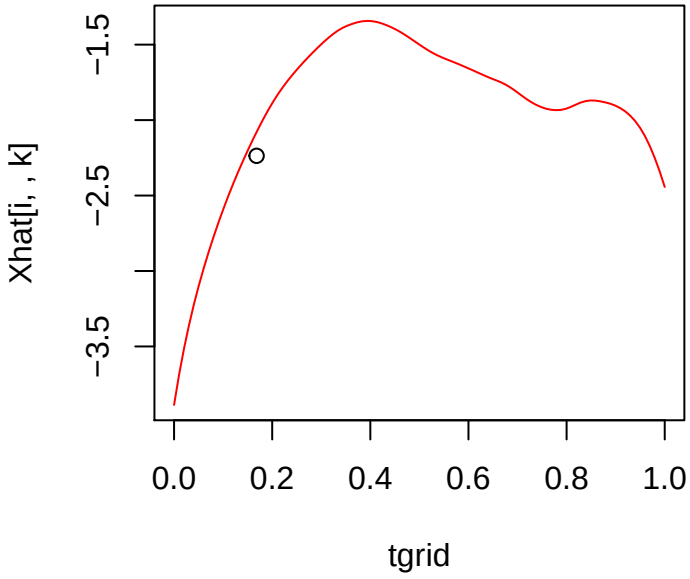
```
## i =  35 ; k =  1
```


Xhat[i, , k]
−0.5
−1.5
−2.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

```
## i =  35 ; k =  2
```


Xhat[i, , k]
−1
−2
−3
−4
−5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

```
## i =  36 ; k =  1
```


Xhat[i, , k]
−1.5
−2.5
−3.5
0.0
0.2
0.4
0.6
0.8
1.0
tgrid

```
## i =  36 ; k =  2
```