STAT547 HW6

Zhiling GU

1. Implement the regularization approach for the scalar-response functional linear regression model.

(a) Apply the code to analyze the medfly dataset fdapace::medfly25. Find and interpret the association between the number of `total remaining eggs` as response and the `egg-laying trajectories` as the predictor function. Set the tuning parameter λ appropriately.

```
## Loading required package: rainbow
```

```
## Loading required package: MASS
```

```
## Loading required package: pcaPP
```

```
## Loading required package: RCurl
```

```
## Loading required package: bitops
```

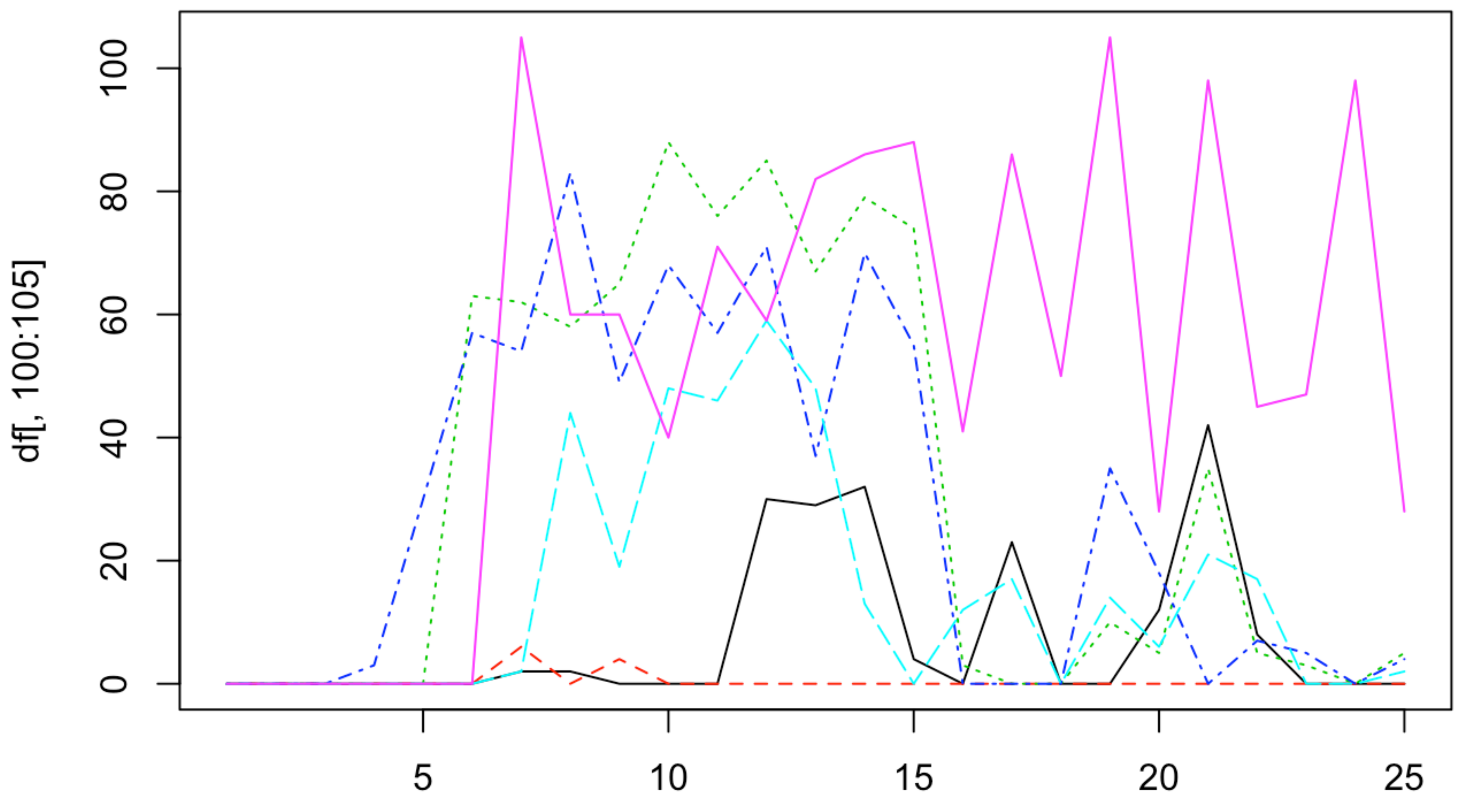
```
## ── Attaching packages ─────────────────── tidyverse 1.3.0 ──
```

```
## ✔ ggplot2 3.2.1     ✔ purrr   0.3.3
## ✔ tibble  2.1.3     ✔ dplyr   0.8.3
## ✔ tidyr   1.0.0     ✔ stringr 1.4.0
## ✔ readr   1.3.1     ✔ forcats 0.4.0
```

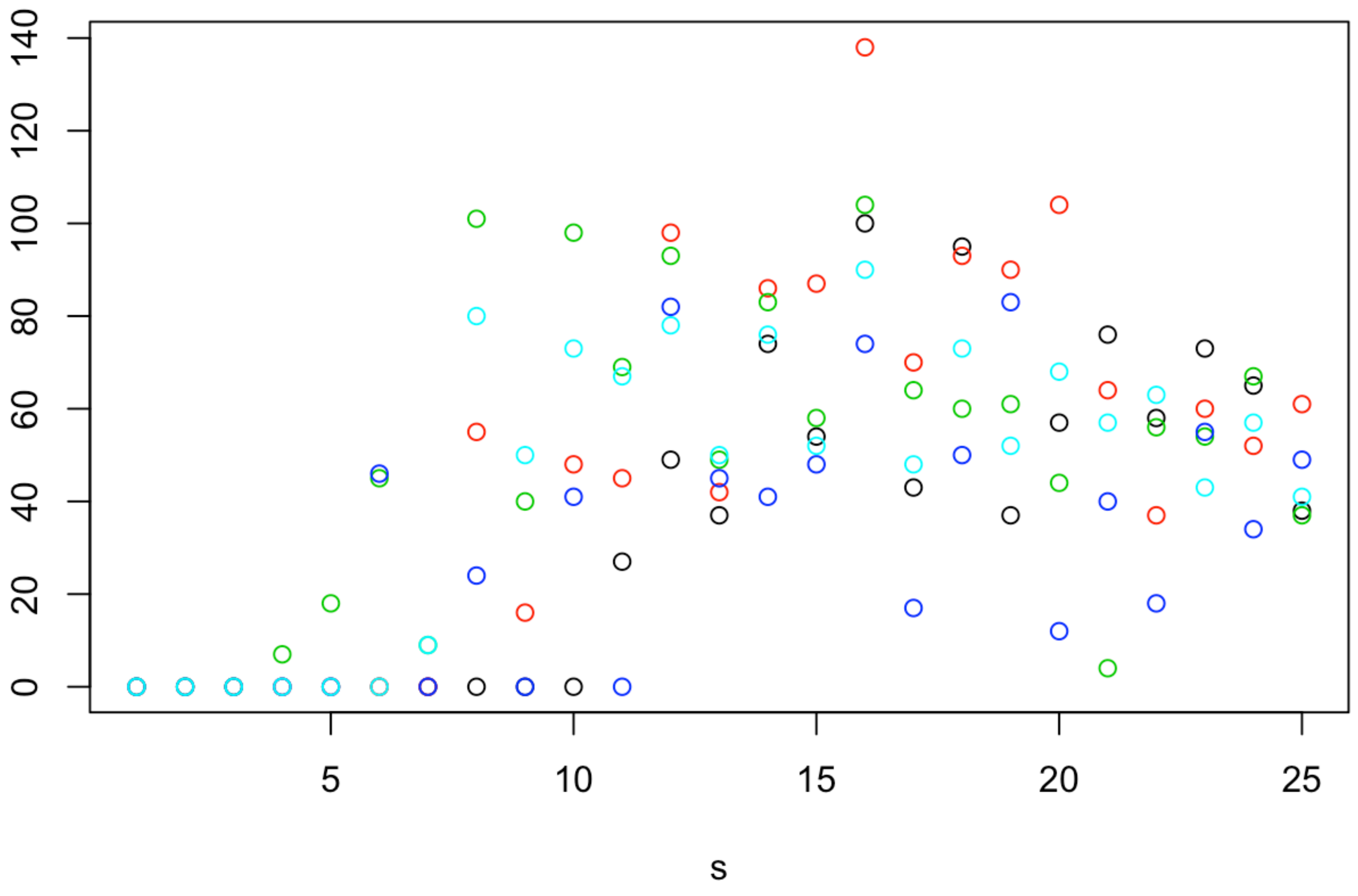
```
## ── Conflicts ────────────────────── tidyverse_conflicts() ──
## ✖ tidyr::complete() masks RCurl::complete()
## ✖ dplyr::filter()   masks stats::filter()
## ✖ dplyr::lag()      masks stats::lag()
## ✖ dplyr::select()   masks MASS::select()
```

```
##
## Attaching package: 'reshape2'
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
#### Preparation
data("medfly25")
dat <- medfly25
denseSamp <- MakeFPCAInputs(IDs=dat$ID, tVec=dat$Days, yVec=dat$nEggs)

## Response: total remaining eggs
y <- dat %>%
  group_by(ID) %>%
  summarize(y=nEggsRemain[1]) %>%
  .$y

## Predictor function: egg-laying trajectories
X <- denseSamp
Y <- y
optnsX <- list()

## Explore the pattern of laid eggs
df <- as.data.frame(X$Ly)
matplot(df[,100:105], type = "l")
```



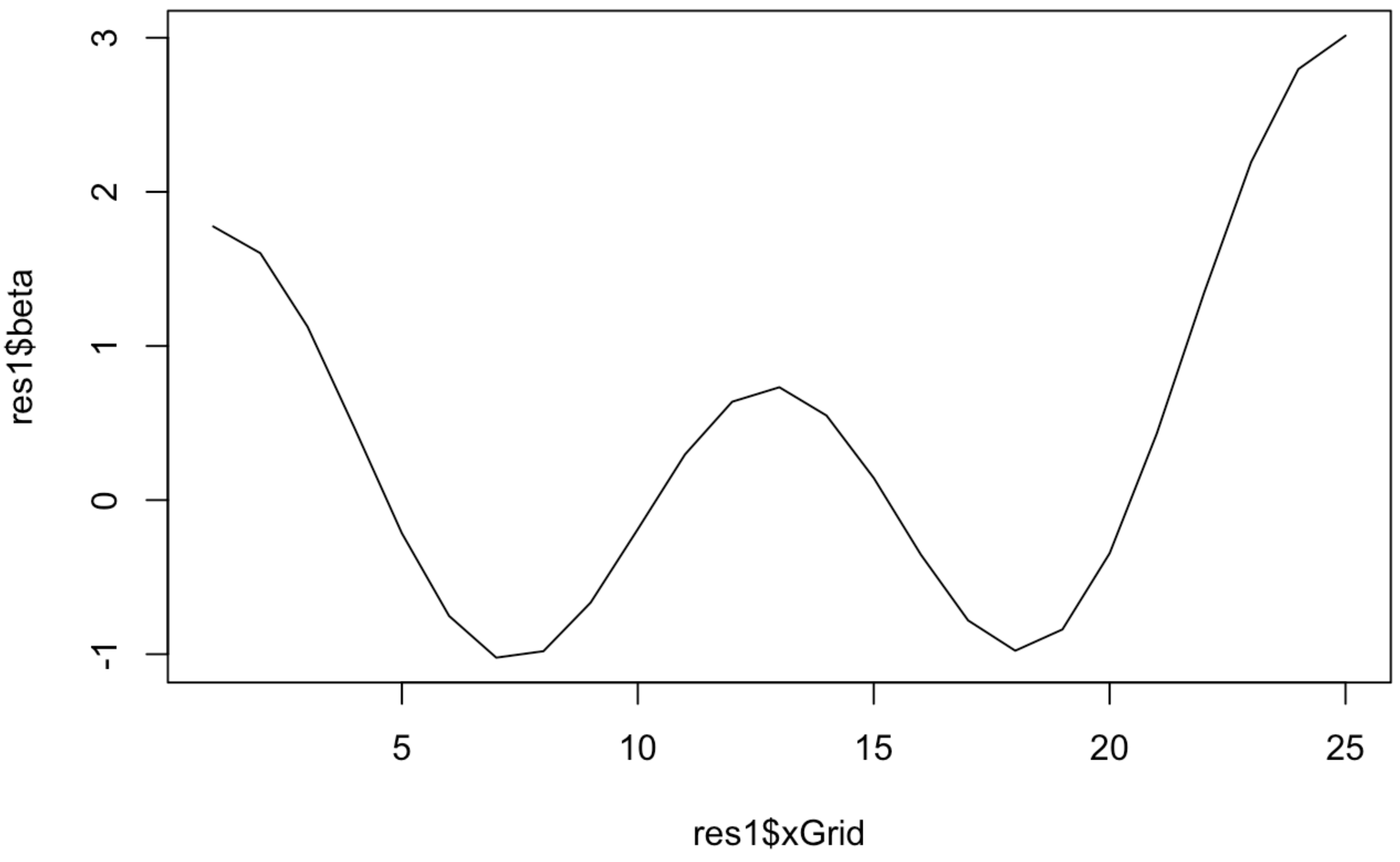
```
CreatePathPlot(inputData = denseSamp, subset=1:5)
```



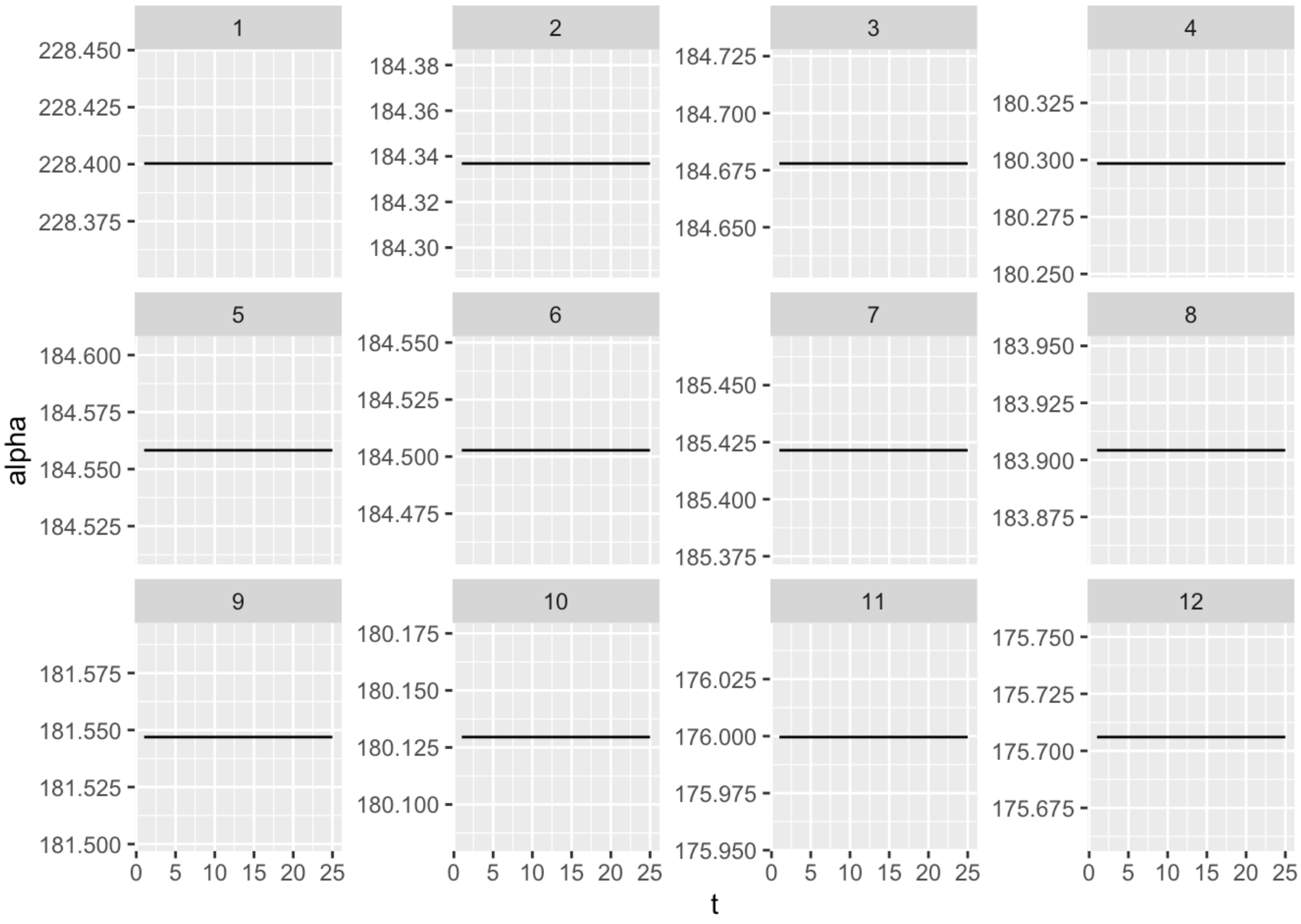
```
## ------------------------------------------------------------------- ##
## Scalar-response functional regression: Regularization approach
## ------------------------------------------------------------------- ##

## Setup: number of basis function and create the base
m <- 0   ## the order of derivative to penalize, usually m = 1  or 2.
t <- denseSamp$Lt[[1]]
dt <- t[-1]- t[-length(t)]
if (length(unique(dt)) !=1) {warning("unequal time step!")} else{dt =  unique(dt)}
optnsX <- list()
lambda <- .1
K = 5
basis <- CreateBasis(K, seq(0,1, length.out = length(t)),type="fourier")
B <- basis

## Regularization method in functional regression
ReguBasisReg <- function(Y, X, XTest, optnsX=list(), lambda = .1, K = 5) {
  basis <- CreateBasis(K, seq(0,1, length.out = length(t)))
  B <- basis
  # Y <- y[trainInd]; X <-trainSamp
  n <- length(Y) ## the number of obs
  ## 1.
  ## Estimate C_i, i = 1, 2, ..., n
  C <- c()
  for (i in 1:n){
      model <- lm( X$Ly[[i]] ~ B )
      C <- rbind(C, unname(model$coefficients[-2])) ## Note the first basis is ones,
the coefficient is the intercept
  }


  J <- matrix(nrow = K, ncol = K)
  for (i in 1:K){
    for (j in i:K){
      J[i,j ] <- sum(B[,i] %*% B[,j] *dt) ## approximation to the integral (\int B(t)
B^T(t) dt)
      J[j,i ] <- J[i,j ]
    }
  }

  Omega <- if (m == 0) {J} else{warning("zero order penalty is applied!"); J}

  desMat <- rbind(c(n, t(rep(1, n)) %*% (C %*% J)), cbind(  t(C %*% J) %*% rep(1,n) ,
t(C %*% J) %*% (C %*% J)+ lambda * Omega ))

  coeff <- solve(desMat, c(n * mean(Y),   t(C %*% J) %*% Y))
  b <- coeff[-1]
  a <- coeff[1]
  betaFun <-  B %*% matrix(b)  ## level of Y explained by linear combination of basis
  alpha <- a ## explained by intercepte
  beta <- b
  residual <- sum (desMat %*% coeff - c(n * mean(Y),   t(C %*% J) %*% Y))^2
  res <- list(
    alpha = alpha,
    beta = betaFun,
    residual = residual,
    xGrid = t,
    C  = C
  )

  ## 2.
  if (!missing(XTest)) {  ## missing(x) checks if arguemnt x is missing
    yPred <- sapply(XTest$Ly, function(z){C.y <- lm(z ~ B)$coeff[-2]; alpha + C.y %*%
J %*% matrix(beta) })
    res <- c(res, list(yPred=yPred))
  }
  res

}

res1 <- ReguBasisReg(y, denseSamp, optnsX = list())
str(res1)
```

```
## List of 5
##  $ alpha   : num 166
##  $ beta    : num [1:25, 1] 1.776 1.602 1.125 0.465 -0.216 ...
##  $ residual: num 1.28e-16
##  $ xGrid   : int [1:25] 1 2 3 4 5 6 7 8 9 10 ...
##  $ C       : num [1:789, 1:5] 35.5 50.9 50 30.8 46.1 ...
```

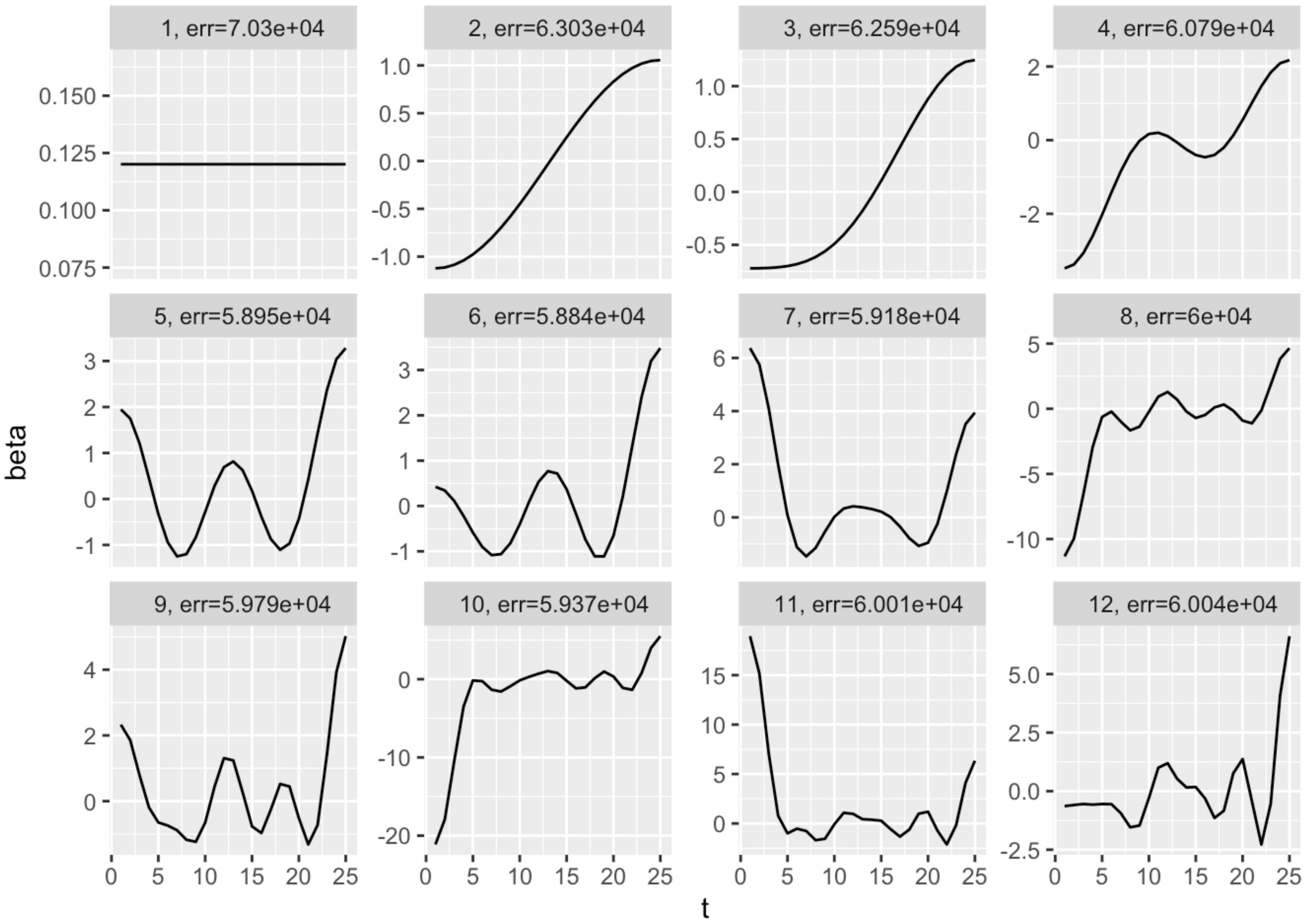
```
plot(res1$xGrid, res1$beta, type='l')  ## only possible when K = length(t)
```



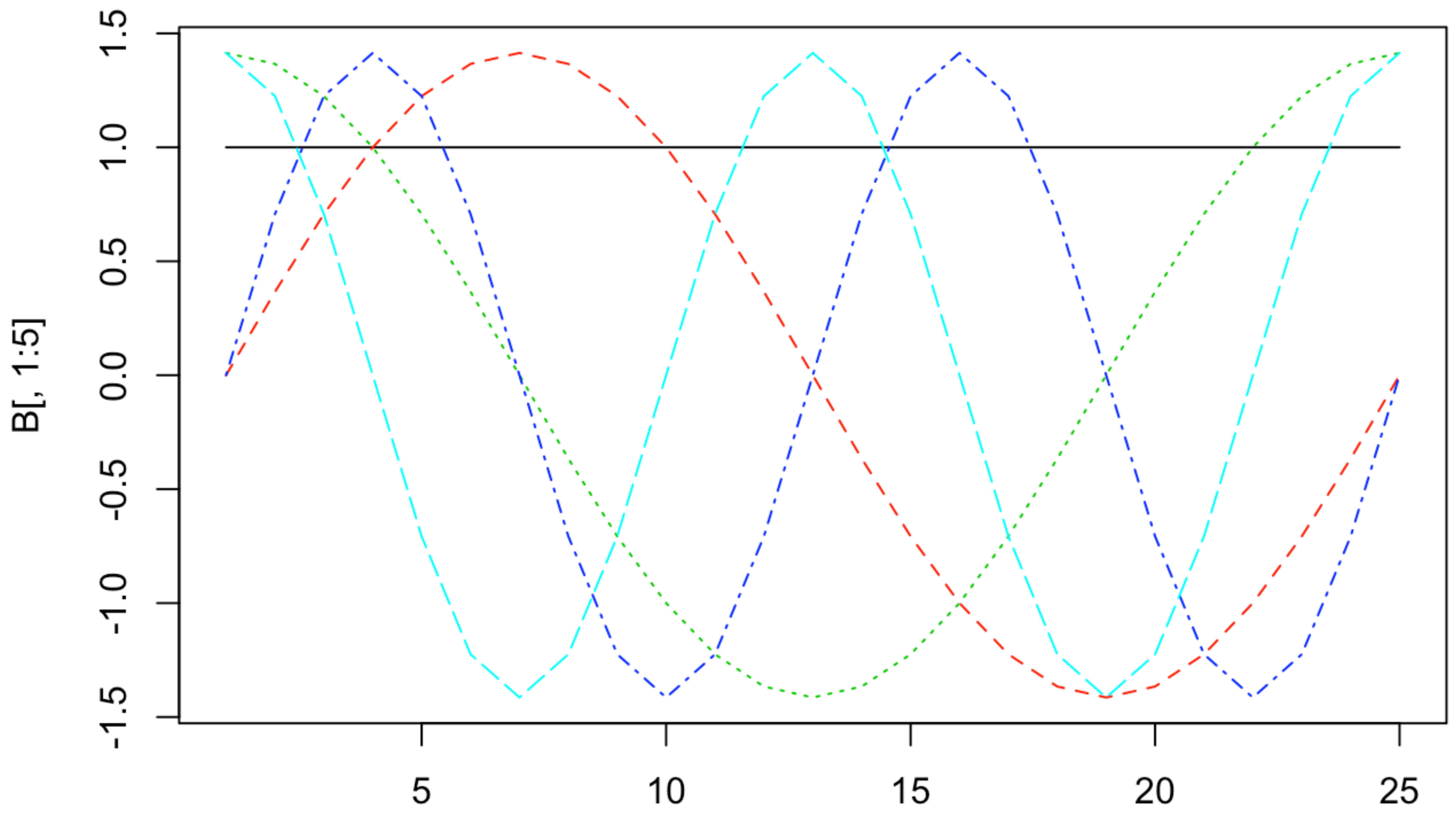
```
## Two-fold CV to select K
n <- length(Y) ## the number of obs
 ## Randomly reselect 1/2 of the data, without replacement
trainInd <- sort(sample(n, floor(n/2)))
testInd <- setdiff(seq_len(n), trainInd)

CV_K <- plyr::ldply(seq(1,12), function(K) {
  # browser()
  trainSamp <- sapply(denseSamp,
                      `[`,  ## ?"[" # act as a function!
                      trainInd,
                      simplify = FALSE)

  testSamp <- sapply(denseSamp,
                     `[`,
                     testInd,
                     simplify = FALSE)

  res <- ReguBasisReg (y[trainInd], trainSamp, XTest=testSamp, K = K)
  err <- mean((res$yPred - y[testInd])^2)
  data.frame(lambda = unname(lambda),
             K=unname(K),
             err=unname(err),   ## unname: Remove the names or dimnames attribute of
an R object.
             t=unname(res$xGrid),
             beta=unname(res$beta),
             alpha=unname(res$alpha))
})

#### Plot the result of functional regression (estimates of alpha and beta)
ggplot(CV_K, aes(x=t, y=alpha)) + geom_line() + facet_wrap(~K, scales='free_y')
```



```
ggplot(CV_K, aes(x=t, y=beta)) + geom_line() + facet_wrap(~sprintf('%4g, err=%.4g', K
, err), scales='free_y')
```

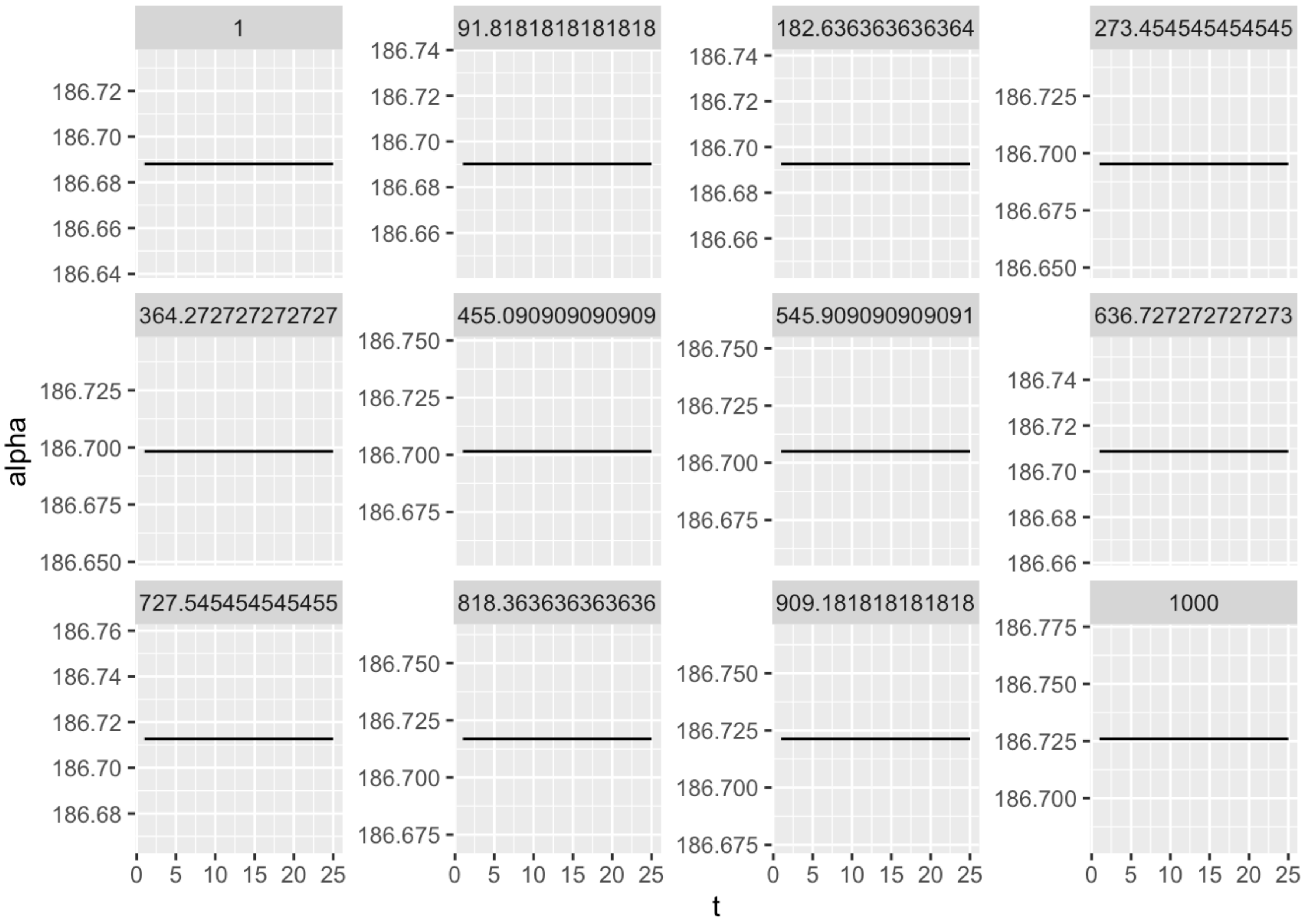


```
matplot(B[,1:5], type='l')
```

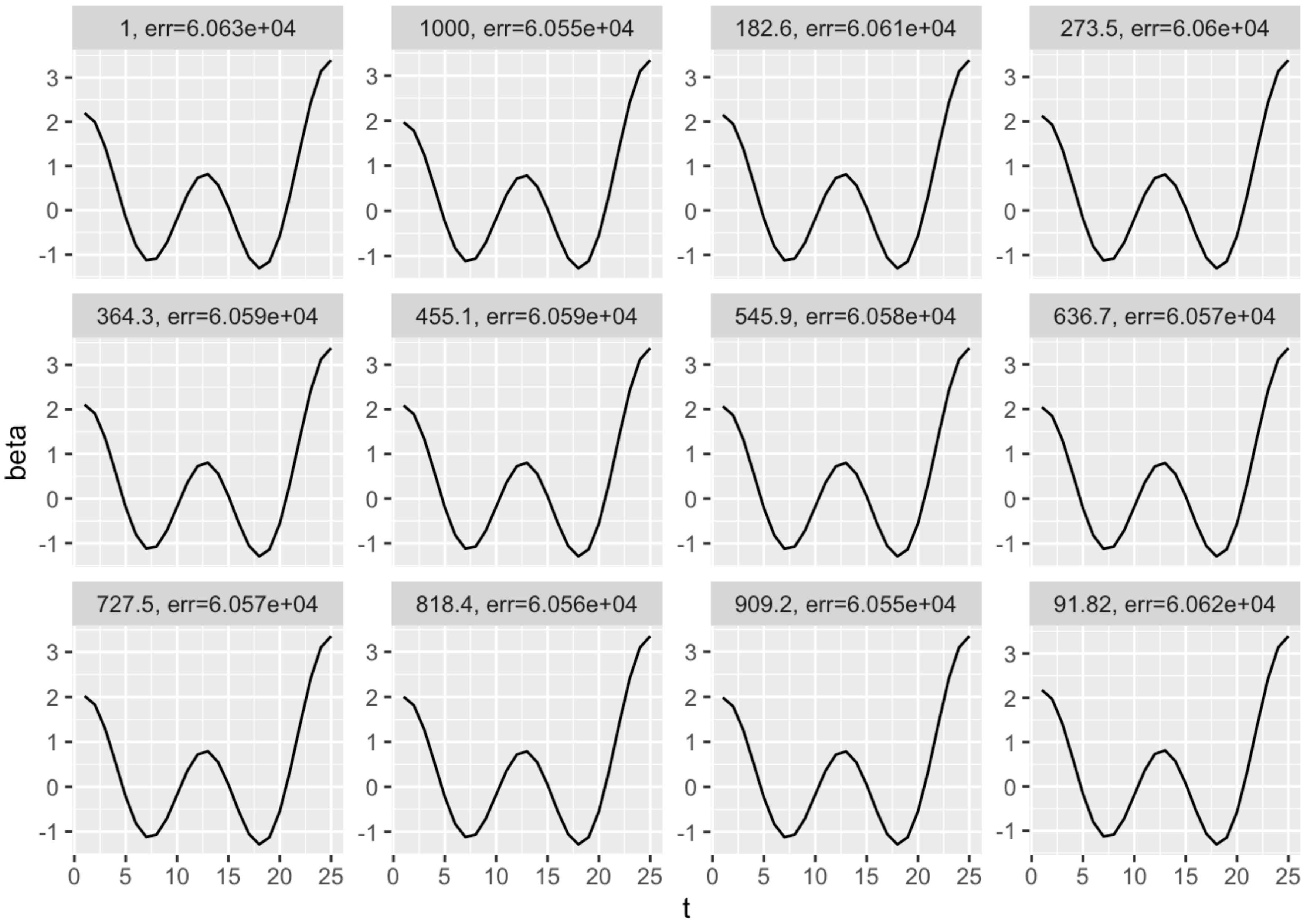


From the graph above we choose the realtively small number of basis with lower error `K = 5`. Then fix `K`, select the penalty parameter `$\lambda$` using two-fold CV again.

```r
## Two-fold CV to choose regularization parameter lambda
n <- length(Y) ## the number of obs
 ## Randomly reselect 1/2 of the data, without replacement
trainInd <- sort(sample(n, floor(n/2)))
testInd <- setdiff(seq_len(n), trainInd)

trainSamp <- sapply(denseSamp,
                    `[`,  ## ?"[" # act as a function!
                    trainInd,
                    simplify = FALSE)

testSamp <- sapply(denseSamp,
                   `[`,
                   testInd,
                   simplify = FALSE)
CV_lam <- plyr::ldply(seq(1,1000, length.out = 12), function(lambda) {
  # browser()


  res <- ReguBasisReg (y[trainInd], trainSamp, XTest=testSamp, K = 5,  lambda = lambd
a)
  err <- mean((res$yPred - y[testInd])^2)
  data.frame(lambda = unname(lambda),
             K=unname(K),
             err=unname(err),   ## unname: Remove the names or dimnames attribute of
an R object.
             t=unname(res$xGrid),
             beta=unname(res$beta),
             alpha=unname(res$alpha))
})

#### Plot the result of functional regression (estimates of alpha and beta)
ggplot(CV_lam, aes(x=t, y=alpha)) + geom_line() + facet_wrap(~lambda, scales='free_y'
)
```



```
ggplot(CV_lam, aes(x=t, y=beta)) + geom_line() + facet_wrap(~sprintf('%.4g, err=%.4g'
, lambda, err), scales='free_y')
```

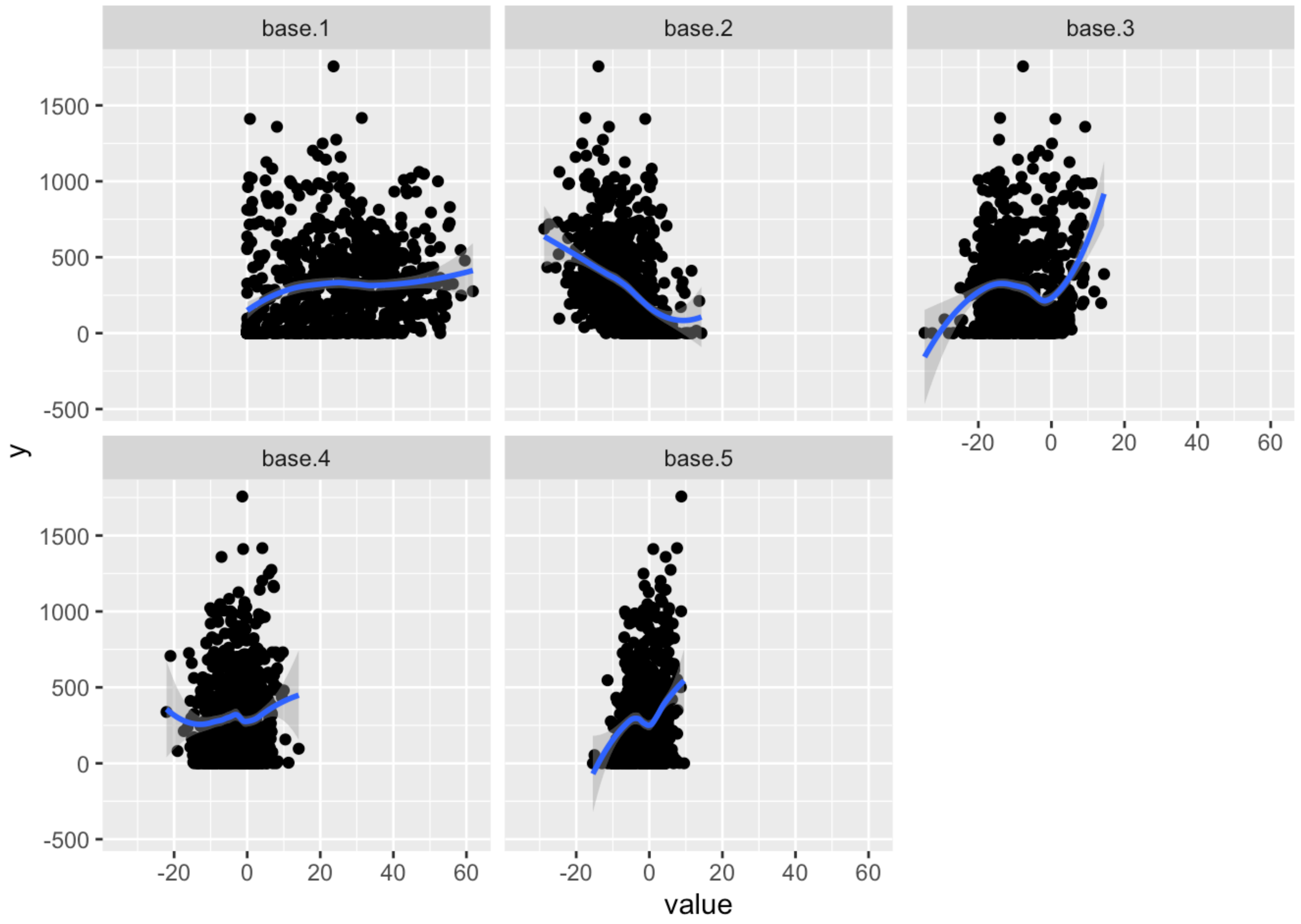


```
res1 <- ReguBasisReg (Y, X, K = 5,  lambda = 100)
plotDat <- data.frame(base=res1$C,
                      y=Y) %>%
            melt(id.var='y')
str(plotDat)
```

```
## 'data.frame':    3945 obs. of  3 variables:
##  $ y       : int  717 434 334 809 329 344 158 147 189 223 ...
##  $ variable: Factor w/ 5 levels "base.1","base.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ value   : num  35.5 50.9 50 30.8 46.1 ...
```

```
ggplot(plotDat, aes(x=value, y=y)) + geom_point() + geom_smooth() + facet_wrap(~varia
ble)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



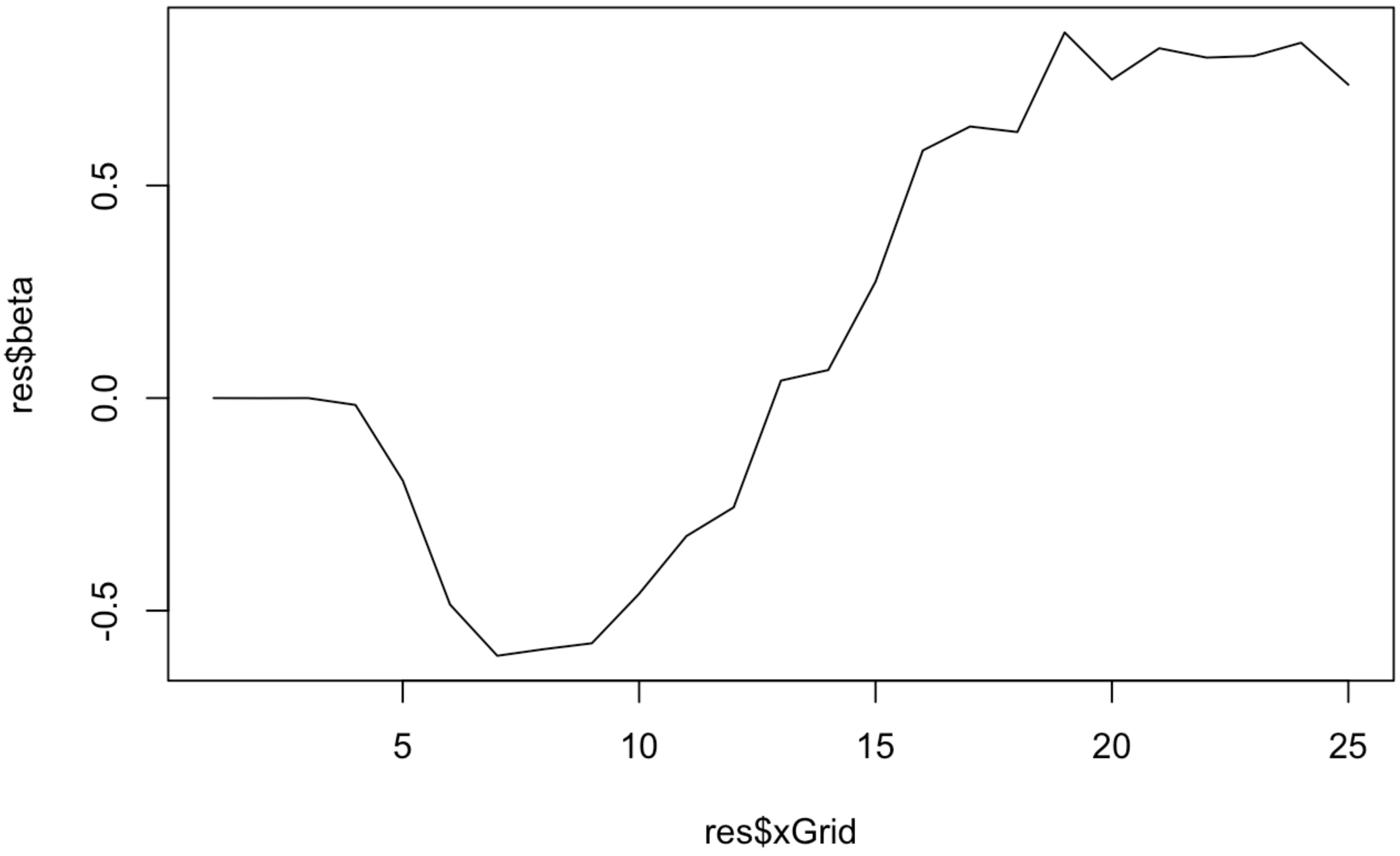
Note the change in number of `$\lambda$` does not affect the error significantly in estimation when $K = 5$. In fact, `$\lambda$` is important only when K is large, which makes sense, since it is a penalty parameter for model complexity. For $K = 5$, we make a rough choice $\lambda = 100$.

(b) Compare your results with those given by an FPC regression approach (code available in the Notebook §4.4).

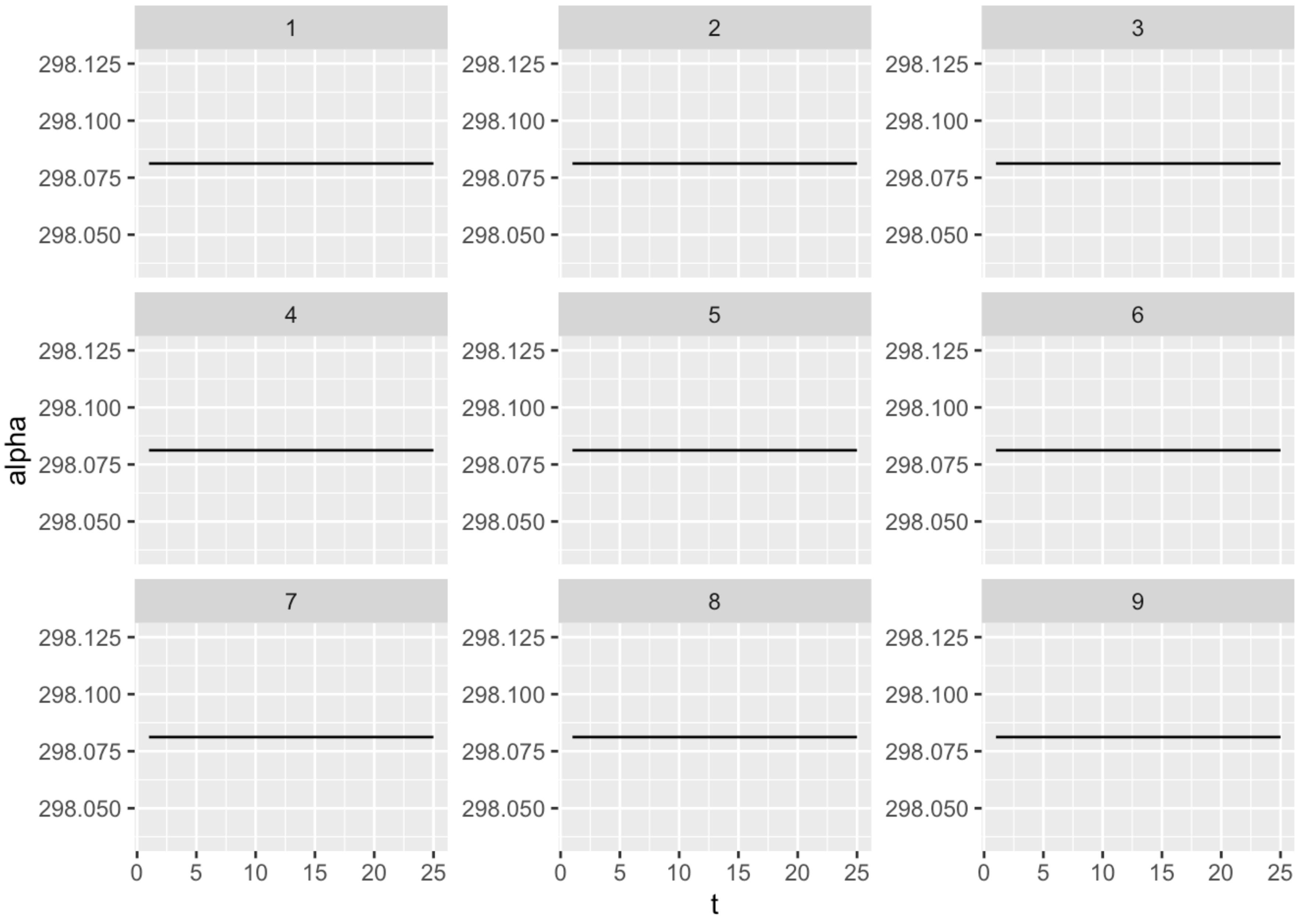
```
## ---------------------------------------------------------------------- ##
## Scalar-response functional regression: FPC regression approach
## ---------------------------------------------------------------------- ##

#### Customized FPCA regression: if XTest exists, result include the prediction at XT
est
fpcReg <- function(Y, X, XTest, optnsX=list()) {
  ## 1.
  ## First estimate fpcs
  fpcaRes <- FPCA(X$Ly, X$Lt, optnsX)
  xiEst <- fpcaRes$xiEst
  dat <- data.frame(y=Y, x=xiEst)
  model <- lm(y ~ ., dat)  ## Note the usage of "."
  b <- model$coefficients[-1]
  a <- model$coefficients[1]
  betaFun <- fpcaRes$phi %*% matrix(b)
  alpha <- a
  res <- list(
    alpha = alpha,
    beta = betaFun,
    xGrid = fpcaRes$workGrid,
    fpcaRes = fpcaRes
  )
  ## 2.
  if (!missing(XTest)) {  ## missing(x) checks if arguemnt x is missing
    xiTest <- predict(fpcaRes, XTest$Ly, XTest$Lt,
                      xiMethod=fpcaRes$optns$methodXi,
                      K=fpcaRes$selectK)
    yPred <- predict(model, setNames(data.frame(xiTest), names(dat)[-1]))
    res <- c(res, list(yPred=yPred))
  }
  res
}

res <- fpcReg(y, denseSamp, optnsX = list(methodSelectK=3))
str(res)
```

```
## List of 4
##  $ alpha  : Named num 295
##   ..- attr(*, "names")= chr "(Intercept)"
##  $ beta   : num [1:25, 1] 0.00 -2.76e-04 -9.08e-17 -1.61e-02 -1.94e-01 ...
##  $ xGrid  : num [1:25] 1 2 3 4 5 6 7 8 9 10 ...
##  $ fpcaRes:List of 19
##   ..- attr(*, "class")= chr "FPCA"
```

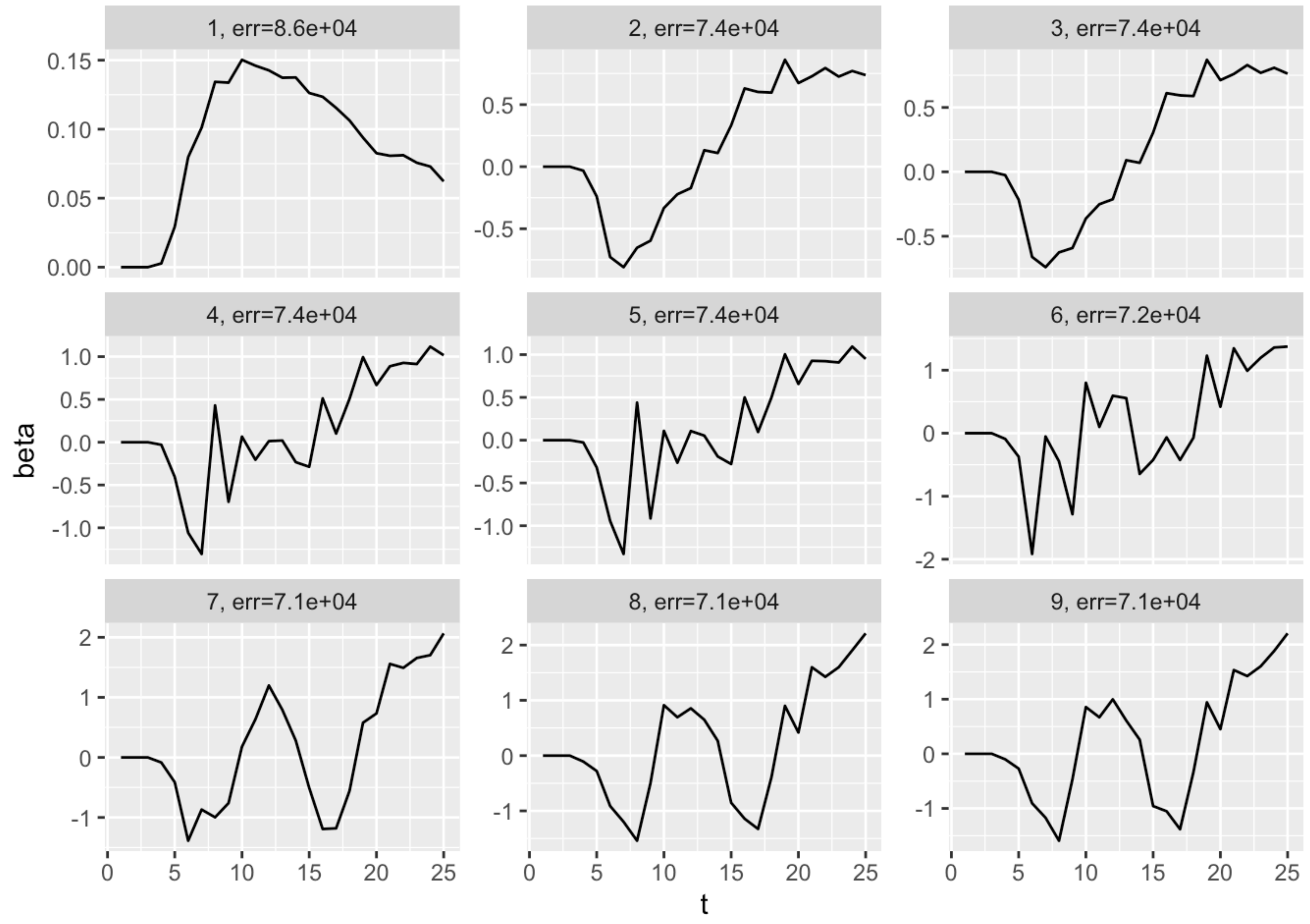
```
plot(res$xGrid, res$beta, type='l')
```


res$beta
0.5
0.0
-0.5
5
10
15
20
25
res$xGrid

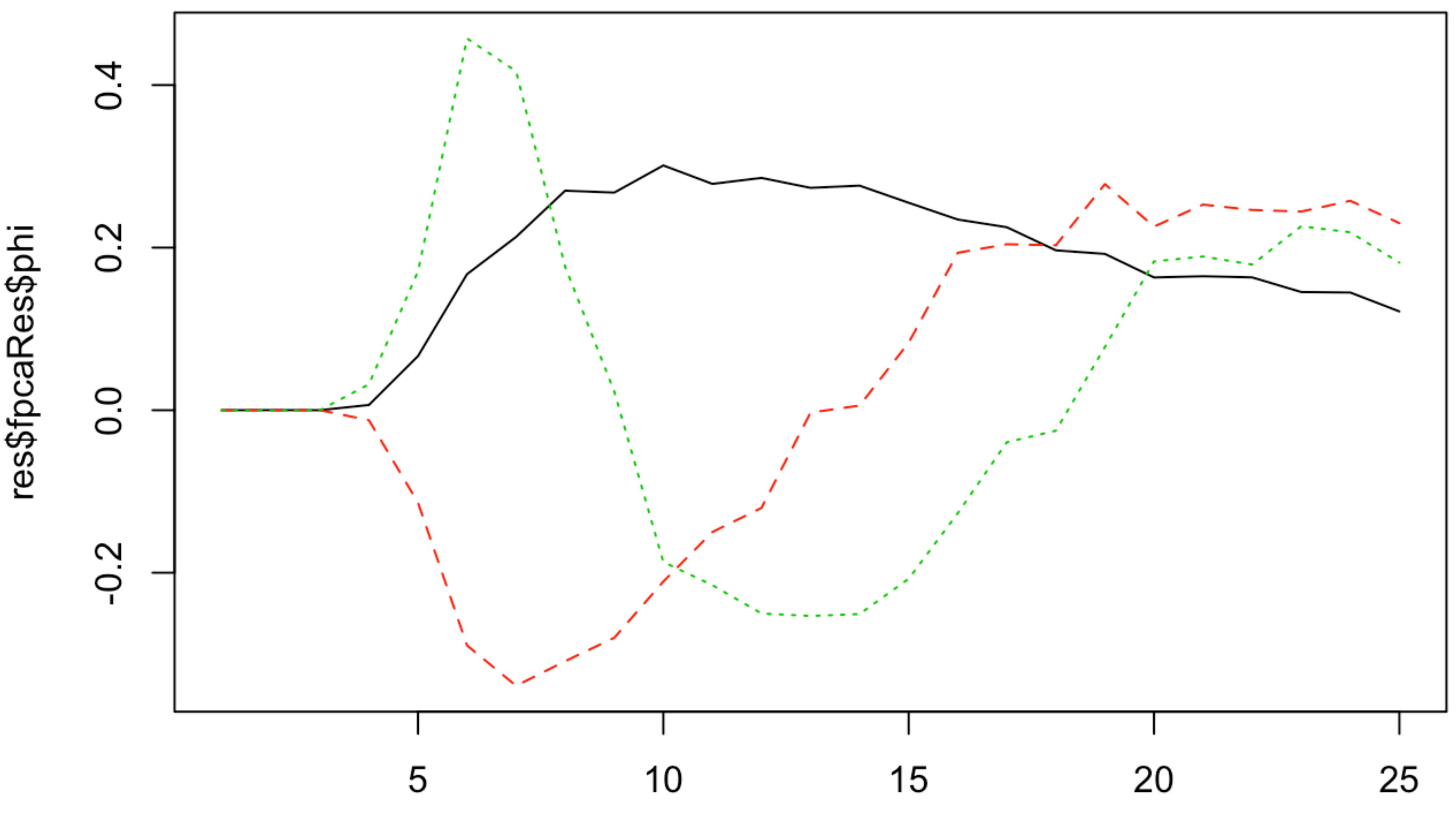
```
## Two-fold CV
n <- length(y)
trainInd <- sort(sample(n, floor(n/2)))  ## Randomly reselect 1/2 of the data, withou
t replacement
testInd <- setdiff(seq_len(n), trainInd)

MPE <- plyr::ldply(seq(1, 9), function(K) {
  # browser()
  trainSamp <- sapply(denseSamp,
                      `[`,  ## ?"[" # act as a function!
                      trainInd,
                      simplify = FALSE)

  testSamp <- sapply(denseSamp,
                     `[`,
                     testInd,
                     simplify = FALSE)

  res <- fpcReg(y[trainInd], trainSamp, XTest=testSamp, optnsX = list(methodSelectK=K
))
  err <- mean((res$yPred - y[testInd])^2)
  data.frame(K=unname(K),
             err=unname(err),   ## unname: Remove the names or dimnames attribute of
an R object.
             t=unname(res$xGrid),
             beta=unname(res$beta),
             alpha=unname(res$alpha))
})

#### Plot the result of functional regression (estimates of alpha and beta)
ggplot(MPE, aes(x=t, y=alpha)) + geom_line() + facet_wrap(~K, scales='free_y')
```



```
ggplot(MPE, aes(x=t, y=beta)) + geom_line() + facet_wrap(~sprintf('%d, err=%.2g', K,
err), scales='free_y')
```



```
matplot(res$fpcaRes$phi, type='l')
```

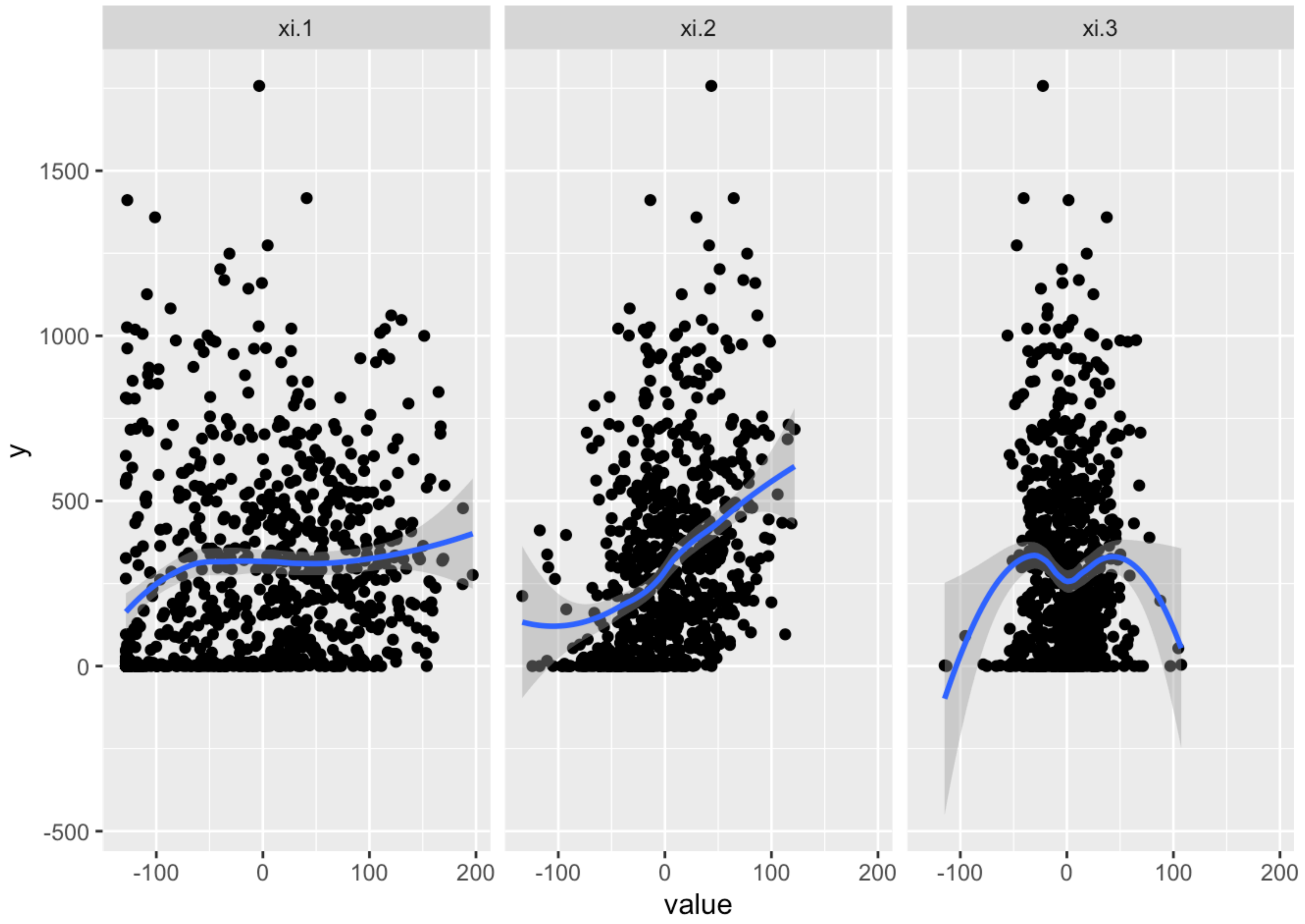


```
plotDat <- data.frame(xi=res$fpcaRes$xiEst,
                      y=y) %>%
           melt(id.var='y')
str(plotDat)
```

```
## 'data.frame':    2367 obs. of  3 variables:
##  $ y       : int  717 434 334 809 329 344 158 147 189 223 ...
##  $ variable: Factor w/ 3 levels "xi.1","xi.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ value   : num  49.9 140.3 145.7 31.6 123.3 ...
```

```
ggplot(plotDat, aes(x=value, y=y)) + geom_point() + geom_smooth() + facet_wrap(~varia
ble)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Note that the error is pretty close between two methods, around $6.4e + 04$.

2. Concurrent Regression Model

Use the functional concurrent regression model to analyze the US GDP data posted with this homework. An R implementation fdapace::FCReg is available. Explore different tuning parameter settings and interpret the results.

Recall functional concurrent regression model: fiven a sample of pared functions $(X_i(t), Y_i(t)), t = 1, \ldots, n$ where $Y_i(t)$ only depends on the random variable $X_i(t)$ valued at the current time.

$$Y_i(t) = \alpha(t) + X_i(t)^T \beta(t) + \varepsilon_i(t)$$

In our case, let $Y_i(t)$ denote the logarithm of perCapitalGDP for i-th state at time t. Let $X_{i1}(t) $ denote logarithm of percentage of labor out of total population, $X_{i2}(t) $ denote logarithm of total population for i-th state at time t. Here we take the logarithm to match the level of each variable to achieve more accurate estimates.

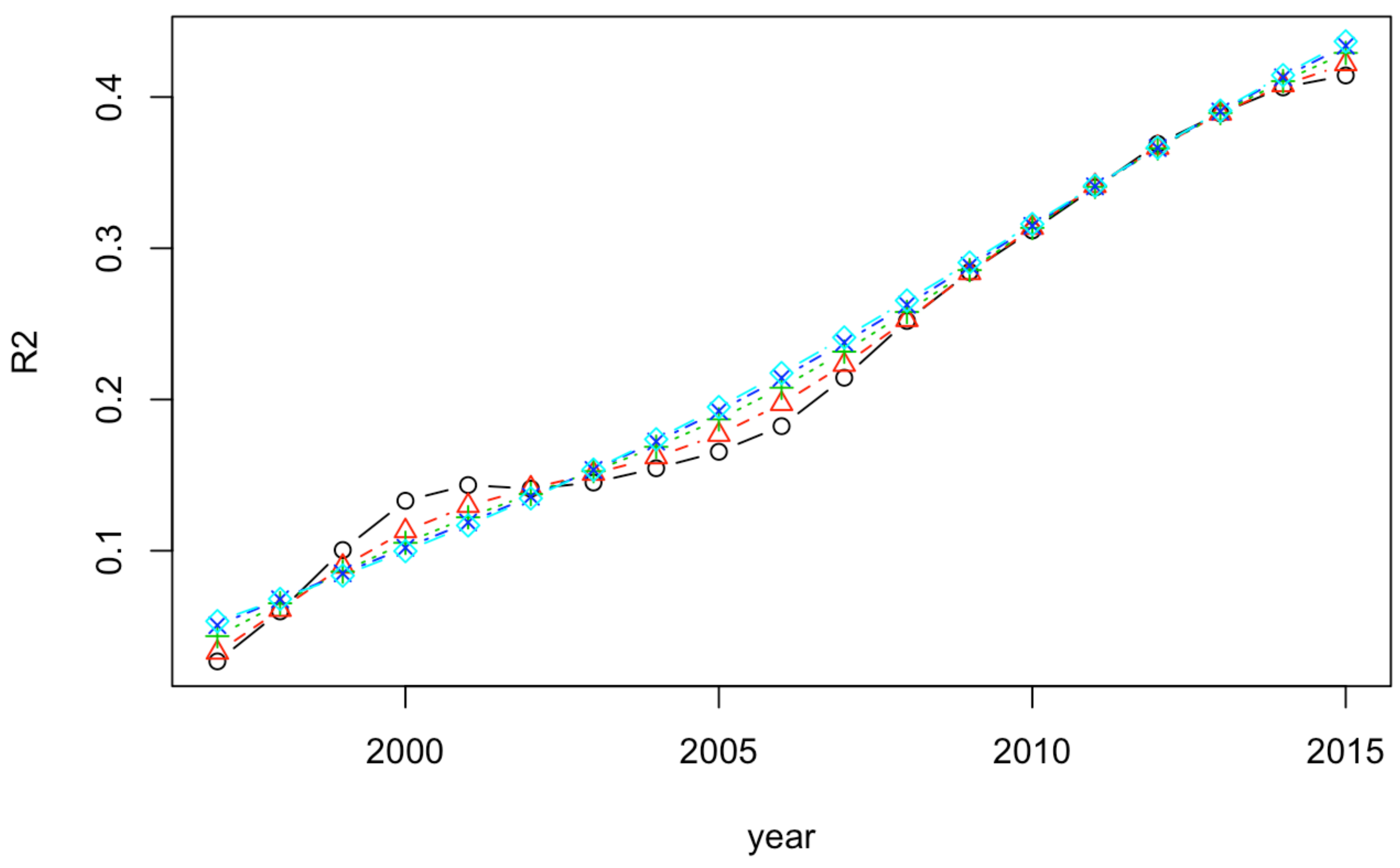
We choose same bandwidth for mean and covariance functions estimation, and change them together to fund the influence of bandwidth on the estimation. The betas estimaed becomes smooth throughout time as we increase the bandwidth, of course. From R2 plot, we see increase in bandwidth helps to improve the overall model fitting performance. This can be explained by higher bandwidth decreases the affects of outliers.

From $\beta_0, \beta_1, \beta_2$ plots, we see trends of each estimates throughout the years. The overall level of GDPperCapita (β_0) increases which may result from higher productivity due to technology. The log percentage of labor force and log GDPperCapita becomes more positively correlated (β_1) throughout the years. The log total population and log GDPperCapita exibits varied correlattion (β_2) throughout the years. We may conclude that the effect of labor force structure (percentage of labor force in the total population) plays a more important role in perCapitaGDP.

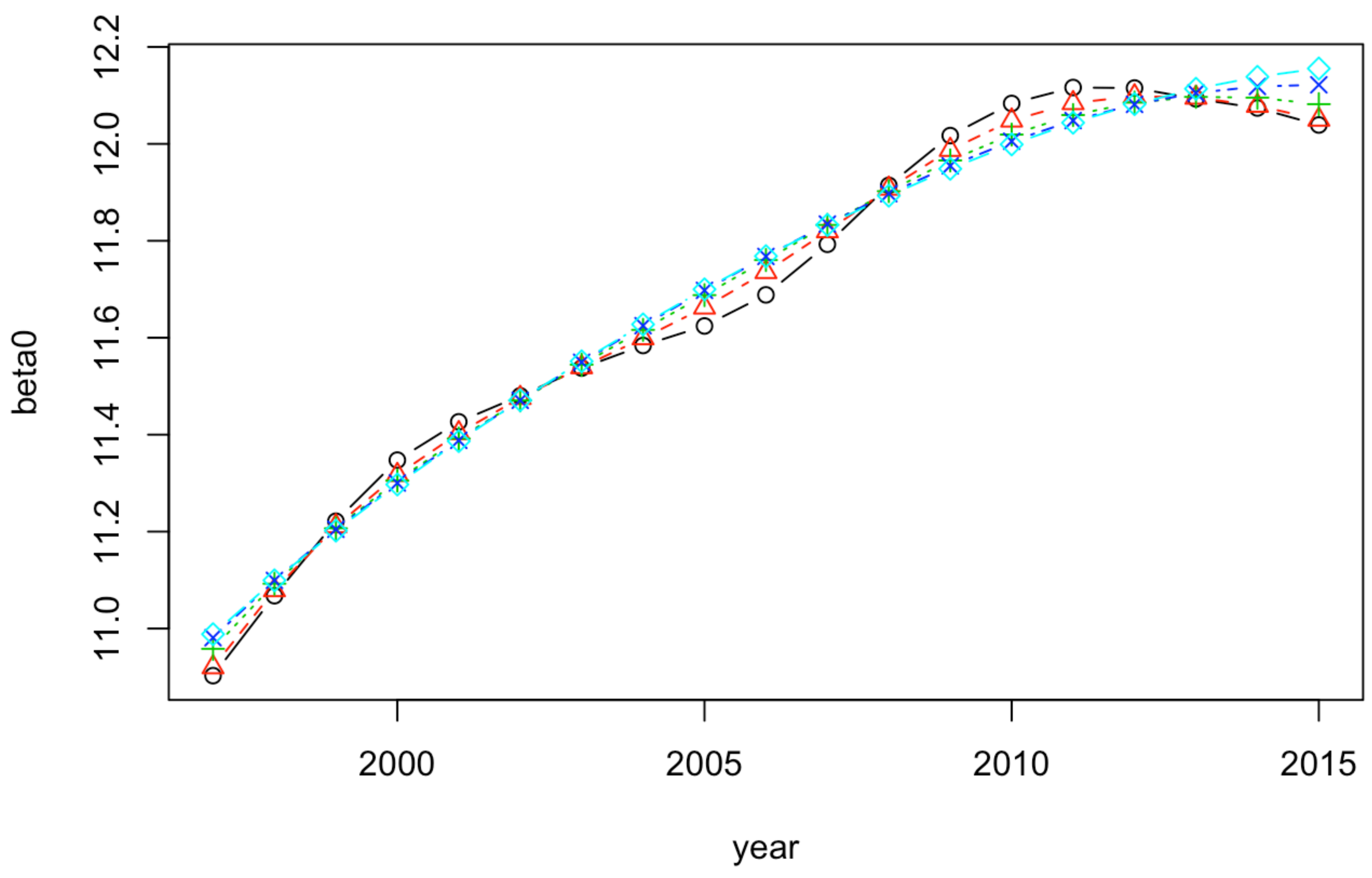
```
dat <- read.csv("USGDP.csv")
head(dat)
```

```
##        States    t totalPopulation totalLaborForce perCapitaGDP
## 1     Alabama 1997         4367935         2133573        31465
## 2     Arizona 1997         4736990         2308569        34485
## 3    Arkansas 1997         2601090         1242594        30470
## 4  California 1997        32486010        15790063        41314
## 5    Colorado 1997         4018293         2217553        43632
## 6 Connecticut 1997         3349348         1752540        54783
```

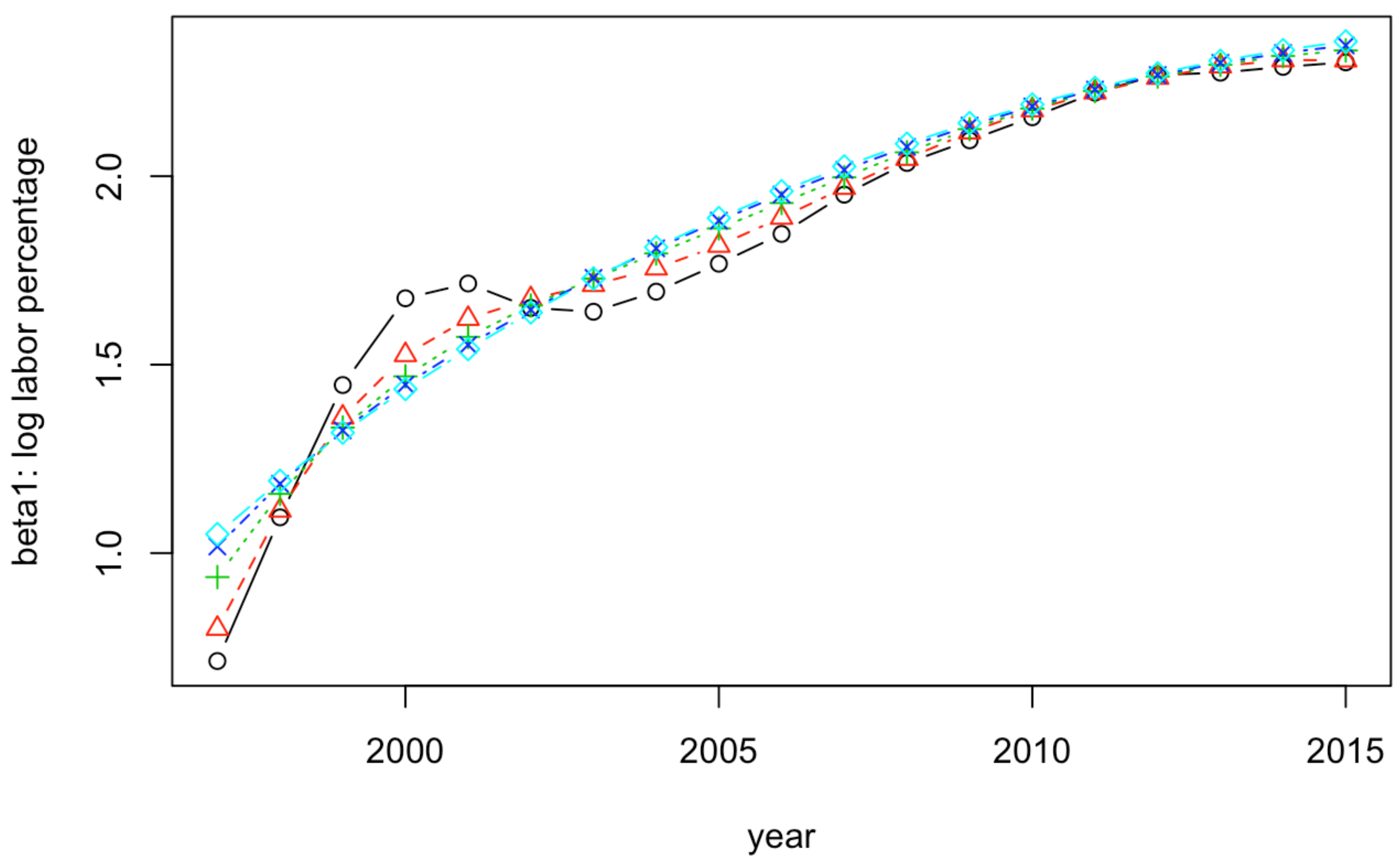
```
## Bandwidth for covariance and mean function estimation
bw <- .1
## Get the desired functional data structure
## Transform the data to alleviate collinearity problem
x1 <- log(dat$totalLaborForce)  - log(dat$totalPopulation)
x2 <- log(dat$totalPopulation )
y  <- log(dat$perCapitaGDP)
X_1 <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=x1)
X_2 <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=x2)
Y <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=y )
vars <- list(logLaborPerc = X_1, logtotalPop = X_2, logGDP = Y)
fcReg2D <- FCReg(vars = vars,
userBwMu = bw, userBwCov = bw, outGrid = unique(dat$t))



bw.vec <- 1:5
R2.mat <- c()
beta1.mat <- c()
beta2.mat <- c()
beta0.mat <- c()
for (bw in bw.vec){
  fcReg2D <- FCReg(vars = vars,
userBwMu = bw, userBwCov = bw, outGrid = unique(dat$t))
  R2.mat <- rbind(R2.mat, fcReg2D$R2)
  beta0.mat <- rbind(beta0.mat, fcReg2D$beta0)
  beta1.mat <- rbind(beta1.mat, fcReg2D$beta[1,])
  beta2.mat <- rbind(beta2.mat, fcReg2D$beta[2,])
}
## rows: bandwidth; columns: different years
matplot(unique(dat$t), t(R2.mat), type = "b", xlab = "year", ylab = "R2", pch=seq(len
gth(bw.vec)), col=seq(length(bw.vec)))
```



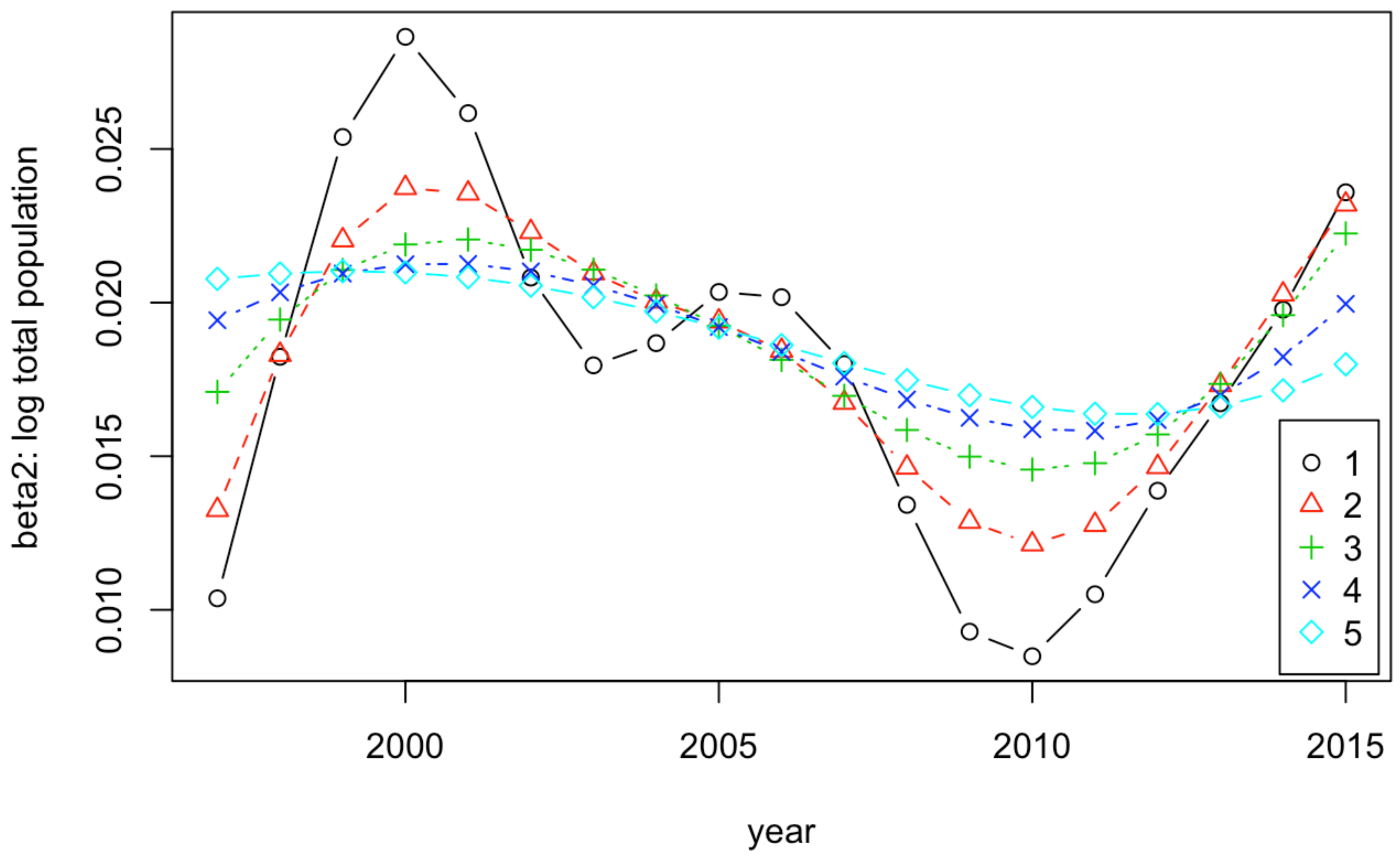
```
matplot(unique(dat$t), t(beta0.mat), type = "b", xlab = "year",ylab = "beta0", pch=se
q(length(bw.vec)), col=seq(length(bw.vec)))
```



```
matplot(unique(dat$t), t(beta1.mat), type = "b", xlab = "year",ylab = "beta1: log lab
or percentage", pch=seq(length(bw.vec)), col=seq(length(bw.vec)))
```



```
matplot(unique(dat$t), t(beta2.mat), type = "b", xlab = "year",ylab = "beta2: log tot
al population", pch=seq(length(bw.vec)), col=seq(length(bw.vec)))
legend("bottomright", inset=0.01, legend=bw.vec, col=seq(length(bw.vec)),pch=seq(leng
th(bw.vec)),
bg= ("white"), horiz=F)
```



3. Functional-response functional linear regression model

Implement the FPC regression approach for a functional-response functional linear regression model and apply it to analyze the US GDP data. Compare the results and interpretation with those in the previous question.

Recall the functional-response functional linear regression: given $\{X_i(s), Y_i(s), s \in T_1, t \in T_2\}$ be iid realizations of a pair of stochastic processes.

$$Y_i(t) = \alpha(t) + \int_{T_1} \beta(t,s)^T X_i(s)ds + \epsilon_i(t) = \alpha(t) + \int_{T_1} \beta_1(t,s)X_{i1}(s)ds + \int_{T_1} \beta_2(t,s)X_{i2}(s)ds + \epsilon_i(t), t \in T_2$$

where $i = 1, \ldots, n,\ \epsilon_i(t)$ is a zero-mean L2 stochastic process independent from X_i.

$T_1 = T_2 = \{1997, \ldots, 2015\}$

```r
## Make use of FLM
x1 <- log(dat$totalLaborForce)  - log(dat$totalPopulation)
x1.2 <- log(dat$totalLaborForce)
x2 <- log(dat$totalPopulation )
y  <- log(dat$perCapitaGDP)

X_1 <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=x1)
X_1.2 <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=x1.2)
X_2 <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=x2)
Y <-  MakeFPCAInputs(IDs=dat$States, tVec=dat$t, yVec=y )

t <- unique(dat$t)
vars <- list(logLaborPerc = X_1, logtotalPop = X_2, logGDP = Y)
denseFLM <- FLM(Y = Y, X = list(logLaborPerc = X_1, logtotalPop = X_2) )
denseFLM2 <- FLM(Y = Y, X = list(logLabor =X_1.2, logtotalPop = X_2) )
# denseFLM3 <- FLM(Y = Y, X = list(logLaborPerc = X_1) )

# install.packages('htmlwidgets')
# install.packages("plotly")
library(plotly)
```

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

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

```
## The following object is masked from 'package:MASS':
##
##     select
```

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

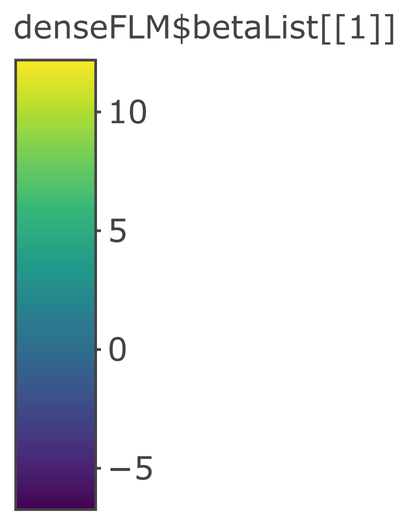
```
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
library('htmlwidgets')
p <- plot_ly(z = ~denseFLM$betaList[[1]], x=~denseFLM$workGridX[[1]] ,  y=~denseFLM$w
orkGridX[[1]], ylab = "s",  zlab ="beta1", theta = -30, phi = 30) %>% add_surface()
```

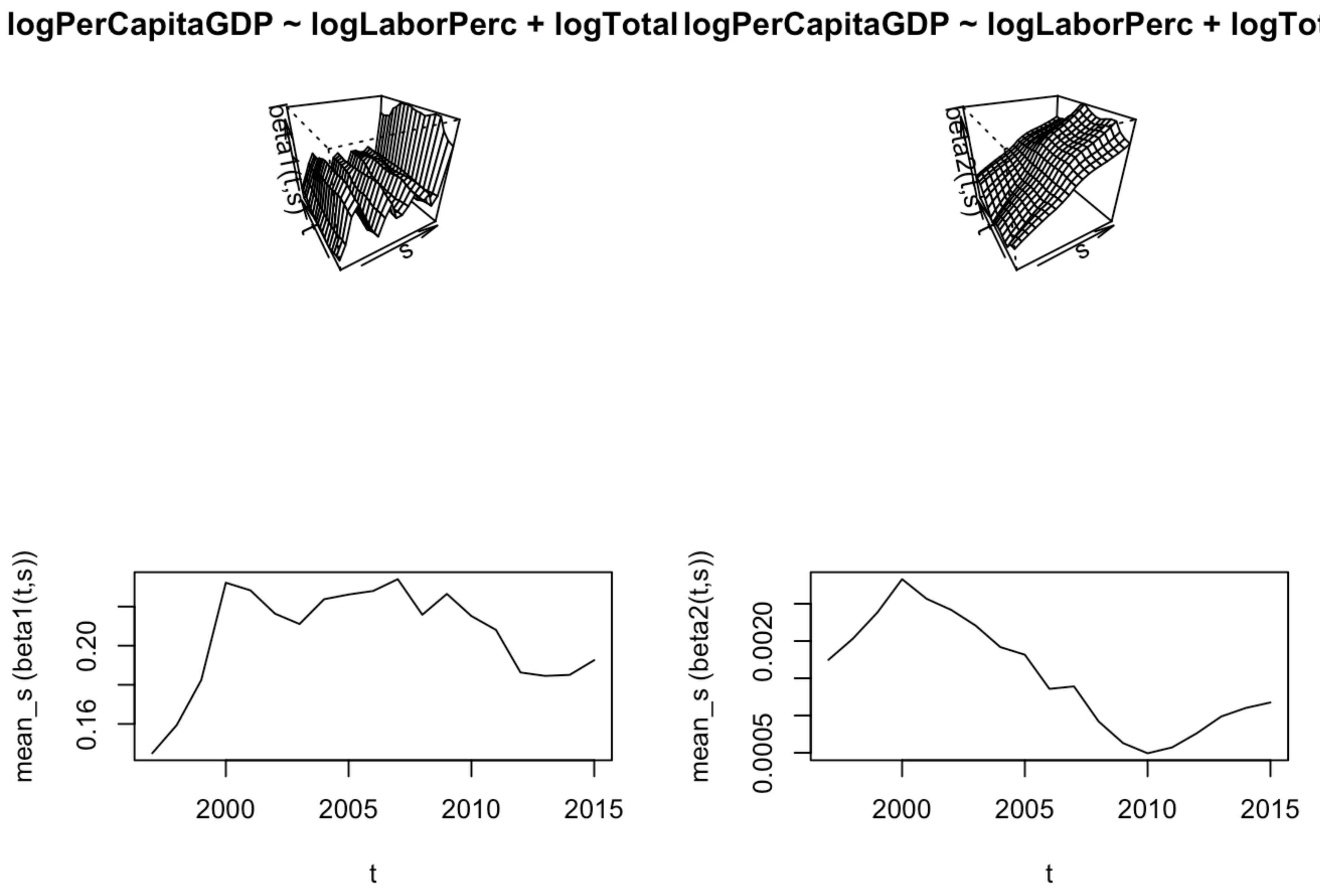
```
## This version of Shiny is designed to work with 'htmlwidgets' >= 1.5.
##     Please upgrade via install.packages('htmlwidgets').
```

```
p
```

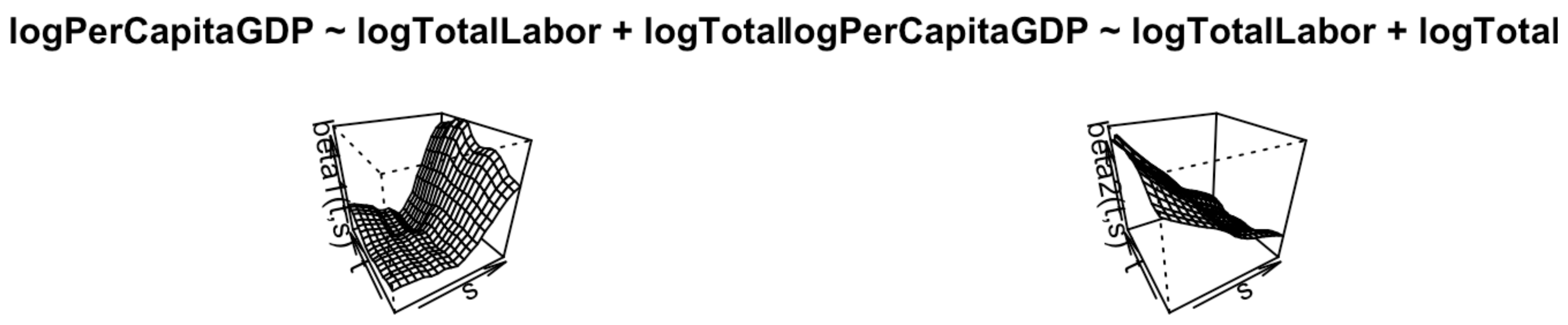
```
## Warning: 'surface' objects don't have these attributes: 'ylab', 'zlab', 'theta', '
phi'
## Valid attributes include:
## 'type', 'visible', 'name', 'uid', 'ids', 'customdata', 'meta', 'hoverlabel', 'stre
am', 'uirevision', 'z', 'x', 'y', 'text', 'hovertext', 'hovertemplate', 'connectgaps'
, 'surfacecolor', 'cauto', 'cmin', 'cmax', 'cmid', 'colorscale', 'autocolorscale', 'r
eversescale', 'showscale', 'colorbar', 'coloraxis', 'contours', 'hidesurface', 'light
position', 'lighting', 'opacity', '_deprecated', 'hoverinfo', 'xcalendar', 'ycalendar
', 'zcalendar', 'scene', 'idssrc', 'customdatasrc', 'metasrc', 'zsrc', 'xsrc', 'ysrc'
, 'textsrc', 'hovertextsrc', 'hovertemplatesrc', 'surfacecolorsrc', 'hoverinfosrc', '
key', 'set', 'frame', 'transforms', '_isNestedKey', '_isSimpleKey', '_isGraticule', '
_bbox'
```

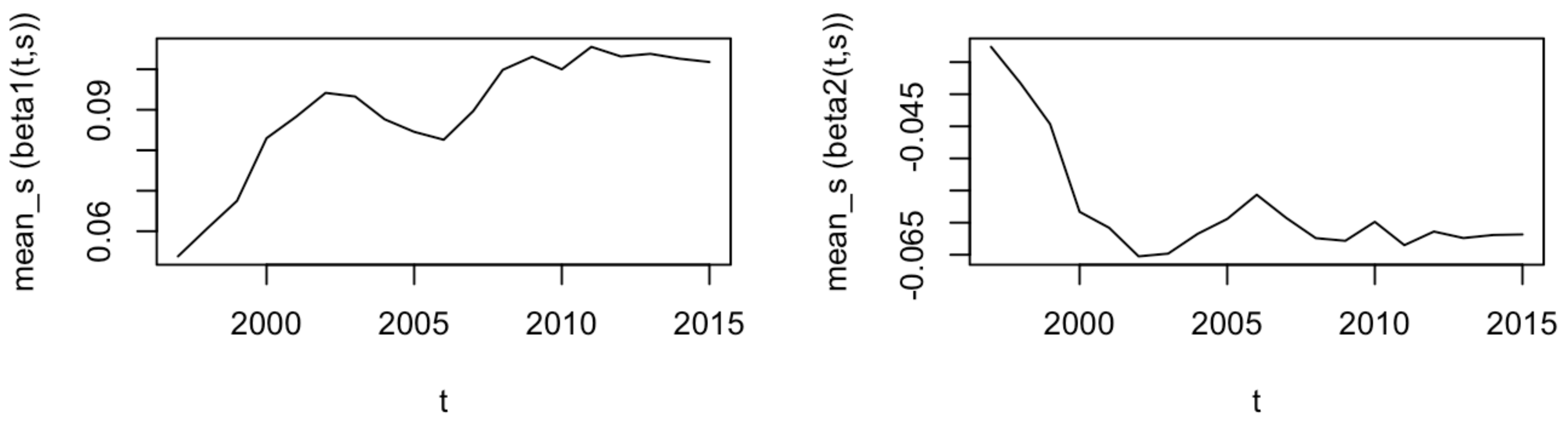


```
par(mfrow= c(2,2))
persp(x = t, y = t, z = denseFLM$betaList[[1]] , xlab = "s",  zlab ="beta1(t,s)", the
ta = -30, phi = 30, main = "logPerCapitaGDP ~ logLaborPerc + logTotalPop")
persp(x = t, y = t, z = denseFLM$betaList[[2]] , xlab = "s",  zlab ="beta2(t,s)", the
ta = -30, phi = 30, main = "logPerCapitaGDP ~ logLaborPerc + logTotalPop")
plot(t, colMeans(denseFLM$betaList[[1]]), ylab = "mean_s (beta1(t,s))", type = "l")
plot(t, colMeans(denseFLM$betaList[[2]]), ylab = "mean_s (beta2(t,s))", type = "l")
```



```
par(mfrow= c(2,2))
persp(x = t, y = t, z = denseFLM2$betaList[[1]] , xlab = "s",  zlab ="beta1(t,s)", th
eta = -30, phi = 30, main = "logPerCapitaGDP ~ logTotalLabor + logTotalPop")
persp(x = t, y = t, z = denseFLM2$betaList[[2]] , xlab = "s",  zlab ="beta2(t,s)", th
eta = -30, phi = 30, main = "logPerCapitaGDP ~ logTotalLabor + logTotalPop")
plot(t, colMeans(denseFLM2$betaList[[1]]), ylab = "mean_s (beta1(t,s))", type = "l")
plot(t, colMeans(denseFLM2$betaList[[2]]), ylab = "mean_s (beta2(t,s))", type = "l")
```





```
# par(mfrow= c(1,2))
# persp(x = t, y = t, z = denseFLM3$betaList[[1]] , ylab = "s",  zlab ="beta1(t,s)",
theta = -30, phi = 30, main = "logPerCapitaGDP ~ logLaborPerc")
# plot(denseFLM$yHat,y,xlab='fitted Y', ylab='observed Y')
# abline(coef=c(0,1),col=8)
```

Note the intercept estimation of Functional-regression is pretty close to zero, which is because the change in the level is absorbed by the principal components. From FLM result of `logPerCapitaGDP ~ logLaborPerc + logTotalPop`, the relationship between localPerCapitaGDP and logLaborPerc (β_1) showed rigglet behaviour but upward trend throughout the years while localPerCapitaGDP and logTotalPop (β_2) exhibits varied correlation. This result is roughly consistent with what we get from concurrent model. Note in the concurrent model, β_0 exibits siginificant uptrend, which in the FPC model is absorbed in $\beta_1(t,s)$ and $\beta_2(t,s)$.

Same analysis is also performed for `logPerCapitaGDP ~ logTotalLabor + logTotalPop`.