STA547: HW5

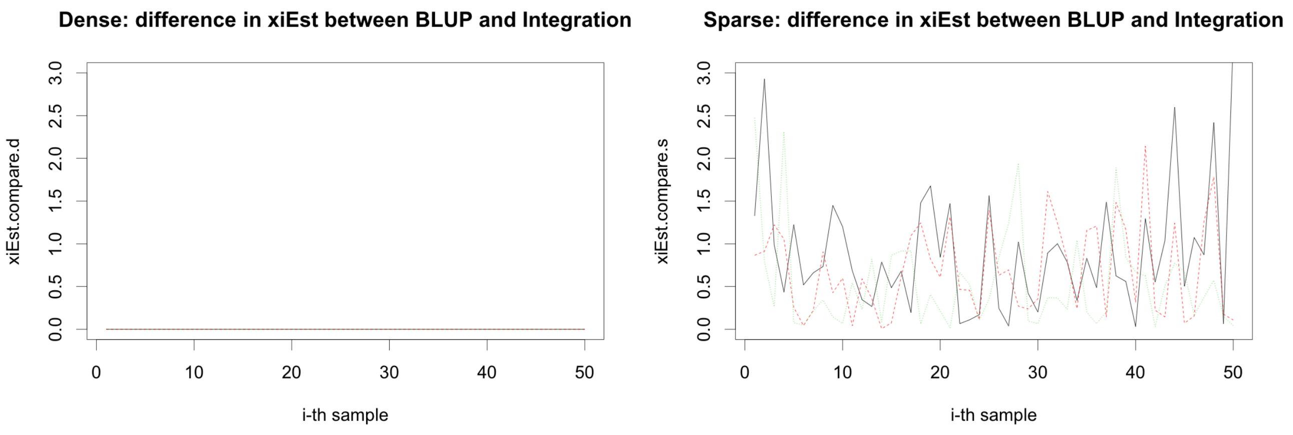
Zhiling Gu

1. (a) From the table we see the dense data leads to a better estimation of the number of components. In addition, FVE gives the highest K, while AIC and BIC give lower K which is expected since the later two methods incoporate penalty for high K.

	TRUE	FVE	AIC	BIC
dense	3	3	3	3
sparse	3	6	3	3
dense	20	20	20	20
sparse	20	7	3	1

```r
###  Q1   ###
n <- 50  # number of samples
M <- 100 # number of obs per sample
c0 <- 1  # coeff for eigen values
K <- 20  # number of eigen components
##   Simulate dense functional data
df.d <- MakeGPFunctionalData(n, M =M, mu = rep(0, M), K =K,
                     lambda = sapply(rep(1, K), function(x) {c0 * x^(-2)}), sigma = 0,
                         basisType = "cos")

##   Simulate sparse functional data
df.s <- MakeSparseGP(n, rdist = runif, sparsity = 2:9, muFun = function(x)
                    rep(0, length(x)), K = K, lambda = rep(1, K), sigma = 0,
                    basisType = "cos", CovFun = NULL)

## (a) Compare component number between different selection methods
#### dense case
df.d.Ly <- split(df.d$Y, row(df.d$Y)) # change the original matrix form into list
df.d.Lt <- rep(list(df.d$pts),n)
res.d1 <- FPCA(df.d.Ly,df.d.Lt, optns = list(methodSelectK = 'FVE'))
res.d2 <- FPCA(df.d.Ly,df.d.Lt, optns = list(methodSelectK = 'AIC'))
res.d3 <- FPCA(df.d.Ly,df.d.Lt, optns = list(methodSelectK = 'BIC'))
res.compare.d <- data.frame("TRUE" = K,"FVE" = res.d1$selectK,"AIC" = res.d2$selectK,"BIC"
     = res.d3$selectK )
rownames(res.compare.d) <- "Number of Component: dense"
res.compare.d

#### sparse case
res.s1 <- FPCA(df.s$Ly,df.s$Lt, optns = list(methodSelectK = 'FVE'))
res.s2 <- FPCA(df.s$Ly,df.s$Lt, optns = list(methodSelectK = 'AIC'))
res.s3 <- FPCA(df.s$Ly,df.s$Lt, optns = list(methodSelectK = 'BIC'))
res.compare.s <- data.frame("TRUE" = K,"FVE" = res.s1$selectK,"AIC" = res.s2$selectK,"BIC"
     = res.s3$selectK )
rownames(res.compare.s) <- "Number of Component: sparse"
res.compare.s
```

(b) Use $K = 3, M = 20$ for the ease of computation. From the table of the absolute difference between two methods, we see the FPC scores estimates are close for dense case, but differ for sparse case. Aside, the integration methods is significantly faster than BLUP method.



```r
## (b) & (c) Compare estimated FPC scores between integration method and BLUP
#### dense case
res.d1 <- FPCA(df.d.Ly,df.d.Lt, optns = list(methodSelectK = 'FVE', methodXi = 'CE'))
res.d2 <- FPCA(df.d.Ly,df.d.Lt, optns = list(methodSelectK = 'FVE', methodXi = 'IN'))
xiEst.compare.d <- abs(res.d1$xiEst-res.d2$xiEst)

#### sparse case
res.s1 <- FPCA(df.s$Ly,df.s$Lt, optns = list(methodSelectK = 'FVE',maxK = K, methodXi = '
    CE'))
res.s2 <- FPCA(df.s$Ly,df.s$Lt, optns = list(methodSelectK = 'FVE',maxK = K, methodXi = '
    IN'))
xiEst.compare.s <- abs(res.s1$xiEst-res.s2$xiEst)

png("q1bc.png",width = 2000, height = 800, units = "px", pointsize = 30)
par(mfrow= c(1,2))
matplot(xiEst.compare.d, type = 'l', ylim = c(0,3), xlab = "i-th sample", main = "Dense:
    difference in xiEst between BLUP and Integration")
matplot(xiEst.compare.s, type = 'l', ylim = c(0,3), xlab = "i-th sample", main = "Sparse:
    difference in xiEst between BLUP and Integration")
dev.off()
```

2. Extend the PACE method for bivariate sparsely observed longitudinal data, and create a rudimentary implementation. Apply your implementation to analyze the bivariate process of weight and body length growth of Tammar wallabies. Since the scales of the two processes vastly differ you may want to normalize them before the analysis.

   Consider the following process:

$$Y_{ij} = X_i(T_{ij}) + \epsilon_{ij} = \mu(T_{ij}) + \sum_{r=1}^{\infty} \xi_{ri}\{(D\phi_r)(T_{ij})\} + \epsilon_{ij}$$

   where $i = 1, \dots, 166$. Y_{ij} is the j-th subject observed at T_{ij}. $\{X_i\}_{i=1,\dots,166}$ are sampled from a bivariate process X in $\mathbb{H}$,

$$X_i(t) = (X_{1i}(t), \dots X_{pi}(t))^T, \quad p = 2 \text{ for bivariate process}$$

   Here we denote weight of as X_{1i}, and the length as X_{2i}, the observed j-th weights and lengths are Y_{1ij} and Y_{2ij} respectively. and $\epsilon_{ij} = (\epsilon_{1ij}, \dots, \epsilon_{pij})^T$ are distributed independely with mean 0 and variance $\sigma^2 = (\sigma_1, \dots, \sigma_p^2)^T$.

   We also introduce a normalized process of X_i: $Z_i(t) = (Z_{1i}(t), \dots Z_{pi}(t))^T$ where

$$Z_{ki}(t) = v_k(t)^{-1/2}\{X_{ki}(t) - \mu_k(t)\}$$

   and a normalized process of Y_i: $U_{ij} = (U_{1ij}, \dots, U_{pij})^T$ where

$$U_{kij} = v_k(T_{ij})^{-1/2}(Y_{kij} - \mu_k(T_{\{}ij\}))$$

thereafter we can rewrite the previous process as

$$U_{ij} = Z_i(T_{ij}) + \varepsilon_{ij} = \sum_{r=1}^{\infty} xi_{ri}\phi_r(T_{ij}) + \varepsilon_{ij}$$

where $\varepsilon_{ij} = (\varepsilon_{1ij}, \dots, \varepsilon_{pij})^T$ are mutully independent with mean 0 and variance $\zeta_{\{}ij\}^2 = (\zeta_1, \dots, \zeta_p^2)^T$
Consider

$$\begin{aligned}
C(s,t) &= \{C_{kl}(s,t)\}_{1\le k,l\le p} \\
C_k(s,t) &= (C_{k1}(s,t)\dots, C_{kp}(s,t))^T \\
C_{kl}(s,t) &= \{v_k(s)v_l(t)\}^{-1/2}G_{kl}(s,t) \\
G_{kl}(s,t) &= cov(X_k(s), X_l(t)) \\
v_k(t) &= G_{kk}(t) \\
D(t) &= diag(v_1(t)^{1/2}, \dots, v_p(t)^{1/2}) \\
\phi_r &= (\phi_{1r}, \dots, \phi_{pr})^T,\ r = 1, 2, \dots \text{ is a set of orthonormal basis functions in } \mathbb{H}
\end{aligned}$$

Define the integral operator $\mathcal{A} : \mathbb{H} \to \mathbb{H}$ s.t.

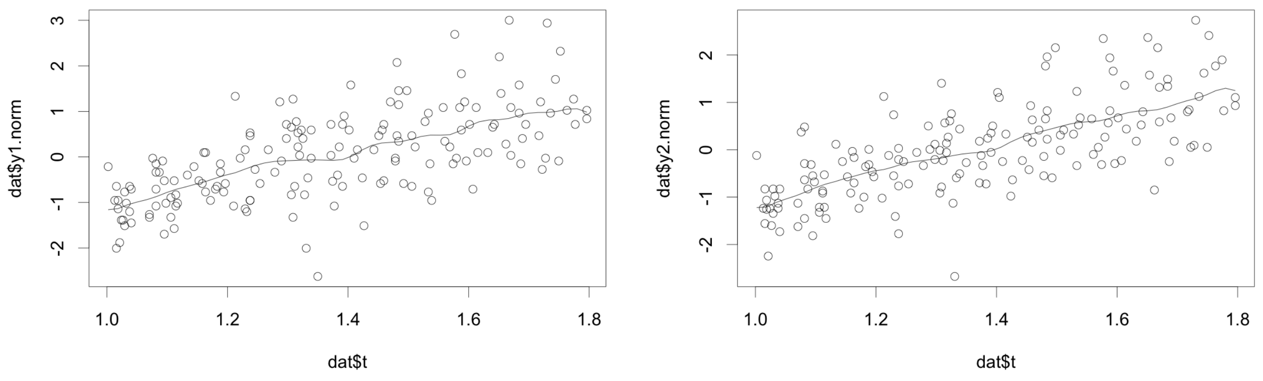
$$(\mathcal{A}f)(s) = \int C(s,t)f(t)dt = \begin{pmatrix} \langle C_1(s,\cdot), f\rangle_{\mathbb{H}} \\ \vdots \\ \langle C_p(s,\cdot), f\rangle_{\mathbb{H}} \end{pmatrix}$$

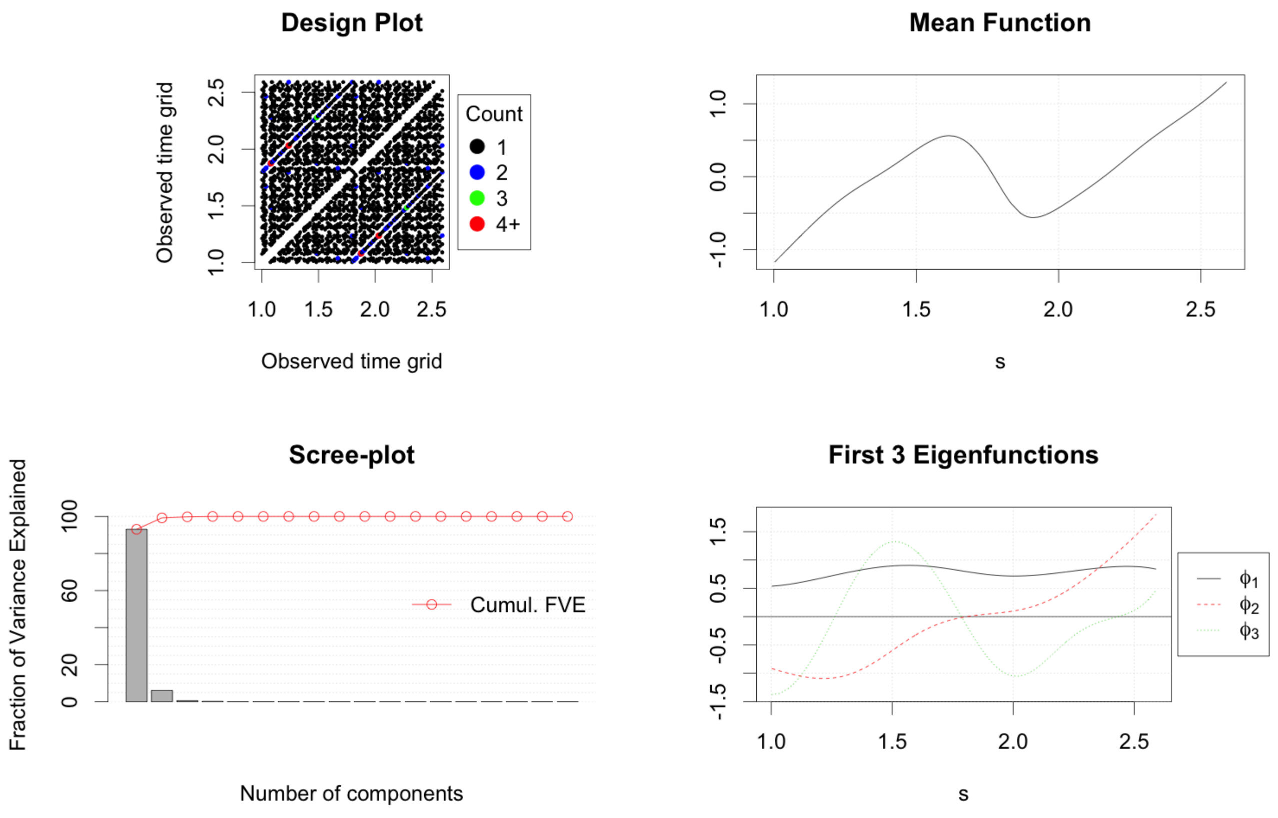
where $\langle C_k(s,\cdot), f\rangle_{\mathbb{H}} = \sum_{l=1}^{p}\langle C_{kl}(s,\cdot), f_l\rangle_{\mathbb{H}}$

From the paper, we can stack the two process and retrieve the principal components for each random variable by looking for the corresponding entries.

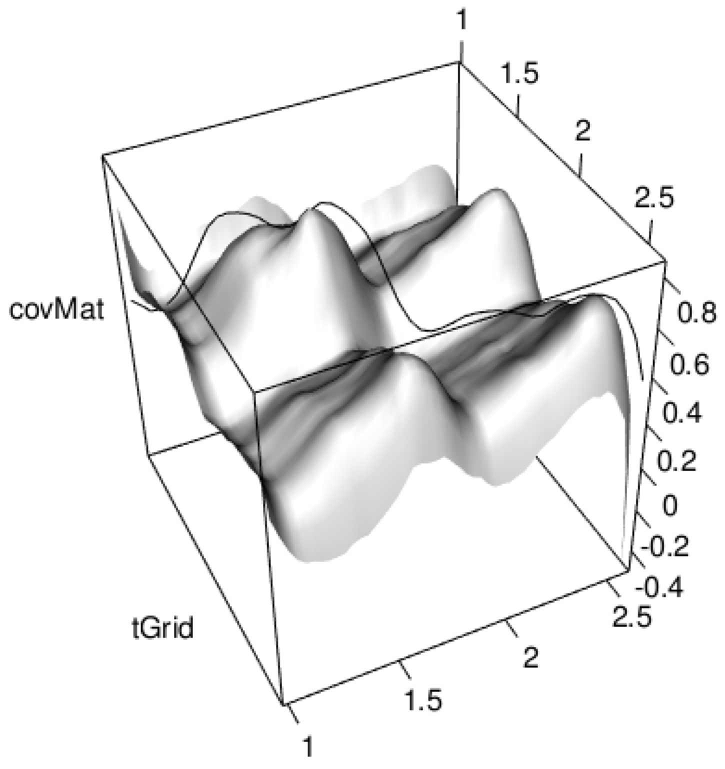
```r
###  Q2   ###
## Extend the PACE method for bivariate sparsely observed longitudinal data
dat <- read.delim('http://www.statsci.org/data/oz/wallaby.txt') %>%
  select(Age, Weight, Leng, Anim) %>%  # select takes the columns
  mutate(Age = Age/365.24 , Weight=Weight , Leng = Leng ) %>%
  na.omit %>%
  filter(Age >= truncAge[1] & Age <= truncAge[2]) %>%
  dplyr::rename(t = Age, y1 = Weight, y2 = Leng,   id = Anim)

ngrid <- 50
hmu <- 0.1

## Data Normalizing
dat$y1.norm <- scale((dat$y1))
dat$y2.norm <- scale(dat$y2)
# dat$y1.norm <- (dat$y1 - min(dat$y1))/diff(range(dat$y1))
# dat$y2.norm <- (dat$y2 - min(dat$y2))/diff(range(dat$y2))

## Mean function estimation
resMu1 <- locfit(y1.norm ~ lp(t, deg=1, h=hmu), dat, ev=lfgrid(mg=ngrid))
resMu2 <- locfit(y2.norm ~ lp(t, deg=1, h=hmu), dat, ev=lfgrid(mg=ngrid))
muObs1 <- predict(resMu1, dat$t) # interpolate
muObs2 <- predict(resMu2, dat$t) # interpolate

muHat1 <- predict(resMu1)
muHat2 <- predict(resMu2)
png("q2mu.png",width = 2200, height = 800, units = "px", pointsize = 30)
par(mfrow=c(1,2))
plot(dat$t, dat$y1.norm)
lines(tGrid, muHat1)
plot(dat$t, dat$y2.norm)
lines(tGrid, muHat2)
dev.off()

## Dataframe stacking
```

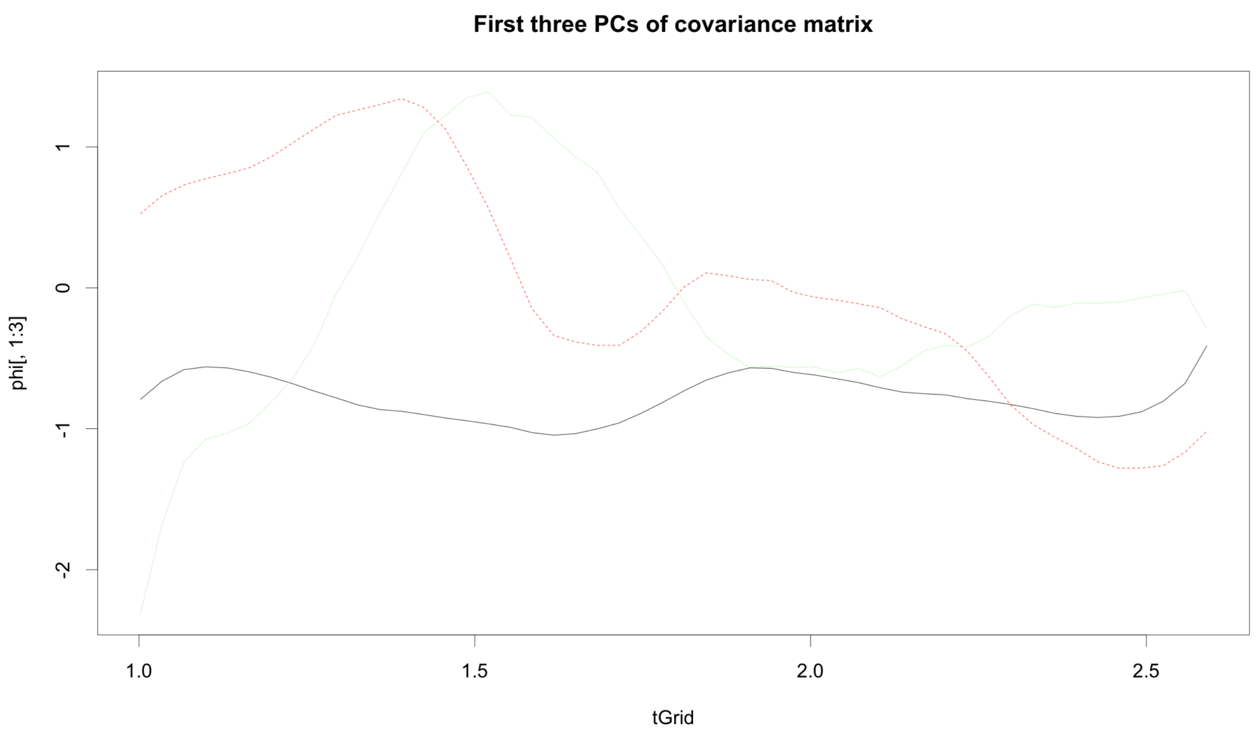
```r
#### Note the time for y_2 is shifted, otherwise error of duplicate time t would occur
#### Also note the shift in time t would not affect the estimation of mean function, covariance function or eigen function.
#### It would only affect the FPC scores.
df <- data.frame('t' = c(dat$t, dat$t+diff(range(dat$t))), 'y' =  c(dat$y1, dat$y2), 'id' =c(dat$id,dat$id))
tGrid <- seq(min(df$t), max(df$t), length.out=ngrid)  # working grid
samp <- MakeFPCAInputs(df$id, df$t, df$y)
res <- FPCA(samp$Ly, samp$Lt, list(userBwMu=0.1, userBwCov=0.2, FVEthreshold=1))
png("q2fpca.png",width = 1500, height = 1000, units = "px", pointsize = 30)
plot(res)
dev.off()
```



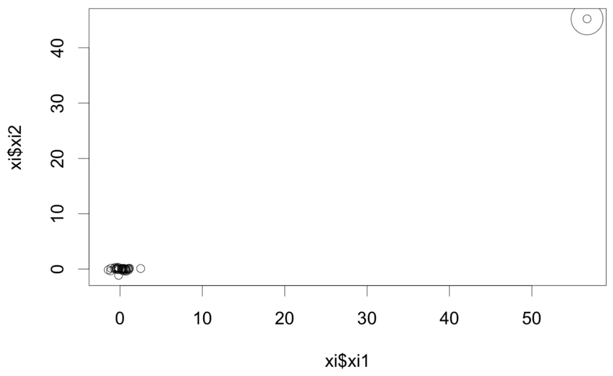


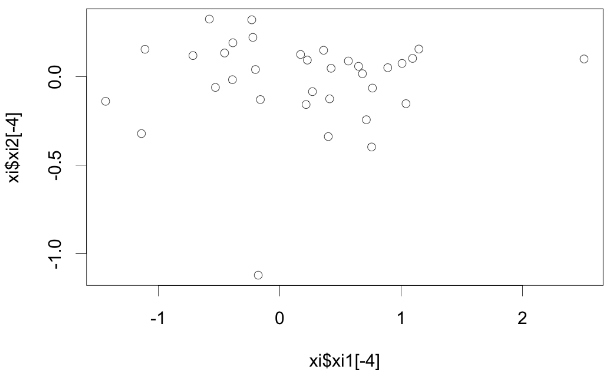
```r

## Covariance estimation
#### Get raw cov
muHat <- predict(resMu)
muObs <- predict(resMu, df$t) # interpolate
df$yCenter <- df$y - muObs
rcov <- plyr::ddply(df, 'id', function(d) {
  raw <- c(tcrossprod(d$yCenter))
  TT <- expand.grid(t1=d$t, t2=d$t)
  cbind(TT, raw=raw)
})

#### fit raw covariance
library(rgl)
plot3d(rcov$t1, rcov$t2, rcov$raw)
#### bandwidth for covariance estimation
hCov <- 0.2
resCov <- locfit(raw ~ lp(t1, t2, deg=1, h=hCov), rcov %>% filter(t1 != t2), ev=lfgrid(mg=ngrid
    ), kern='epan')
covMat <- matrix(predict(resCov), ngrid, ngrid)
persp3d(tGrid, tGrid, covMat, add=FALSE, col='white')
#### find the eigen value and functions of fitted raw covariances
eig <- eigen(covMat)
eig$values
#### remove the negative eigen values
rmInd <- eig$values <= 0
eig$vectors <- eig$vectors[, !rmInd, drop=FALSE]
eig$values <- eig$values[!rmInd]
lam <- eig$values * diff(range(tGrid)) / ngrid
phi <- eig$vectors / sqrt(diff(range(tGrid)) / ngrid)
png("q2pc.png",width = 2200, height = 800, units = "px", pointsize = 30)
matplot(tGrid, phi[, 1:3], type='l',main= "First three PCs of covariance matrix")
dev.off()

#### diagnonal elemenet
diagRes <- locfit(raw ~ lp(t1, deg=1, h=hCov),
                  rcov %>% filter(t1 == t2),  # %>% passes the result to the outside function,
                  ev=lfgrid(mg=ngrid))
Vhat <- predict(diagRes)
lines3d(tGrid, tGrid, Vhat)
sig2 <- mean(Vhat - diag(covMat))
sig2
```

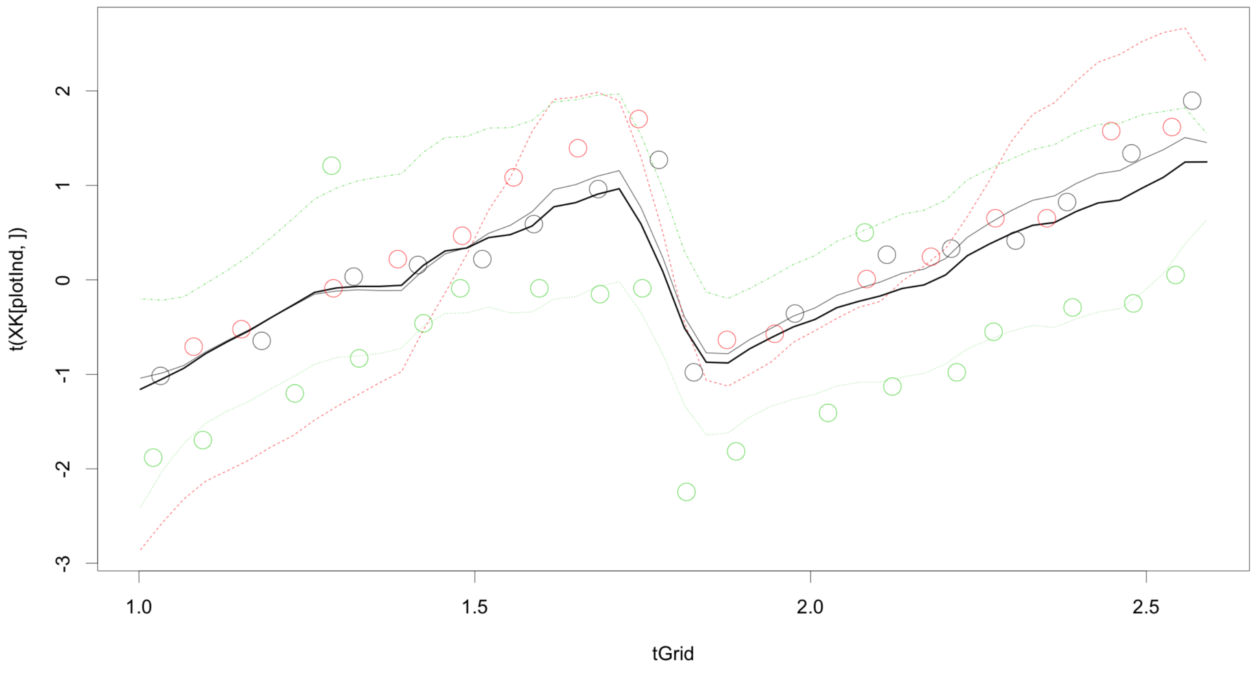




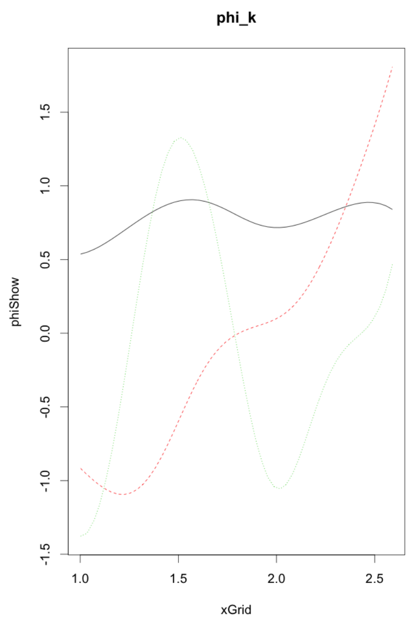
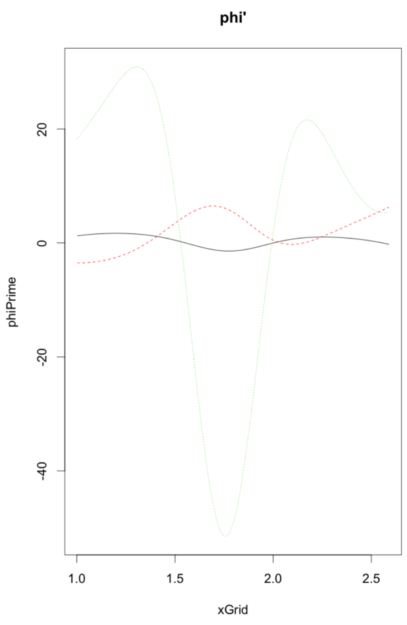
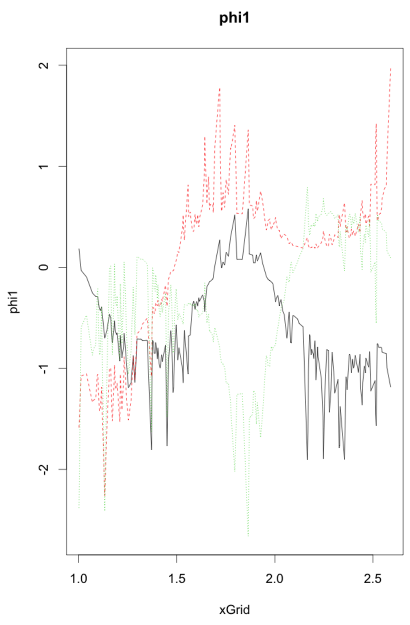
```r
#### BLUP step: xi estimation
K <- 3
LambdaK <- diag(lam[seq_len(K)], nrow=K)
xi <- plyr::ddply(df, 'id', function(d) {
  #browser()
  str(d)
  # Linear interpolation
  Phii <- matrix(fdapace::ConvertSupport(tGrid, d$t, phi=phi[, seq_len(K), drop=FALSE]), ncol=K
      )
  SigYi <- fdapace::ConvertSupport(tGrid, d$t, Cov=covMat) + diag(sig2, nrow=nrow(d))
  ## xi estimation step.
  res <- c(LambdaK %*% t(Phii) %*% solve(SigYi, d$yCenter))
  names(res) <- paste0('xi', seq_len(K))
  res
})
# Note a extreme value at 4-th observation:(id = 55) an outlier
png("q2xi.png",width = 2200, height = 800, units = "px", pointsize = 30)
plot(xi$xi1, xi$xi2)
points(xi$xi1[4], xi$xi2[4], type='p', cex=4)
plot(xi$xi1[-4], xi$xi2[-4])
dev.off()
head(xi)

##### X_K: reconstruct X
XK <- as.matrix(xi[, -1]) %*% t(phi[, seq_len(K), drop=FALSE]) + matrix(muHat, nrow=nrow(xi),
    ncol=ngrid, byrow=TRUE)
plotInd <- c(1:3,35)
png("q2XK.png",width = 2200, height = 800, units = "px", pointsize = 30)
matplot(tGrid, t(XK[plotInd, ]), type='l', col = plotInd)
lines(tGrid, muHat, lwd=3)
points(df$t[df$id==45], df$y[df$id==45], type='p', col=1, cex=2)
points(df$t[df$id==47], df$y[df$id==47], type='p', col=2,cex=2)
points(df$t[df$id==53], df$y[df$id==53], type='p', col=3, cex=2)
points(df$t[df$id==127], df$y[df$id==127], type='p', col=35,cex=2)
dev.off()
```







```
## Derivative of mean function
FPC <- FPCAder(res, list(method='FPC1', bwMu=0.2, bwCov=0.2))
DPC <- FPCAder(res, list(method='DPC', bwMu=0.2, bwCov=0.2))
DPCFVE <- cumsum(DPC$lambdaDer) / sum(DPC$lambdaDer)
xGrid <- DPC$workGrid
plot(xGrid, DPC$mu, type='l', main='mu')
plot(xGrid, DPC$muDer, type='l', main='mu\'')

phiShow <- res$phi[, seq_len(K)]

png("q2muder.png",width = 2000, height = 1000, units = "px", pointsize = 30)
par(mfrow = c(1,3))
matplot(xGrid, phiShow, type='l', main='phi_k')
phiPrime <- FPC$phiDer[, seq_len(K)]
matplot(xGrid, phiPrime, main='phi\'', type='l')
phi1 <- DPC$phiDer[, seq_len(K)]
matplot(xGrid, phi1, main='phi1', type='l')
dev.off()
```


phi_k
phiShow
xGrid
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
1.0
1.5
2.0
2.5

phi'
phiPrime
xGrid
-40
-20
0
20
1.0
1.5
2.0
2.5

phi1
phi1
xGrid
-2
-1
0
1
2
1.0
1.5
2.0
2.5