Package 'FRegSigCom'

May 7, 2018

Type Package

Title Functional Regression using Signal Compression Approach

Version 0.2.2

Author Ruiyan Luo, Xin Qi

Maintainer Ruiyan Luo <rluo@gsu.edu>

Description Signal compression methods for function-on-function regression with functional response and functional predictors, including linear models with both scalar and functional predictors for a small number of functional predictors, linear models with functional predictors for a large number of functional predictors, stepwise selection for FOF models with two-way interactions, and nonlinear models. References are linked to via the URL below.

URL https://doi.org/10.1080/01621459.2016.1164053,
https://www.sciencedirect.com/science/article/pii/S0047259X16302536?via%3Dihub.

License GPL-2

Encoding UTF-8

LazyData TRUE

NeedsCompilation yes

Imports Rcpp

LinkingTo Rcpp, RcppEigen

Depends fda, Matrix

Suggests refund, MASS

Collate 'multiple_function_on_function.R'
'nonlinear_function_on_function.R'
'high_dimensional_function_on_function.R' 'ff.interaction.R'
'RcppExports.R'

RoxygenNote 6.0.1.9000

Repository CRAN

Date/Publication 2018-05-07 09:12:13

R topics documented:

`air` *Air quality data*

Description

Data collected hourly in 355 days (days with missing values removed) in a significantly polluted area within an Italian city.

Usage

```
data("air")
```

Format

A list of 7 matrices with 355 rows and 24 columns:

NO2 Hourly observation of concentration level of NO2 in 355 days

CO Hourly observation of concentration level of CO in 355 days

NMHC Hourly observation of concentration level of NMHC in 355 days

NOx Hourly observation of concentration level of NOx in 355 days

C6H6 Hourly observation of concentration level of C6H6 in 355 days

temperature Hourly observation of concentration level of temperature in 355 days

humidity Hourly observation of concentration level of humidity in 355 days

Source

Data link

References

De Vito, S. etal (2008) Sensors and Actuators B: Chemical, 129: 50-757.

Xin Qi and Ruiyan Luo. (Accepted) Nonlinear function on function additive model with multiple predictor curves. Stistica Sinica.

See Also

cv.nonlinear

Examples

```
data(air)
str(air)
```

`cv.ff.interaction` *Cross-validation for function-on-function regression models with specified main effects and two-way interaction terms*

Description

This function is used to perform cross-validation and build the final model using the signal compression approach for the following linear function-on-function regression model with main effects, two-way interaction between them, and quadratic terms. Let $\{X_i(s), 1 \leq i \leq p\}$ be p potential functional predictors. The model is given by

$$Y(t) = \mu(t) + \sum_{i \in M} \int_{a_i}^{b_i} X_i(s)\beta_i(s,t)ds + \sum_{(i,j) \in I} \int_{a_i}^{b_i} \int_{a_j}^{b_j} X_i(u)X_j(v)\gamma_{ij}(u,v,t)dudv + \epsilon(t),$$

where $\mu(t)$ is the intercept function. The index set M of main effects is a subset of $\{1, ..., p\}$, and the index set I of interactions and quadratic effects is a subset of the collection of all possible pairs $\{(i,j), 1 \leq i \leq j \leq p\}$. The $\{\beta_i(s,t), i \in M\}$ and $\{\gamma_{ij}(u,v,t), (i,j) \in I\}$ are the corresponding coefficient functions. The $\epsilon(t)$ is the noise function.

Usage

```
cv.ff.interaction(X, Y, t.x, t.y, main.effect, interaction.effect=NULL,
       s.n.basis=40, t.n.basis=40, inter.n.basis=20, basis.type="Bspline",
       K.cv=5, all.folds=NULL, upper.comp=8, thresh=0.01)
```

Arguments

`X`	a list of p potential functional predictors. Its i-th element is the $n \times m_i$ data matrix for the i-th potential functional predictor $X_i(s)$, where n is the sample size and m_i is the number of observation time points for $X_i(s)$.
`Y`	the $n \times m$ data matrix for the functional response $Y(t)$, where n is the sample size and m is the number of the observation time points for $Y(t)$.

`t.x` a list of length p. Its i-th element is the vector of obesrvation time points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.

`t.y` the vector of obesrvation time points of the functional response $Y(t)$.

`main.effect` a vector of indices for main effects. It is a subset of $\{1, 2, ..., p\}$.

`interaction.effect`
a matrix of two columns. Each row of this matrix specifies the index of a two-way interaction or a quadratic effect. Default is NULL.

`s.n.basis` the number of basis functions used for estimating the functions $\psi_{ik}(s)$ (see the reference for details). Default is 40.

`t.n.basis` the number of basis functions used for estimating the functions $w_k(t)$. Default is 40.

`inter.n.basis` the number of one-dimensional basis functions used to construct the tensor product basis functions for estimating the functions $\phi_{ijk}(u, v)$. Default is 20.

`basis.type` the type of basis functions. Only "BSpline" (default) and "Fourier" are supported.

`K.cv` the number of CV folds. Default is 5.

`all.folds` a list of cross validation sets. The length of the list is equal to `K.cv`. The i-th element of the list is the collection of indices of observations assigned to the i-th cross validation set. Default is NULL, and the cross validation sets are generated in function cv.ff.interaction.

`upper.comp` the upper bound for the maximum number of components to be calculated. Default is 8.

`thresh` a number between 0 and 1 specifying the minimum proportion of variation to be explained by each selected component relative to all the selected components. This will help determining the upper bound of the number of components to be used in CV, and the optimal number of components will be determined from 1,2,..., to the minimum of this uppber bound and upper.com by CV. A smaller thresh value leads to more components to be selected and a longer running time. A larger thresh value needs less running time, but may miss some important components and lead to a larger prediction error. Default is 0.01.

Value

An object of the "cv.ff.interaction" class, which is used in the function `pred.ff.interaction` for prediction and `getcoef.ff.interaction` for extracting the estimated coefficient functions.

`fitted_model` a list for interval use.

`y_penalty_inv` a list for internal use.

`X` the input `X`.

`Y` the input `Y`.

`x.smooth.params`
a list for internal use.

`y.smooth.params`
a list for internal use.

Author(s)

Xin Qi and Ruiyan Luo

References

Ruiyan Luo and Xin Qi (under revision) Interaction model and model selection for function-on-function regression.

See Also

pred.ff.interaction, getcoef.ff.interaction

Examples

```
library(FRegSigCom)
data(ocean)

Y=ocean$Salinity
X=list()
X[[1]]=ocean$Potential.density
X[[2]]=ocean$Temperature
X[[3]]=ocean$Oxygen
X[[4]]=ocean$Chloropigment
n.curves=length(X)
ntot=dim(Y)[1]
ntrain=50
ntest=ntot-ntrain
X.uncent=X
for(i in 1:n.curves){
  X[[i]]=scale(X.uncent[[i]],center=TRUE, scale=FALSE)
}
lengthX=dim(X[[1]])[2]
lengthY=dim(Y)[2]
t.x=seq(0,1,length=lengthX)
t.y=seq(0,1,length=lengthY)
I.train=sample(1:ntot, ntrain)
X.train=list()
X.test=list()
t.x.all=list()
for(i in 1:n.curves)
{
  X.train[[i]]=X[[i]][I.train,]
  X.test[[i]]=X[[i]][-I.train,]
  t.x.all[[i]]=t.x
}
Y.train=Y[I.train, ]
Y.test=Y[-I.train, ]


#############################################################################
# an interaction model with given main effects and two-way interactions
```

```
#########################################################################
main.effect=c(1,2,3)
inter.effect=rbind(c(1,1),c(1,2))
fit.inter=cv.ff.interaction(X.train, Y.train, t.x.all, t.y,  main.effect, inter.effect)
Y.pred=pred.ff.interaction(fit.inter,  X.test)
error.inter=mean((Y.pred-Y.test)^2)
print(c("error.inter=", error.inter))
#coef.obj=getcoef.ff.interaction(fit.inter)
#str(coef.obj)
```

---

cv.hd *Cross-validation for sparse linear function-on-function regression with a large number of functional predictors*

Description

Conduct cross-validation and build the final model for the following function-on-function regression model:

$$Y(t) = \mu(t) + \sum_{i=1}^{p} \int_{a_i}^{b_i} X_i(s)\beta_i(s,t)ds + \varepsilon(t)$$

where $\mu(t)$ is the intercept function. The $\{X_i(s), 1 \leq i \leq p\}$ are p functional predictors and $\{\beta_i(s,t), 1 \leq i \leq p\}$ are corresponding coefficient functions. The $\epsilon(t)$ is the noise function. The p can be much larger than the sample size.

We require that all the sample curves of each functional predictor are observed in a common dense set of time points, but the set can be different for different functional predictor. All the sample curves of the response are observed in a common dense set.

This method estimates a special decomposition of the coefficient functions induced by the KL expansion of the signal function:

$$\beta_i(s,t) = \sum_{k=1}^{\infty} \psi_{ik}(s)w_k(t), 1 \leq i \leq p.$$

We first estimate $\{\psi_{ik}(s), 1 \leq i \leq p\}$ for each $0 < k < K$, with a simultaneous sparse and smooth penalty imposed,

$$P(\psi_1(s), ..., \psi_p(s)) = \tau\{(1-\lambda)\sum_{i=1}^{p} ||\psi_i||_\eta^2 + \lambda(\sum_{i=1}^{p} ||\psi_i||_\eta)^2\},$$

where $||\psi_i||_\eta^2 = ||\psi_i||^2 + \eta||\psi_i''||^2$. Then we estimate $\{w_k(t), 0 < k < K_0\}$ by a penalized least square problem with the smoothness of $w_k(t)$ tuned by κ, where K is the number of componets to be selected and will be chosen by cross-validation. The simultaneous sparse and smooth penalty will make most estimates of $\psi_{ik}(s)$'s, and hence $\beta_i(s,t)$'s, exactly equal to zero.

Usage

```
cv.hd(X, Y, t.x.list, t.y, K.cv = 5, s.n.basis = 25, t.n.basis = 50,
       thresh = 0.01)
```

Arguments

X	a list of length p. Its i-th element is the $n \times m_i$ data matrix for the i-th funtional predictor $X_i(s)$, where n is the sample size and m_i is the number of the observation points for $X_i(s)$.
Y	the $n \times m$ data matrix for the functional response $Y(t)$, where n is the sample size and m is the number of the observation points for $Y(t)$.
t.x.list	a list of length p. Its i-th element is the vector of observation points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.
t.y	the vector of obesrvation points of the functional response $Y(t)$.
K.cv	the number of CV folds. Default is 5.
s.n.basis	the number of B-spline basis functions used for estimating the functions $\psi_{ik}(s)$. Default is 25.
t.n.basis	the number of B-spline basis functions used for estimating the functions $w_k(t)$. Default is 50.
thresh	a number between 0 and 1 specifying the minimum proportion of variation to be explained by each selected component relative to all the selected components. This will determine the upper bound of the number of components for CV, and the optimal number of components will be determined from 1,2,..., to this uppber bound by CV. A smaller thresh value leads to more components to be selected and a longer running time. A larger thresh value needs less running time, but may miss some important components and lead to a larger prediction error. Default is 0.01.

Value

An object of the "cv.hd" class, which is used in the function pred.hd for prediction and getcoef.hd for extracting the estimated coefficient functions.

errors	list for CV errors.
min.error	minimum CV error.
opt.index	index of the optimal tunning parameters.
params.set	the set of candidate values for tuning parameters used in CV. It's a matrix with 4 columns corresponding to the tuning parameters τ, λ, η, κ, respectively.
opt.K	optimal number of components to be selected.
opt.tau	optimal value for τ tuning the simultaenous sparse-smooth penalty on $\{\psi_{ik}(s), 1 \leq i \leq p\}$.
opt.lambda	optimal value for λ, the sparsity parameter for $\{\psi_{ik}(s), 1 \leq i \leq p\}$.
opt.eta	optimal value for η, the smoothness parameter for $\psi_{ik}(s)$.
opt.kappa	optimal value for κ, the smoothness parameter for $w_k(t)$.
maxK.ret	a list for internal use.
t.y	the input t.y.
y.params	a list for internal use.

Author(s)

Ruiyan Luo and Xin Qi

References

Xin Qi and Ruiyan Luo (2018) Function-on-Function Regression with thousands of predictive curves, Jounal of Multivariate Analysis 163:51-66. https://doi.org/10.1016/j.jmva.2017.10.002

Examples

```
########################################
#toy example using the air quality data with p=1
########################################
data(air)
t.x=seq(0,1,length=24)
t.y=seq(0,1,length=24)
air.cv=cv.hd(X=list(air[[2]][1:20,]), Y=air[[1]][1:20,], list(t.x), t.y,
             K.cv=2, s.n.basis = 8, t.n.basis = 8)
air.pred=pred.hd(air.cv, list(air[[2]][1:2,]))
predict.error=mean((air.pred-air[[1]][1:2,])^2)
print(c("predict error=", predict.error))



########################################
#example with simulated data and p=200
########################################

rm(list=ls())
ptm <- proc.time()
library(MASS)

n.curves=200 # total number of predictor curves.

t.x=seq(0,1,length.out=80) # observation points for X_i(s).
t.y=seq(0,1,length.out=100) # observation points for Y(t).
ntrain <- 100 # number of observations in training data
nnew <-50 # number of new observations
ntot <- ntrain+nnew
########################################
##generate the predictor curves using cos() basis functions.
#########################################
W <- list()
for(j in 1:(n.curves+5))
{
  W[[j]] <- 0
  for(k in 1:30)
  W[[j]] <- W[[j]]+rnorm(ntot) %*% t(cos(k*pi*t.x))/k
}
X=lapply(1:n.curves,
function(k){(W[[k]]+W[[k+1]]+W[[k+2]]+W[[k+3]]+W[[k+4]]
```

```
            +W[[k+5]])/k})
########################################
##generate the coefficient functions. Only the first five are nonzero.
#########################################
mu=sin(2*pi*t.y)
B.coef=list()
B.coef[[1]]=sapply(1:length(t.y), function(k){4*t.x^2*t.y[k]})
B.coef[[2]]=sapply(1:length(t.y), function(k){4*sin(pi*t.x)*cos(pi*t.y[k])})
B.coef[[3]]=sapply(1:length(t.y), function(k){4*exp(-((t.x-0.5)^2+(t.y[k]-0.5)^2)/2)})
B.coef[[4]]=sapply(1:length(t.y), function(k){4*t.x/(1+t.y[k]^2)})
B.coef[[5]]=sapply(1:length(t.y), function(k){4*log(1+t.x)*sqrt(t.y[k])})
########################################
##generate the response curves
#########################################
Y=0
for(j in 1:5){
Y<- Y+ (X[[j]] %*% B.coef[[j]])/length(t.x)
}
E=sqrt(0.01)*matrix(rnorm(ntot*length(t.y)), ntot, length(t.y))
Y=rep(1,ntot)%*%t(mu)+Y+E
X.train=lapply(1:n.curves, function(k){X[[k]][(1:ntrain),]})
X.new=lapply(1:n.curves, function(k){X[[k]][-(1:ntrain),]})
Y.train <- Y[1:ntrain,]
Y.new <- Y[-(1:ntrain),]
E.new=E[-(1:ntrain),]

########################################
##fit the model using the sparse function-on-function method
#########################################
t.x.list=lapply(1:n.curves, function(k){t.x})
fit.cv <- cv.hd(X.train, Y.train, t.x.list, t.y,   K.cv=5, s.n.basis=25, t.n.basis=40)
Y.pred=pred.hd(fit.cv, X.new)
predict.error=mean((Y.pred-Y.new)^2)
est.error=mean((Y.pred-Y.new+E.new)^2)
print(c("predict error=", predict.error))
print(c("estimation error=", est.error))
est.coefficient=getcoef.hd(fit.cv)
mu.est=est.coefficient[[1]]
beta.est=est.coefficient[[2]]
```

`cv.nonlinear` *Cross-validation for nonlinear function-on-function regression*

Description

This function is used to perform cross-validation and build the final model using signal compression approach for the following nonlinear function-on-function regression model:

$$Y(t) = \mu(t) + \sum_{i=1}^{p} \int_{a_i}^{b_i} F_i(X_i(s), s, t)ds + \varepsilon(t)$$

where $\mu(t)$ is the intercept function, $\{F_i(x,s,t), 1 \le i \le p\}$ are all unspecified nonlinear functions of x, s, t, $\{X_i(s), 1 \le i \le p\}$ are functional predictors, and $\epsilon(t)$ is the noise function.

In this method, we require that all the sample curves of each functional predictor be observed in a common dense set, but the observation points can be different for different functional predictors. All the sample curves of the functional response are observed in a common dense set.

This method estimates a special decomposition:

$$F_i(x,s,t) = \sum_{k=1}^{\infty} G_{ik}(x,s) w_k(t), 1 \le i \le p.$$

We first estimate $\{G_{ik}(x,s), 1 \le i \le p\}$ for each $k > 0$ by solving a generalized functional eigenvalue problem with smoothness tuning parameter λ on its Sobolev norm, and then estimate $\{w_k(t), k > 0\}$ by penalized least square with smoothness tuning parameter κ.

Usage

```
cv.nonlinear(X, Y, t.x.list, t.y, s.n.basis = 30, x.n.basis = 30,
        t.n.basis = 30, K.cv = 5, upper.com = 15, thresh = 0.005)
```

Arguments

`X`	a list of length p, the number of functional predictors. Its i-th element is the $n \times m_i$ data matrix for the i-th functional predictor $X_i(s)$, where n is the sample size and m_i is the number of observation time points for $X_i(s)$.
`Y`	the $n \times m$ data matrix for the functional response $Y(t)$, where n is the sample size and m is the number of the observation time points for $Y(t)$.
`t.x.list`	a list of length p. Its i-th element is the vector of obesrvation time points of the i-th functional predictor $X_i(s)$, $1 \le i \le p$.
`t.y`	the vector of obesrvation time points of the functional response $Y(t)$.
`s.n.basis`	the number of B-spline basis functions for the argument s. Default is 30.
`x.n.basis`	the number of B-spline basis functions for x. Default is 30.
`t.n.basis`	the number of B-spline basis functions for t. Default is 30.
`K.cv`	the number of CV folds. Default is 5.
`upper.com`	the upper bound for the maximum number of components need to calculate. Default is 15.
`thresh`	a number between 0 and 1 specifying the minimum proportion of variation to be explained by each selected component relative to all the selected components. This will help determining the upper bound of the number of components to be used in CV, and the optimal number of components will be determined from 1,2,..., to the minimum of this uppber bound and upper.com by CV. A smaller thresh value leads to more components to be selected and a longer running time. A larger thresh value needs less running time, but may miss some important components and lead to a larger prediction error. Default is 0.005.

Value

A fitted CV-object, which is used in the function `pred.nonlinear` for prediction and `getcoef.nonlinear` for extracting the estimated coefficient functions.

`opt.index`	index of the optimal lambda.
`min.error`	minimum CV error.
`errors`	list for CV errors.
`lambda.set`	the set of λ values used in CV.
`kappa.set`	the set of κ values used in CV.
`opt.K`	optimal number of components to be selected.
`opt.lambda`	optimal value for λ, the tuning parameter for $\{G_{ik}(x,s), 1 \leq i \leq p\}$.
`opt.kappa`	optimal value for κ, the smoothness tuning parameter for $w_k(t)$.
`fit.1`	a list for internal use.
`y.params`	a list for internal use.

Author(s)

Ruiyan Luo and Xin Qi

References

Xin Qi and Ruiyan Luo. (Accepted) Nonlinear function on function additive model with multiple predictor curves. Stistica Sinica.

Examples

```
########################################
# toy example
# fit a nonlinear regression model with p=1 in the air quality data
########################################

ptm <- proc.time()
data(air)
t.x=seq(0,1,length=24)
t.y=seq(0,1,length=24)
air.nonlinear.cv=cv.nonlinear(X=list(air[[2]][1:60,]), Y=air[[1]][1:60,], list(t.x),
           t.y, K.cv=2, s.n.basis = 8, x.n.basis = 10, t.n.basis = 8)
air.pred=pred.nonlinear(air.nonlinear.cv, list(air[[2]][1:2,]))
est.coefficient=getcoef.nonlinear(air.nonlinear.cv)
print(proc.time()-ptm)




########################################
# example with simulated data and p=3
########################################
rm(list=ls())
```

```
ptm <- proc.time()
library(MASS)

n.curves=3 # the number of predictor curves.
t.x=seq(0,1,length.out=80) # observation points for all six predictor curves.
t.y=seq(0,1,length.out=100) # observation points for response curve.
nnew <- 50
ntrain <-100
ntot <- nnew+ntrain

#####################################################
#generate the predictor curves using sin() and cos() basis functions.
#####################################################
X=list()
X[[1]]=X[[2]]=X[[3]]=0
Sigma=matrix(0.5, 3, 3)
diag(Sigma)=1
for(k in 1:20){
  coef.mtx.1=mvrnorm(ntot, rep(0,3), Sigma)
  coef.mtx.2=mvrnorm(ntot, rep(0,3), Sigma)
  X[[1]]=X[[1]]+coef.mtx.1[,1]%*%t(sin(k*pi*t.x))/k+coef.mtx.2[,1]%*%t(cos(k*pi*t.x))/k
  X[[2]]=X[[2]]+coef.mtx.1[,2]%*%t(sin(k*pi*t.x))/k+coef.mtx.2[,2]%*%t(cos(k*pi*t.x))/k
  X[[3]]=X[[3]]+coef.mtx.1[,3]%*%t(sin(k*pi*t.x))/k+coef.mtx.2[,3]%*%t(cos(k*pi*t.x))/k
}

#####################################################
#generate the nonlinear function and calculate the \int_0^1F_j(X_j(s),s,t)ds, j=1,..,3.
#####################################################
A.1=sapply(1:ntot, function(i){t.x})
A.1=t(A.1)
S.1.mtx=sapply(1:length(t.y), function(i){
         tmp= (A.1+X[[1]]-3*t.y[i])^2; apply(tmp,1,mean)})
S.2.mtx= sapply(1:length(t.y), function(i){
         tmp=cos(2*X[[2]]*A.1)+5*sin(X[[2]]*t.y[i]^(0.5)) ; apply(tmp,1,mean)})
S.3.mtx=sapply(1:length(t.y), function(i){
         tmp= 1/(1+X[[3]]^2+A.1*t.y[i]); apply(tmp,1,mean)})

#####################################################
#generate the noise function, and response function
#####################################################
S.mtx=0.4*(S.1.mtx+S.2.mtx+S.3.mtx)
E<-matrix(rnorm(ntot*length(t.y)),ntot, length(t.y))
mu.val<- cos(4*pi*t.y)
Y<-t(sapply(1:ntot, function(i){mu.val+S.mtx[i,] + E[i,]}))
X.train=list()
X.new=list()
for(j in 1:n.curves)
{  X.train[[j]]=X[[j]][1:ntrain,]
   X.new[[j]]=X[[j]][-(1:ntrain),]
}
Y.train <- Y[1:ntrain, ]
Y.new <- Y[-(1:ntrain), ]
E.new <- E[-(1:ntrain),]
```

```
#######################################################
#fit the nonlinear function-on-function model and make prediction
#######################################################
t.x.list=lapply(1:n.curves, function(k){t.x})
fit.cv=cv.nonlinear(X.train, Y.train, t.x.list, t.y)
Y.pred=pred.nonlinear(fit.cv, X.new)
nonlinear.error=mean((Y.pred-Y.new)^2)
print(c("nonlinear prediction error=",nonlinear.error))
nonlinear.esterror=mean((Y.pred-Y.new+E.new)^2)
print(c("nonlinear estimation.error=",nonlinear.esterror))

#######################################################
#extract the estimated mu(t), F(x,s,t)
#######################################################
F.list=getcoef.nonlinear(fit.cv)
mu=F.list$mu
F=F.list$F
X.grid=F.list$X.grid
print(proc.time()-ptm)
#   user  system elapsed
#  59.30    7.63   66.94
```

`cv.sigcom` *Cross-validation for linear function-on-function regression*

Description

This function is used to perform cross-validation and build the final model using the signal compression approach for the following linear function-on-function regression model that includes functional (and scalar) predictors:

$$Y(t) = \mu(t) + Z^T\alpha(t) + \sum_{i=1}^{p}\int_{a_i}^{b_i} X_i(s)\beta_i(s,t)ds + \varepsilon(t)$$

where $\mu(t)$ is the intercept function. Z is a multivariate predictor and $\alpha(t)$ is the vector of corresponding coefficient functions. Z =NULL means that there is no scalar predictor. The $\{X_i(s), 1 \leq i \leq p\}$ are p functional predictors and $\{\beta_i(s,t), 1 \leq i \leq p\}$ are their corresponding coefficient functions, where p is a positive integer. When p is large (e.g., greater than 6), one may want to consider sparsity penalty (see `cv.hd`()). The $\epsilon(t)$ is the noise function.

We require that all the sample curves of each functional predictor are observed in a common dense grid of time points, but the grid can be different for different predictors. All the sample curves of the functional response are observed in a common dense grid.

This method estimates a special decomposition of the coefficient functions:

$$\beta_i(s,t) = \sum_{k=1}^{\infty}\psi_{ik}(s)w_k(t), 1 \leq i \leq p$$

We first estimate $\{\psi_{ik}(s), 1 \leq i \leq p\}$ for each $k > 0$ by solving a generalized functional eigenvalue problem with penalty

$$\lambda \sum_{i=1}^{p} \{||\psi_{ik}||^2 + \tau ||\psi''_{ik}||^2\}$$

and then estimate $w_k(t), k > 0$ by penalized least square with smoothness tuning parameter κ.

Usage

```
cv.sigcom(X, Y, t.x, t.y, Z = NULL, s.n.basis = 50, t.n.basis = 50,
          K.cv = 5, upper.comp = 10, thresh = 0.005, tol = 1e-12)
```

Arguments

X	a list of length p, the number of functional predictors. Its i-th element is the $n \times m_i$ data matrix for the i-th functional predictor $X_i(s)$, where n is the sample size and m_i is the number of observation time points for $X_i(s)$.
Y	the $n \times m$ data matrix for the functional response $Y(t)$, where n is the sample size and m is the number of the observation time points for $Y(t)$.
t.x	a list of length p. Its i-th element is the vector of obesrvation time points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.
t.y	the vector of obesrvation time points of the functional response $Y(t)$.
Z	the $n \times q$ data matrix for multivariate scalar predictors, where n is the sample size and q is the number of the scalar predictors. Default is NULL, indicating no scalar predictors.
s.n.basis	the number of B-spline basis functions used for estimating the functions $\psi_{ik}(s)$. Default is 50.
t.n.basis	the number of B-spline basis functions used for estimating the functions $w_k(t)$. Default is 50.
K.cv	the number of CV folds. Default is 5.
upper.comp	the upper bound for the maximum number of components to be calculated. Default is 10.
thresh	a number between 0 and 1 specifying the minimum proportion of variation to be explained by each selected component relative to all the selected components. This will help determining the upper bound of the number of components to be used in CV, and the optimal number of components will be determined from 1,2,..., to the minimum of this uppber bound and upper.com by CV. A smaller thresh value leads to more components to be selected and a longer running time. A larger thresh value needs less running time, but may miss some important components and lead to a larger prediction error. Default is 0.005.
tol	the iteration convergence tolerance.

Value

An object of the "cv.sigcom" class, which is used in the function `pred.sigcom` for prediction and `getcoef.sigcom` for extracting the estimated coefficient functions.

`t.x.list`	the input t.x.
`t.y`	the input t.y.
`Z`	the input Z
`opt.index`	index of the optimal $lambda$.
`opt.lambda`	optimal value for $lambda$, the parameter tuning the penalty on $\psi_{ik}(s), 1 \leq i \leq p$.
`opt.tau`	optimal value for tau, the smoothness parameter for $\psi_{ik}(s), 1 \leq i \leq p$.
`opt.kappa`	optimal value for $kappa$, the smoothness parameter for $w_k(t)$.
`opt.K`	optimal number of components to be selected.
`min.error`	minimum CV error.
`errors`	list for CV errors.
`x.smooth.params`	a list for internal use.
`y.smooth.params`	a list for internal use.
`fit.1`	a list for internal use.
`is.null.Z`	a logic value indicating whether Z is null.

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo and Xin Qi (2017) Function-on-Function Linear Regression by Signal Compression, Journal of the American Statistical Association. 112(518), 690-705. https://doi.org/10.1080/01621459.2016.1164053

Examples

```
#################################################################
# Example 1: function-on-function model without scalar predictors
#################################################################

ptm <- proc.time()
library(MASS)

n.curves <- 2 #number of predictor curves
nnew <- 50 # number of observations in new data
ntrain <- 10 # number of observations in training data
ntot <- nnew+ntrain
t.x <- seq(0,2,length=50) # all the four predictor curves are observed
             # at 50 equally spaced points in [0,2].
t.y <- seq(0,1,length=50) # the response curve is observed at 50
```

```
              # equally spaced points in [0,1].

# functions for mu(t) and beta_i(s,t)
mu.fun <- function(t){
    2*exp(-(t-1)^2)*sin(2*pi*t)
}
beta.fun.1 <- function(s,t)
{
    sin(1.5*pi*s)*sin( pi*t)
}
beta.fun.2 <- function(s,t)
{
    exp(-5*(s-0.5)^2-3*(t-0.5))+2*exp(-5*(s-1.5)^2-3*(t-0.5))
}

#generate the predictor curves using sin() and cos() basis functions.
 X=list()
 Sigma=matrix(0.5, 2, 2)
 diag(Sigma)=1
 sins=lapply(1:5, function(i){t(sin(i*pi*t.x))/i})
 coss=lapply(1:5, function(i){t(cos(i*pi*t.x))/i})
 coefs1=lapply(1:5, function(i){mvrnorm(ntot,rep(0,2),Sigma)})
 coefs2=lapply(1:5, function(i){mvrnorm(ntot,rep(0,2),Sigma)})
 X[[1]]=Reduce("+",lapply(1:5, function(i){coefs1[[i]][,1] %*% sins[[i]]
             +coefs2[[i]][,1] %*% coss[[i]]}))
 X[[2]]=Reduce("+",lapply(1:5, function(i){coefs1[[i]][,2] %*% sins[[i]]
             +coefs2[[i]][,2] %*% coss[[i]]}))
 mu.val=mu.fun(t.y)
 beta.val=list()
 beta.val[[1]]<- outer(t.x,t.y,beta.fun.1)
 beta.val[[2]]<- outer(t.x,t.y,beta.fun.2)
# generate sample curves for the noise and the respose function.
 E<-matrix(rnorm(ntot*length(t.y)),ntot, length(t.y))
 delta<-(max(t.x)-min(t.x))/length(t.x)
 Y<-t(sapply(1:ntot, function(i){mu.val+(X[[1]][i,]%*% beta.val[[1]]
   +X[[2]][i,]%*% beta.val[[2]])*(max(t.x)-min(t.x))/length(t.x)+ E[i,]}))

# gereate the training data and perform CV
 X.train=lapply(1:n.curves, function(j){X[[j]][1:ntrain,]})
 t.x.list=lapply(1:n.curves,function(j){t.x})
 Y.train <- Y[1:ntrain, ]
 fit.cv=cv.sigcom(X.train, Y.train, t.x.list, t.y, s.n.basis=20, t.n.basis=20)

# prediction and estimation error
 X.new=lapply(1:n.curves, function(j){X[[j]][-(1:ntrain),]})
 Y.new <- Y[-(1:ntrain), ]
 E.new=E[-(1:ntrain), ]
 Y.pred=pred.sigcom(fit.cv, X.new)
 error <- mean((Y.pred-Y.new)^2)
 print(c(" prediction error=", error))
 est.error <- mean((Y.pred-Y.new+E.new)^2)
# extract the estimated intercept and coefficient functions
 print(c("estimation error for regression function (or signal function)=", est.error))
```

```
 coef.est.all=getcoef.sigcom(fit.cv)
 mu.est=coef.est.all[[1]]
 beta.est=coef.est.all[[2]]
print(proc.time()-ptm)

#################################################################
# Example 2: function-on-function model with scalar predictors
#################################################################


ptm <- proc.time()
library(MASS)

n.curves <- 2 # there are two predictor curves. They have different range
              # and observation points
ntest <- 500
ntrain <- 100
ntot <- ntest+ntrain
t.x.1=seq(0,2,length=100) # the vector of observation points for X_1(s)
t.x.2=seq(0,1,length=50) # the vector of observation points for X_1(s)
t.y=seq(0,1,length=60) # the vector of observation points for Y(t)

########################################
# functions for mu(t) and beta_i(s,t)
########################################
mu.fun <- function(t){
    cos(4*pi*t)
}

beta.fun.1 <- function(s,t)
{
  1/(1+s^2+t^2)
}
beta.fun.2 <- function(s,t)
{
   log(1+s*t)
}
########################################
X=list()
X[[1]]=X[[2]]=0
Sigma=matrix(0.3, 2, 2)
diag(Sigma)=1
for(k in 1:20){
   coef.mtx.1=mvrnorm(ntot, rep(0,2), Sigma)
   coef.mtx.2=mvrnorm(ntot, rep(0,2), Sigma)
   X[[1]]=X[[1]]+coef.mtx.1[,1]%*% t(sin(k*pi*t.x.1))/k^2
                +coef.mtx.2[,1]%*% t(cos(k*pi*t.x.1))/k^2
   X[[2]]=X[[2]]+coef.mtx.1[,2]%*% t(sin(k*pi*t.x.2))/k^2
                +coef.mtx.2[,2]%*% t(cos(k*pi*t.x.2))/k^2
  }
 mu.val=mu.fun(t.y)
 beta.val=list()
 beta.val[[1]]<- outer(t.x.1,t.y,beta.fun.1)
```

```
 beta.val[[2]]<- outer(t.x.2,t.y,beta.fun.2)

#  generate the scalar predictors and the corresponding coefficient functions
 Sigma.scalar=matrix(0.6, 3, 3)
 diag(Sigma.scalar)=1
 Z=mvrnorm(ntot, rep(0,3), Sigma.scalar)
 beta.scalar=rbind(exp(-t.y), 1+t.y^2, sin(3*pi*t.y))
 E<-matrix(rnorm(ntot*length(t.y)),ntot, length(t.y))
 Y<- t(sapply(1:ntot, function(i){mu.val
     +X[[1]][i,] %*% beta.val[[1]]*(max(t.x.1)-min(t.x.1))/length(t.x.1)
     +X[[2]][i,] %*% beta.val[[2]]*(max(t.x.2)-min(t.x.2))/length(t.x.2)+ E[i,]}))
 Y=Y+Z%*%beta.scalar
 X.train=lapply(1:n.curves, function(j){X[[j]][1:ntrain,]})
 X.test=lapply(1:n.curves, function(j){X[[j]][-(1:ntrain),]})
 t.x.list=list()
 t.x.list[[1]]=t.x.1
 t.x.list[[2]]=t.x.2
 Z.train<- Z[1:ntrain, ]
 Z.test <- Z[-(1:ntrain), ]
 Y.train <- Y[1:ntrain, ]
 Y.test <- Y[-(1:ntrain), ]
 E.test=E[-(1:ntrain), ]
 fit.cv=cv.sigcom(X.train, Y.train, t.x.list, t.y, Z.train, s.n.basis=40,
                  t.n.basis=40)
 Y.pred=pred.sigcom(fit.cv, X.test, Z.test)
 error<- mean((Y.pred-Y.test)^2)
 print(c(" prediction error=", error))
 est.error<- mean((Y.pred-Y.test+E.test)^2)
 print(c("estimation error for regression function (or signal function)=", est.error))
 coef.est.all=getcoef.sigcom(fit.cv)
 mu.est=coef.est.all[[1]]
 beta.est=coef.est.all[[2]]
 beta.scalar.est=coef.est.all[[3]]
 print(proc.time()-ptm)
#   user  system elapsed
#   2.90    0.69    3.96


#####################################################################
#Example 3: application to the DTI data in 'refund' package
#####################################################


ptm <- proc.time()
library(refund)
data(DTI)
I=which(is.na(apply(DTI$cca,1,mean)))
Y=DTI$cca[-I,]
X=list(DTI$rcst[-I,-(1:12)])
ntot=dim(Y)[1]

lengthX=dim(X[[1]])[2]
lengthY=dim(Y)[2]
```

```
t.x <- list(seq(0,1,length=lengthX))
t.y <- seq(0,1,length=lengthY)
# randomly split all the observations into a training set with 200 observations
# and a test set.
ntrain=200
I=sample(1:ntot, ntrain)
X.train <- list(X[[1]][I,])
Y.train <- Y[I, ]
X.test <- list(X[[1]][-I,])
Y.test <- Y[-I, ]
fit.cv=cv.sigcom(X.train, Y.train, t.x, t.y, s.n.basis=40, t.n.basis=40)
Y.pred=pred.sigcom(fit.cv, X.test)
error<- mean((Y.pred-Y.test)^2)
print(c(" prediction error=", error))
coef.est.all=getcoef.sigcom(fit.cv)
mu.est=coef.est.all[[1]]
beta.est=coef.est.all[[2]]
print(proc.time()-ptm)
#   user  system elapsed
#   6.09    0.10    6.17
```

`getcoef.ff.interaction` *Get the estimated coefficient functions for function-on-function interaction model*

Description

Calculate the estimates for $\mu(t), \beta_i(s,t), \gamma_{ij}(u,v,t)$ for function-on-function interaction model (see the model in `cv.ff.interaction`) based on the output object of the function `cv.ff.interaction`, or the function `step.ff.interaction`.

Usage

```
getcoef.ff.interaction(fit.obj)
```

Arguments

`fit.obj`	the output object of the function `cv.ff.interaction`, or `step.ff.interaction`.

Value

a list providing the given or selected main effects and interactions, together with the corresponding estimated coefficient functions.

`intercept`	the vector of estimated $\mu(t)$ evaluated at the vector `t.y` of the observation points for the response function $y(t)$.

main_effects	the index vector of the input main_effects for cv.ff.interaction or the selected main effects by step.ff.interaction.
coef_main	a list of matrices of the estimated values of the coefficient functions of main effect specified by main_effects. Each matrix gives the estimated values of $\beta_i(s,t)$ at the two-dimensional grid created by the observation point vectors t.x.list[[i]] and t.y, where i is an index in main_effects.
inter_effects	a matrix of two columns showing the input interactions for cv.ff.interaction or the selected interactions by step.ff.interaction. Each row shows the indices of the pair of functional variables in an interaction or quadratic effect.
coef_inter	a list of three-dimensional arrays of estimated values of the coefficient functions of interaction or quadratic effects specified by inter_effects. Each array gives the estimated values of $\gamma_{ij}(u,v,t)$ at the three-dimensional grid created by the observation point vectors t.x.list[[i]], t.x.list[[j]] and t.y, where the pair i,j is in inter_effects.

Author(s)

Xin Qi and Ruiyan Luo

See Also

cv.ff.interaction, step.ff.interaction.

Examples

```
#See the examples in cv.ff.interaction() and step.ff.interaction().
```

---

getcoef.hd *Get the estimated intercept and coefficient functions for sparse linear FOF models*

Description

Calculate the estimates for $\mu(t), \beta_i(s,t)$ based on the object obtained from cv.hd.

Usage

```
getcoef.hd(fit.cv)
```

Arguments

fit.cv	the object obtained from cv.hd.

Value

a list containing

mu	the vector of estimated values for $\mu(t)$ at t.y, the vector of observation time points for the response function.
beta	a list of length p, the number of functional predictors. Its i-th element is a matrix of the estimated values of $\beta_i(s,t)$ at the full grid of observaion time points created by arguments t.x.list[[i]] and t.y for cv.hd.

Author(s)

Ruiyan Luo and Xin Qi

See Also

cv.hd

Examples

```
#See the examples in cv.hd().
```

---

getcoef.nonlinear *Get the estimated intercept and nonlinear functions*

Description

Calculate the estimates for $\mu(t), F_i(x,s,t)$ based on the object obtained from cv.nonlinear.

Usage

```
getcoef.nonlinear(fit.cv, n.x.grid = 100)
```

Arguments

fit.cv	the object obtained from cv.nonlinear.
n.x.grid	the number of grid points of x. The estimated $F_i(x,s,t)$ is calculated in a three-dimensional grid of (x,s,t). The grid points of s and t are the observation points of $X_i(s)$ and $Y(t)$ used in cv.nonlinear, respectively. The grid of x includes n.x.grid equally spaced values between the minimum and maximum of all the discretely observed values of $X_i(s)$. Default of n.x.grid is 100.

Value

a list containing

`mu`	the vector of estimated values of $\mu(t)$ at the observation points of the response function.
`F`	a list of length p, the number of functional predictors. Its i-th element is a three dimensional array with estimated values of $F_i(x, s, t)$ on the three-dimensional grid `X.grid[[i]]*t.x.list[[i]]*t.y` (see below).
`X.grid`	a list of length p. Its i-th element is the vector of grid points for x and includes `n.x.grid` equally spaced values between the minimum and maximum of all the discretely observed values of $X_i(s)$.
`t.x.list`	one of the arguments in `cv.nonlinear`, specifying the list of the vectors of obesrvation points for $X_i(s)$, $1 \leq i \leq p$.
`t.y`	one of the arguments in `cv.nonlinear`, specifying the vector of obesrvation points of the response curve $Y(t)$.

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo and Xin Qi,

See Also

`cv.nonlinear`.

Examples

```
#See the examples in cv.nonlinear().
```

`getcoef.sigcom` *Get the estimated intercept and coefficient functions for linear FOF models*

Description

Calculate the estimates for $\mu(t), \alpha(t), \beta_i(s, t)$ based on the object obtained from `cv.sigcom`.

Usage

```
getcoef.sigcom(fit.cv)
```

Arguments

fit.cv	the object obtained from cv.sigcom.

Value

a list containing

mu	the vector of the estimated values of $\mu(t)$ at the grid of observation points of the response function ($t.y$ for cv.sigcom).
beta	a list of length p, the number of functional predictors. Its i-th component is a matrix of the estimated values of coefficient functions $\beta_i(s,t)$ of functional predictors at the full grid of observaion time points created by arguments t.x.list[[i]] and t.y for cv.sigcom. The columns correspond to different observation points for response varible ($t.y$).
beta.scalar	the matrix of estimated values of the coefficient functions $\alpha(t)$ of scalar predictors, with rows corresponding to different scalar predictors, and columns corresponding to different observation points of the functional response.

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo and Xin Qi, (2017) Function-on-Function Linear Regression by Signal Compression, Journal of the American Statistical Association. http://www.tandfonline.com/doi/abs/10.1080/01621459.2016.1164053

See Also

cv.sigcom

Examples

```
#See the examples in cv.sigcom().
```

---

ocean *Hawaii ocean data*

Description

Five variables, Salinity, Potential Density, Temperature, Oxygen and Chloropigment, were measured every two meters between 0 and 200 meters below the sea surface at a site located 100 kilometers north of Oahu, Hawaii in 116 different days.

Usage

```
data("ocean")
```

Format

A list of 5 matrices:

Salinity a matrix with 116 rows and 101 columns.

Potential.density a matrix with 116 rows and 101 columns.

Temperature a matrix with 116 rows and 101 columns.

Oxygen a matrix with 116 rows and 101 columns.

Chloropigment a matrix with 116 rows and 101 columns.

Source

Hawaii Ocean Time-series

See Also

`cv.ff.interaction`

Examples

```
data(ocean)
str(ocean)
```

---

pred.ff.interaction *Prediction for a linear FOF regression model with two-way interactions*

Description

Make predition for new observations of functional predictors based on the fitted function-on-function interaction model in the output of `cv.ff.interaction` with given main and two-way interaction effects, or the model in the output of `step.ff.interaction` with selected main and two-way interaction effects.

Usage

```
pred.ff.interaction(fit.obj, X.test)
```

Arguments

`fit.obj`	the output object of the function `cv.ff.interaction`, or `step.ff.interaction`.
`X.test`	new observations of functional predictors. It is a list of the same length and the same structure as the input `X` for `cv.ff.interaction` or `step.ff.interaction`.

Value

A matrix containing the predicted values of response curves evaluated at `t.y` for the new observations. The number of rows of the matrix equals to the sample size of the new data set, and the number of columns equals to the length of `t.y`, the input in `cv.ff.interaction` or `step.ff.interaction`.

Author(s)

Xin Qi and Ruiyan Luo

See Also

cv.ff.interaction, step.ff.interaction

Examples

```
#See the examples in cv.ff.interaction() and step.ff.interaction().
```

---

pred.hd *Prediction for sparse linear function-on-function regression*

Description

Make predition for functional response from the CV object obtained by cv.hd.

Usage

```
pred.hd(fit.cv, X.test, t.y.test=NULL)
```

Arguments

fit.cv	the CV object obtained by cv.hd.
X.test	new observations of functional predictors. It is a list of length p, the number of functional predcitors. Each element is the observed matrix from a functional predictor, with rows repsenting observation vectors and columns corresponding to the observation time points.
t.y.test	a vector of observation time points where values of predicted response curves are to be calculated. If t.y.test=NULL (default), t.y in cv.hd will be used.

Value

A matrix containing the predicted values of response curves for the new observations. The number of rows equals to the sample size of the new data set, and the number of columns equals to the length of t.y.test or t.y when t.y.test=NULL.

Author(s)

Ruiyan Luo and Xin Qi

See Also

cv.hd

Examples

```
#See the examples in cv.hd().
```

`pred.nonlinear` *Prediction for nonlinear function-on-function regression*

Description

Make predition for response based on new observations of predictor curves from the CV object obtained by `cv.nonlinear`.

Usage

```
pred.nonlinear(fit.cv, X.test, t.y.test=NULL)
```

Arguments

`fit.cv`	the CV object obtained by `cv.nonlinear`.
`X.test`	new observations of functional predictors. It is a list of length p, the number of functional predcitors. Each element is the observed matrix from a functional predictor, with rows repsenting observation vectors and columns corresponding to the observation time points.
`t.y.test`	a vector of observation time points where values of predicted response curves are to be calculated. If `t.y.test`=NULL (default), `t.y` in `cv.nonlinear` will be used.

Value

A matrix which contains the predicted values of response curves. The number of rows equal to the sample size of the new data set, and the number of columns is equal to the length of `t.y.test` or `t.y` when `t.y.test`=NULL.

Author(s)

Ruiyan Luo and Xin Qi

See Also

`cv.nonlinear`

Examples

```
#See the examples in cv.nonlinear().
```

pred.sigcom *Prediction for linear function-on-function regression using signal compression*

Description

Make predition for functional response from the CV object obtained by cv.sigcom.

Usage

```
pred.sigcom(fit.cv, X.test, Z.test = NULL, t.y.test=NULL)
```

Arguments

fit.cv	the CV object obtained by cv.sigcom.
X.test	new observations for the functional predictors. It is a list of length p, the number of functional predcitors. Each element is the observed matrix from a functional predictor, with rows repsenting observation vectors and columns corresponding to the observation time points.
Z.test	new observations for the scalar predictors. It is a matrix with rows representing observation vectors and columns respresenting scalar variables. Default is NULL, indicating no scalar predictors.
t.y.test	a vector of observation time points where values of predicted response curves are to be calculated. If t.y.test=NULL (default), t.y in cv.sigcom will be used.

Value

A matrix containing the predicted response for the new observations. The number of rows is equal to the sample size of the new data set, and the number of columns is equal to the length of t.y.test or t.y when t.y.test=NULL.

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo and Xin Qi, (2017) Function-on-Function Linear Regression by Signal Compression, Journal of the American Statistical Association. 112(518), 690-705. http://www.tandfonline.com/doi/abs/10.1080/01621459.2016.1164053

See Also

cv.sigcom

Examples

```
#See the examples in cv.sigcom().
```

`step.ff.interaction` *Stepwise variable selection procedure for FOF regression models with two-way interactions*

Description

This function conducts the stepwise procedure to select main effects, two-way interaction and quadratic effects for the following family of linear function-on-function interaction models. Let $\{X_i(s), 1 \leq i \leq p\}$ be p potential functional predictors. The family of models is given by

$$Y(t) = \mu(t) + \sum_{i \in M} \int_{a_i}^{b_i} X_i(s)\beta_i(s,t)ds + \sum_{(i,j) \in I} \int_{a_i}^{b_i} \int_{a_j}^{b_j} X_i(u)X_j(v)\gamma_{ij}(u,v,t)dudv + \epsilon(t)$$

where $\mu(t)$ is the intercept function. The index set M of the main effects is a subset of $\{1, ..., p\}$, and the index set I of the interactions and quadratic effects is a subset of the collection of all possible pairs $\{(i,j), 1 \leq i \leq j \leq p\}$. We require that the models in each step satisfy the hierarchy principle: if the interaction X_iX_j is included in the model, both the main effects X_i and X_j are included. The $\{\beta_i(s,t), i \in M\}$ and $\{\gamma_{ij}(u,v,t), (i,j) \in I\}$ are the corresponding coefficient functions. The $\epsilon(t)$ is the noise function. When the final model is selected, this function also fits the selected model.

Usage

```
step.ff.interaction(X, Y, t.x, t.y, s.n.basis=40, t.n.basis=40, inter.n.basis=20,
       basis.type="Bspline",  K.cv=5, all.folds=NULL, upper.comp=8, thresh=0.01)
```

Arguments

`X`	a list of p potential functional predictors. Its i-th element is the $n \times m_i$ data matrix for the i-th potential functional predictor $X_i(s)$, where n is the sample size and m_i is the number of observation time points for $X_i(s)$.
`Y`	the $n \times m$ data matrix for the functional response $Y(t)$, where n is the sample size and m is the number of the observation time points for $Y(t)$.
`t.x`	a list of length p. Its i-th element is the vector of obesrvation time points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.
`t.y`	the vector of observation time points of the functional response $Y(t)$.
`s.n.basis`	the number of basis functions used for estimating the functions $\psi_{ik}(s)$ (see the reference for details). Default is 40.
`t.n.basis`	the number of basis functions used for estimating the functions $w_k(t)$. Default is 40.
`inter.n.basis`	the number of one-dimensional basis functions used to construct the tensor product basis functions for estimating the functions $\phi_{ijk}(u,v)$. Default is 20.

`basis.type` the type of basis functions. Only "BSpline" (default) and "Fourier" are supported.

`K.cv` the number of CV folds. Default is 5.

`all.folds` a list of cross validation sets. The length of the list is equal to `K.cv`. The i-th element of the list is the collection of indices of observations assigned to the i-th cross validation set. The Default is NULL, and the cross validation sets are generated by this function `step.ff.interaction`.

`upper.comp` the upper bound for the maximum number of components to be calculated. Default is 10.

`thresh` a number between 0 and 1 specifying the minimum proportion of variation to be explained by each selected component relative to all the selected components. This will help determining the upper bound of the number of components to be used in CV, and the optimal number of components will be determined from 1,2,..., to the minimum of this uppber bound and upper.com by CV. A smaller thresh value leads to more components to be selected and a longer running time. A larger thresh value needs less running time, but may miss some important components and lead to a larger prediction error. Default is 0.01.

Value

An object of the "step.ff.interaction" class, which is used in the function `pred.ff.interaction` for prediction and `getcoef.ff.interaction` for extracting the estimated coefficient functions.

`opt.main.effects`
: a vector of indices of the selected main effects.

`opt.interaction.effects`
: a matrix of two columns. Each row specifies the indices of a selected two-way interaction or quadratic effect.

`fitted_model` a list of the output of the fitted selected model and only for internal use.

`y_penalty_inv` a list for interval use.

`X` the input `X`.

`Y` the input `Y`.

`x.smooth.params`
: a list for internal use.

`y.smooth.params`
: a list for internal use.

Author(s)

Xin Qi and Ruiyan Luo

References

Ruiyan Luo and Xin Qi (under revision) Interaction model and model selection for function-on-function regression.

Examples

```
library(FRegSigCom)
data(ocean)

Y=ocean$Salinity
X=list()
X[[1]]=ocean$Potential.density
X[[2]]=ocean$Temperature
X[[3]]=ocean$Oxygen
n.curves=length(X)
ntot=dim(Y)[1]
ntrain=50
ntest=ntot-ntrain
X.uncent=X
for(i in 1:n.curves){
  X[[i]]=scale(X.uncent[[i]],center=TRUE, scale=FALSE)
}
lengthX=dim(X[[1]])[2]
lengthY=dim(Y)[2]
t.x=seq(0,1,length=lengthX)
t.y=seq(0,1,length=lengthY)
I.train=sample(1:ntot, ntrain)
X.train=list()
X.test=list()
t.x.all=list()
for(j in 1:n.curves){
  X.train[[j]]=X[[j]][I.train,]
  X.test[[j]]=X[[j]][-I.train,]
  t.x.all[[j]]=t.x
}
Y.train=Y[I.train, ]
Y.test=Y[-I.train, ]


################################
#model selection
################################

fit.step=step.ff.interaction(X.train, Y.train, t.x.all, t.y)
Y.pred=pred.ff.interaction(fit.step,  X.test)
error.selected=mean((Y.pred-Y.test)^2)
print(c("error.selected=", error.selected))
#coef.obj=getcoef.ff.interaction(fit.step)
#str(coef.obj)
```

Index