Package 'FRegSigCom'

November 15, 2018

Type Package

Title Functional Regression using Signal Compression Approach

Version 0.3.0

Author Ruiyan Luo, Xin Qi

Maintainer Ruiyan Luo <rluo@gsu.edu>

Description Signal compression methods for functional regression. It includes various function-on-function (FOF) regression models such as the linear FOF model with functional response and both scalar and functional predictors for a small number of functional predictors, linear FOF models with a large number of functional predictors, linear FOF model for spiky data, stepwise selection for FOF models with two-way interactions, and nonlinear FOF models. It also includes scalar-on-function regression models with single (SOF) or multivariate (mSOF) scalar response variable, and SOF model for spiky data.

URL http://sites.gsu.edu/rluo/, http://sites.gsu.edu/xqi3/

License GPL-2

Encoding UTF-8

LazyData TRUE

NeedsCompilation yes

Imports Rcpp

LinkingTo Rcpp, RcppEigen

Depends R (>= 2.10), fda, Matrix,

Suggests refund, MASS, wavethresh

Collate 'ffsig.R' 'ff.wave.sigcom.R' 'ff.spike.R' 'ff.nonlinear.R' 'ff.interaction.R' 'high_dimensional_function_on_function.R' 'hd_multi_scalar_on_function.R' 'multi_scalar_on_function.R' 'spiky.scalar.on.function.R' 'RcppExports.R'

RoxygenNote 6.1.0

Repository CRAN

Date/Publication 2018-11-15 08:20:18 UTC

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)
```

---

corn *NIR of corn samples*

Description

This data set consists of measurements on 80 corn samples. For each sample, its moisture, oil, protein and starch values, together with its near-infrared (NIR) spectrum curve, were measured. The wavelength range of the NIR curve is from 1100nm to 2498nm, and the curve was measured at every 2 nm.

Usage

```
data(corn)
```

Format

A list of 2 matrices:

X an 80*700 matrix with each row giving the NIR spectrum of a corn sample.

Y an 80*4 matrix with the 4 columns giving the moisture, oil, protein and starch values, respectively.

Source

Data Link

See Also

`cv.msof`

Examples

```
data(corn)
str(corn)
```

`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 effects and quadratic effects. 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,
       adaptive=FALSE, s.n.basis=40, t.n.basis=40, inter.n.basis=20,
       basis.type.x="Bspline", basis.type.y="Bspline", K.cv=5,
       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 observation 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 $Y(t)$.
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.
adaptive	a logic value indicating whether using adaptive penalty that has different smoothness tuning parameters for different target functions (see Details). Default is FALSE.
s.n.basis	the number of basis functions used for estimating the functions $\psi_{ik}(s)$ (see 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.x	the type of basis functions $\psi_{ik}(s)$. Only "BSpline" (default) and "Fourier" are supported.
basis.type.y	the type of basis functions $w_k(t)$. Only "BSpline" (default) and "Fourier" are supported.
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 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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.

Details

This method uses the decomposition of the coefficient functions

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

and

$$\gamma_{ij}(u,v,t) = \sum_{k=1}^{\infty} \phi_{ij,k}(s) w_k(t), (i,j) \in I$$

where for each $k > 0$, $\{\{\psi_{ik}(s), i \in M\}, \{\phi_{ij,k}(s), (i,j) \in I\}\}$ are estimated by solving a generalized functional eigenvalue problem with the nonadaptive penalty

$$\lambda \sum_{i \in M} \{||\psi_{ik}||^2 + \tau ||\psi''_{ik}||^2\} + \lambda \sum_{(i,j) \in I} \{||\phi_{ij,k}||^2 + \tau(||\partial_{uu}\psi_{ij,k}||^2 + ||\partial_{uv}\psi_{ij,k}||^2 + ||\partial_{vv}\psi_{ij,k}||^2)\}$$

or the adaptive penalty

$$\lambda \sum_{i \in M} \{\omega_{ik}^{(0)} ||\psi_{ik}||^2 + \tau \omega_{ik}^{(2)} ||\psi''_{ik}||^2\} +$$

$$+\lambda \sum_{(i,j) \in I} \{\omega_{ij,k}^{(00)} ||\phi_{ij,k}||^2 + \tau(\omega_{ij,k}^{(20)} ||\partial_{uu}\psi_{ij,k}||^2 + \omega_{ij,k}^{(11)} ||\partial_{uv}\psi_{ij,k}||^2 + \omega_{ij,k}^{(02)} ||\partial_{vv}\psi_{ij,k}||^2)\}$$

and then $\{w_k(t), k > 0\}$ are estimate by regressing $Y(t)$ on $\{\hat{z}_1, ...\hat{z}_K\}$ with nonadaptive penalty $\kappa \sum_{k=0}^{K} \|w''_k\|^2$ or adaptive penalty $\kappa \sum_{k=0}^{K} \omega_k^{(t)} \|w''_k\|^2$ tuned by the smoothness parameter κ. Here $\hat{z}_k = \sum_{i \in M} \int_{a_i}^{b_i} X_i(s)\hat{\psi}_{ik}(s)ds + \sum_{(i,j) \in I} \int_{a_i}^{b_i} \int_{a_j}^{b_j} X_i(u)X_j(v)\hat{\phi}_{ij,k}(u,v)dudv$ and then centered around its sample mean.

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.
`basis.types`	a vector including `basis.type.x` and `basis.type.y`.

Author(s)

Xin Qi and Ruiyan Luo

References

Ruiyan Luo and Xin Qi (2018) Interaction model and model selection for function-on-function regression, Journal of Computational and Graphical Statistics. https://doi.org/10.1080/10618600.2018.1514310

See Also

`pred.ff.interaction`, `getcoef.ff.interaction`

Examples

```
##############################################################################
# Example: interaction function-on-function model with
#          specified effects
##############################################################################


ptm <- proc.time()
library(FRegSigCom)
data(ocean)
Y=ocean[[1]]
Y.train=Y[1:50,]
Y.test=Y[-(1:50),]
t.y=seq(0,1, length.out = ncol(Y))
X.list=list()
X.train.list=list()
X.test.list=list()
t.x.list=list()
for(i in 1:4)
{
  X.list[[i]]=ocean[[i+1]]
  X.train.list[[i]]=X.list[[i]][1:50,]
  X.test.list[[i]]=X.list[[i]][-(1:50),]
  t.x.list[[i]]=seq(0,1, length.out = ncol(X.list[[i]]))
}
main.effect=1:2
inter.effect=rbind(c(1,1), c(1,2), c(2,2))
fit.fix.adaptive=cv.ff.interaction(X.train.list, Y.train, t.x.list, t.y,
           adaptive=TRUE, main.effect, inter.effect)
Y.pred=pred.ff.interaction(fit.fix.adaptive,  X.test.list)

error<- mean((Y.pred-Y.test)^2)
print(c(" prediction error=", error))
print(proc.time()-ptm)
```

---

cv.fof.spike *Cross-validation for linear function-on-function regression on highly densely observed spiky data*

Description

This function is used to perform cross-validation and build the final model for highly densely observed spiky data using the signal compression approach for the following linear 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 their corresponding coefficient functions, where p is a positive integer. 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.

Usage

```
cv.fof.spike(X, Y, t.x, t.y, K.cv = 5, upper.comp = 10, thresh = 0.0001)
```

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 observation 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 $Y(t)$.
`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 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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 0.0001.

Details

We use the decomposition of the coefficient function:

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

and estimate $\{\psi_{ik}(s), 1 \leq i \leq p\}$ for each $k > 0$ by solving a generalized functional eigenvalue problem as given in `cv.sigcom` but with penalty replaced by

$$\lambda \sum_{i=1}^{p} \sum_{j=0}^{J_1} \{e^{\tau(j-J_1)} 2^{2\alpha j} ||b_{ik,j}||^2 + e^{-\tau J_1} ||a_{ik,0}||^2\},$$

where $a_{ik,0}$ and $b_{ik,j}$ $(0 \leq j \leq J_1)$ are vectors of wavelet coefficients for ψ_{ik}. Then we estimate $\{w_k(t), k > 0\}$ by regressing $Y(t)$ on $\{\hat{z}_1, ...\hat{z}_K\}$ using penalized least square method, where $\hat{z}_k = \sum_{i=1}^{p} \int \{X_i(s) - \bar{X}_i(s)\} \hat{\psi}_{ik}(s) ds$.

Value

An object of the "cv.fof.spike" class, which is used in the function `pred.fof.spike` for prediction.

`mu`	a vector containing the estimated values of the intercept function at `t.y`.
`Beta`	a list of p matrices, where the i-th matrix contains the estimated values of the slope coefficient function $beta_i(s,t)$ at grid `t.x` * `t.y`.
`c_fit_cv`	a list for internal use.

Author(s)

Xin Qi and Ruiyan Luo

References

Xin Qi and Ruiyan Luo, (manuscript) Functional regression for highly densely observed functional data with novel regularity.

Examples

```
##############################################################################
# Example: spiky function-on-function regresion
##############################################################################
ptm <- proc.time()
library(FRegSigCom)
library(refund)
data(DTI)
I=which(is.na(apply(DTI$cca,1,mean)))
X=DTI$cca[-I,] # functional response
Y=DTI$rcst[-I,-(1:12)] #functional predictor
t.x <- list(seq(0,1,length=dim(X)[2]))
t.y <- seq(0,1,length=dim(Y)[2])
# randomly split all the observations into a training set with 200 observations
# and a test set.
train.id=sample(1:nrow(Y), 30)
X.train <- list(X[train.id,])
Y.train <- Y[train.id, ]
X.test <- list(X[-(train.id),])
Y.test <- Y[-(train.id), ]
fit.cv=cv.fof.spike(X.train, Y.train, t.x, t.y)
Y.pred=pred.fof.spike(fit.cv, X.test)
error<- mean((Y.pred-Y.test)^2)
print(c("prediction error=", error))

print(proc.time()-ptm)
```

`cv.fof.wv` *Cross-validation for wavelet-based linear function-on-function regression method*

Description

This function performs cross-validation and builds the final model for the following linear function-on-function regression model:

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

where $\mu(t)$ is the intercept function. The $\{Z_i(s), 1 \leq i \leq p\}$ 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. The $\epsilon(t)$ is the noise function. This method first applies the fast wavelet transformation (FWT) to the predictor curves, and transforms the original model to a function-on-scalar model with wavelet coefficients as scalar predictors. Then applying a dimension reduction based on signal compression to approximate the regression function, we have a function-on-scalar model with a small number of uncorrelated scalar predictors.

Usage

```
cv.fof.wv(X, Y, t.y, K.cv = 5, upp.comp=10, thresh=0.01)
```

Arguments

`X`	the $n \times p$ matrx of the wavelet coefficients of the predictor curves, where n is the sample size and q is the total number of wavelet coefficients for original functional predictors.
`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.y`	the vector of obesrvation time points of the functional response $Y(t)$.
`K.cv`	the number of CV folds. Default is 5.
`upp.comp`	the upper bound for the maximum number of components to be calculated. Default is 10.
`thresh`	a number between 0 and 1 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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.

Details

This method first expresses the functional predictors and $\beta_i(.,t)$ using wavelet expansion. Let $\boldsymbol{X}$ and $\boldsymbol{\beta}(t)$ denote the vectors of all concatenated wavelet coefficients. Then the original model is transformed to

$$Y(t) = \mu(t) + \boldsymbol{X}^T\boldsymbol{\beta}(t) + \varepsilon(t).$$

We use the decomposition $\boldsymbol{\beta}(t) = \sum_{k=1}^{K} \boldsymbol{\alpha}_k w_k(t)$ based on the KL expansion of $\boldsymbol{X}^T\boldsymbol{\beta}(t)$, where $\boldsymbol{\alpha}_k$'s are vectors of the same length as $\boldsymbol{\beta}(t)$. We estimate $\boldsymbol{\alpha}_k$ for each k by solving the panelized generalized eigenvalue problem

$$max_{\boldsymbol{\alpha}} \frac{\boldsymbol{\alpha}^T\hat{\boldsymbol{B}}\boldsymbol{\alpha}}{\boldsymbol{\alpha}\hat{\boldsymbol{\Sigma}}\boldsymbol{\alpha} + P(\boldsymbol{\alpha})}$$

$$\text{s.t.} \quad \boldsymbol{\alpha}\hat{\boldsymbol{\Sigma}}\boldsymbol{\alpha} = 1$$

$$\text{and} \quad \boldsymbol{\alpha}\hat{\boldsymbol{\Sigma}}\boldsymbol{\alpha}_{k'} = 0 \quad \text{for} \quad k' < k$$

where $\hat{\boldsymbol{B}} = \sum_{\ell=1}^{n}\sum_{\ell'=1}^{n}\{x_\ell - \bar{x}\}\int\{y_\ell(t) - \bar{y}(t)\}\{y_{\ell'}(t) - \bar{y}(t)\}dt\{x_{\ell'} - \bar{x}\}^T/n^2$, $\hat{\boldsymbol{\Sigma}} = \sum_{\ell=1}^{n}\{x_\ell - \bar{x}\}\{x_\ell - \bar{x}\}^T/n$, and penalty

$$P(\boldsymbol{\alpha}) = \tau\{(1-\lambda)||\boldsymbol{\alpha}||_2^2 + \lambda||\boldsymbol{\alpha}||_1^2\}.$$

Then we estimate $w_k(t), k > 0$ by regressing $Y(t)$ on $\{\hat{z}_1, ...\hat{z}_K\}$ with penalty $\kappa\sum_{k=0}^{K}\|w_k''\|^2$ tuned by the smoothness parameter κ. Here $\hat{z}_k = (\boldsymbol{X} - \bar{\boldsymbol{X}})^T\hat{\boldsymbol{\alpha}}_k$.

Value

An object of the "cv.fof.wv" class, which is used in the function `pred.fof.wv` for prediction.

`min.error`	minimum CV error.
`X`	input X.
`Y`	input Y.
`errors`	list for CV errors.
`opt.K`	optimal number of components to be selected.
`opt.smooth`	optimal smooth tuning parameter κ.
`min.tau`	optimal tuning parameter τ.
`min.lambda`	optimal tuning parameter λ.
`params.set`	set of tuning parameters using in CV.
`...`	other output for internal use.

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo, Xin Qi and Yanhong Wang. (2016) Functional wavelet regression for function-on-function linear models. Electronic Journal of Statistics. 10(2): 3179-3216.

Examples

```
#############################################################################
## Example: wavelet function-on-function regresion
#############################################################################
ptm <- proc.time()
library(FRegSigCom)
library(wavethresh)
library(refund)
data(DTI)

I=which(is.na(apply(DTI$cca,1,mean)))
Y=DTI$cca[-I,] # functional response
X=DTI$rcst[-I,21:52] #functional predictor
n.wv=5

diagmat <- diag(2^n.wv)
W.x <- diagmat
for(i in 1:2^n.wv){
  tmp <- wd(diagmat[i,])
  tmp.cof <- accessC(tmp, level=0)
  for(j in 0:(n.wv-1))
    tmp.cof <- c(tmp.cof, accessD(tmp, level=j))
  W.x[,i] <- tmp.cof
}
X.wv=X

t.y <- seq(0,1,length=dim(Y)[2])
# randomly split all the observations into a training set with 200 observations
# and a test set.
train.id=sample(1:nrow(Y), 50)
X.wv.train <- X.wv[train.id,]
Y.train <- Y[train.id, ]
X.wv.test <- X.wv[-(train.id),]
Y.test <- Y[-(train.id), ]

fit.cv=cv.fof.wv(X.wv.train, Y.train, t.y, upp.comp=5) # use default upp.comp or larger
Y.pred=pred.fof.wv(fit.cv, X.wv.test)
error<- mean((Y.pred-Y.test)^2)
print(c(" prediction error=", error))
print(proc.time()-ptm)
```

---

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.

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 functional 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 observation 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.

Details

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

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

We first estimate $\boldsymbol{\psi}_k = (\psi_{1k}(s), ..., \psi_{pk}(s))$ for each $0 < k < K$, by solving the generalied functional eigenvalue problem

$$max_{\boldsymbol{\psi}} \frac{\int\int \boldsymbol{\psi}(s)^T \hat{B}(s,s') \boldsymbol{\psi}(s') ds ds'}{\int\int \boldsymbol{\psi}(s)^T \hat{\Sigma}(s,s') \boldsymbol{\psi}(s') ds ds' + P(\boldsymbol{\psi})}$$

$$\text{s.t.} \quad \int\int \boldsymbol{\psi}(s)^T \hat{\Sigma}(s,s') \boldsymbol{\psi}(s') ds ds' = 1$$

$$\text{and} \quad \int\int \boldsymbol{\psi}(s)^T \hat{\Sigma}(s,s') \boldsymbol{\psi}_{k'}(s') ds ds' = 0 \quad \text{for} \quad k' < k$$

with the simultaneous sparse and smooth penalty

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

where $||\psi_i||_\eta^2 = ||\psi_i||^2 + \eta||\psi_i''||^2$. Here $\hat{B}$ and $\hat{\Sigma}$ are $p * p$ matrices of functions with the (i,j) element respectively as $\hat{B}_{ij}(s,s') = \sum_{\ell=1}^{n}\sum_{\ell'=1}^{n}\{x_{\ell,i}(s) - \bar{x}_i(s)\}\int\{y_\ell(t) - \bar{y}(t)\}\{y_{\ell'}(t) - \bar{y}(t)\}dt\{x_{\ell',j}(s') - \bar{x}_j(s')\}/n^2$ and $\hat{\Sigma}_{ij}(s,s') = \sum_{\ell=1}^{n}\{x_{\ell,i}(s) - \bar{x}_i(s)\}\{x_{\ell,j}(s') - \bar{x}_j(s')\}/n$, where $(x_{\ell,1}, ..., x_{\ell,p}, y_\ell)$ represents the ℓ-th sample. Then we estimate $\{w_k(t), 0 < k < K\}$ by regressing $\{y_\ell(t)\}$ on $\{\hat{z}_{\ell,1}, ...\hat{z}_{\ell,K}\}$ with penalty $\kappa\sum_{k=0}^{K}\|w_k''\|^2$ tuned by the smoothness parameter κ. Here $\hat{z}_{\ell,k} = \sum_{i=1}^{p}\int(x_{\ell,i}(s) - \bar{x}_i(s))\hat{\psi}_{ik}(s)ds$. The number of selected componets K will be chosen by cross-validation.

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 $\{\psi_{ik}(s), 1 \le i \le p\}$.
`opt.lambda`	optimal value for λ, the sparsity parameter for $\{\psi_{ik}(s), 1 \le i \le p\}$.
`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.x.list	the input t.x.list.
t.y	the input t.y.
y.params	a list for internal use.

Author(s)

Xin Qi and Ruiyan Luo

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.msof *Cross-validation for linear multivariate scalar-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 multivariate scalar-on-function regression model:

$$\boldsymbol{Y} = \boldsymbol{\mu} + \sum_{i=1}^{p} \int_{a_i}^{b_i} X_i(s)\boldsymbol{\beta}_i(s)ds + \boldsymbol{\epsilon},$$

where $\boldsymbol{Y}$ is an m-dimensional multivariate response variable, $\boldsymbol{\mu}$ is the m-dimensional intercept vector. The $\{X_i(s), 1 \leq i \leq p\}$ are p functional predictors and $\{\boldsymbol{\beta}_i(s), 1 \leq i \leq p\}$ are their corresponding m-dimensional vector of coefficient functions, where p is a positive integer. The $\boldsymbol{\epsilon}$ is the random noise vector.

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.

Usage

```
cv.msof(X, Y, t.x.list, nbasis = 50, K.cv = 5, upper.comp = 10,
        thresh = 0.001)
```

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`	an $n \times q$ data matrix for the response Y or a n-dimensional vector if there is only one scalar response, where n is the sample size, and q is the number of scalar response variables.
`t.x.list`	a list of length p. Its i-th element is the vector of observation time points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.
`nbasis`	the number of basis functions used for estimating the vector of functions $\psi_{ik}(s)$ (see the reference for details). 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 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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 0.001.

Details

We use the decomposition $\boldsymbol{\beta}_i(s) = \sum_{k=1}^{K} \alpha_{ki}(s)\boldsymbol{w}_k$, $1 \leq i \leq p$, based on the KL expansion of $\sum_{i=1}^{p} \int X_i(s)\boldsymbol{\beta}_i(s)ds$. Let $\boldsymbol{Y}_\ell = (Y_{\ell,1}, ..., Y_{\ell,m})^T$ and $\boldsymbol{X}_\ell(s) = (X_{\ell,1}(s), ..., X_{\ell,p}(s))^T$, $1 \leq \ell \leq n$, denote n independent samples. We estimate $\boldsymbol{\alpha}_k(s) = (\alpha_{k1}(s), ..., \alpha_{kp}(s))^T$ for each k by solving the panelized generalized functional eigenvalue problem

$$max_\alpha \frac{\int\int \boldsymbol{\alpha}(s)^T \hat{\boldsymbol{B}}(s,s')\boldsymbol{\alpha}(s')dsds'}{\int\int \boldsymbol{\alpha}(s)^T \hat{\boldsymbol{\Sigma}}(s,s')\boldsymbol{\alpha}(s')dsds' + P(\boldsymbol{\alpha})}$$

$$\text{s.t.} \quad \int\int \boldsymbol{\alpha}(s)^T \hat{\Sigma}(s,s')\boldsymbol{\alpha}(s')dsds' = 1$$

$$\text{and} \quad \int\int \boldsymbol{\alpha}(s)^T \hat{\boldsymbol{\Sigma}}(s,s')\boldsymbol{\alpha}_{k'}(s')dsds' = 0 \quad \text{for} \quad k' < k$$

where $\hat{\boldsymbol{B}}(s,s') = \sum_{\ell=1}^{n}\sum_{\ell'=1}^{n}\{\boldsymbol{X}_\ell(s) - \bar{\boldsymbol{X}}(s)\}\{\boldsymbol{Y}_\ell - \bar{\boldsymbol{Y}}\}^T\{\boldsymbol{Y}_{\ell'} - \bar{\boldsymbol{Y}}\}\{\boldsymbol{X}_{\ell'}(s') - \bar{\boldsymbol{X}}(s')\}^T/n^2$, $\hat{\boldsymbol{\Sigma}}(s,s') = \sum_{\ell=1}^{n}\{\boldsymbol{X}_\ell(s) - \bar{\boldsymbol{X}}(s)\}\{\boldsymbol{X}_\ell(s') - \bar{\boldsymbol{X}}(s')\}^T/n$, and penalty

$$P(\boldsymbol{\alpha}) = \lambda\sum_{i=1}^{p}\{||\alpha_i||^2 + \tau||\alpha_i''||^2\}.$$

Then we estimate $\{w_k(t), k > 0\}$ by regressing $\{\boldsymbol{Y}_\ell\}$ on $\{\hat{z}_{\ell,1}, ...\hat{z}_{\ell,K}\}$ using least square method. Here $\hat{z}_{\ell,k} = \int(\boldsymbol{X}_\ell(s) - \bar{\boldsymbol{X}}(s))^T\hat{\boldsymbol{\alpha}}_k(s)ds$.

Value

An object of the "cv.msof" class, which is used in the function `pred.msof` for prediction.

`fitted_model`	a list containing information about fitted model.
`is_Y_vector`	a logic value indicating whether `Y` is a vector.
`Y`	input data `Y`.
`x.smooth.params`	a list for internal use.

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo and Xin Qi (Submitted)

Examples

```
##############################################################################
# Example: multiple scalar-on-function regression
##############################################################################
```

```
ptm <- proc.time()
library(FRegSigCom)
data(corn)
X=corn$X
Y=corn$Y
ntrain=60 # in paper, we use 80 observations as training data
xtrange=c(0,1) # the range of t in x(t).
t.x.list=list(seq(0,1,length.out=ncol(X)))
train.index=sample(1:nrow(X), ntrain)
X.train <- X.test <- list()
X.train[[1]]=X[train.index,]
X.test[[1]]=X[-(train.index),]
Y.train <- Y[train.index,]
Y.test <- Y[-(train.index),]

fit.cv.1=cv.msof(X.train, Y.train, t.x.list)# the cv procedure for our method
Y.pred=pred.msof(fit.cv.1, X.test) # make prediction on the test data

pred.error=mean((Y.pred-Y.test)^2)
print(c("pred.error=",pred.error))

print(proc.time()-ptm)
```

`cv.msof.hd` *Cross-validation for high dimensional linear multivariate scalar-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 multivariate scalar-on-function regression model with a large number of functional predictors:

$$\boldsymbol{Y} = \boldsymbol{\mu} + \sum_{i=1}^{p} \int_{a_i}^{b_i} X_i(s)\boldsymbol{\beta}_i(s)ds + \boldsymbol{\epsilon}$$

where $\boldsymbol{Y}$ is an m-dimensional multivariate response variable, $\boldsymbol{\mu}$ is the m-dimensional intercept vector. The $\{X_i(s), 1 \leq i \leq p\}$ are p functional predictors and $\{\boldsymbol{\beta}_i(s), 1 \leq i \leq p\}$ are their corresponding m-dimensional vector of coefficient functions, where p is a positive integer. The $\boldsymbol{\epsilon}$ is the random noise vector.

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.

Usage

```
cv.msof.hd(X, Y, t.x.list, n.basis = 25, K.cv = 5, upper.comp = 10,
           thresh = 0.02)
```

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	an $n \times q$ data matrix for the response Y or a n-dimensional vector if there is only one scalar response, where n is the sample size, and q is the number of scalar response variables.
t.x.list	a list of length p. Its i-th element is the vector of observation time points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.
n.basis	the number of basis functions used for estimating the vector of functions $\psi_{ik}(s)$ (see the reference for details). 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 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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 0.02.

Details

We use the decomposition $\boldsymbol{\beta}_i(s) = \sum_{k=1}^{K} \alpha_{ki}(s)\boldsymbol{w}_k$, $1 \leq i \leq p$, based on the KL expansion of $\sum_{i=1}^{p} \int X_i(s)\boldsymbol{\beta}_i(s)ds$. Let $\boldsymbol{Y}_\ell = (Y_{\ell,1}, ..., Y_{\ell,m})^T$ and $\boldsymbol{X}_\ell(t) = (X_{\ell,1}(s), ..., X_{\ell,p}(s))^T$, $1 \leq \ell \leq n$, denote n independent samples. We estimate $\boldsymbol{\alpha}_k = (\alpha_{k1}, ..., \alpha_{kp})^T$ for each k by solving the panelized generalized functional eigenvalue problem

$$max_\alpha \frac{\int\int \boldsymbol{\alpha}(s)^T \hat{\boldsymbol{B}}(s,s')\boldsymbol{\alpha}(s')dsds'}{\int\int \boldsymbol{\alpha}(s)^T \hat{\boldsymbol{\Sigma}}(s,s')\boldsymbol{\alpha}(s')dsds' + P(\boldsymbol{\alpha})}$$

$$\text{s.t.} \quad \int\int \boldsymbol{\alpha}(s)^T \hat{\Sigma}(s,s')\boldsymbol{\alpha}(s')dsds' = 1$$

$$\text{and} \quad \int\int \boldsymbol{\alpha}(s)^T \hat{\boldsymbol{\Sigma}}(s,s')\boldsymbol{\alpha}_{k'}(s')dsds' = 0 \quad \text{for} \quad k' < k$$

where $\hat{\boldsymbol{B}}(s,s') = \sum_{\ell=1}^{n}\sum_{\ell'=1}^{n}\{\boldsymbol{X}_\ell(s) - \bar{\boldsymbol{X}}(s)\}\{\boldsymbol{Y}_\ell - \bar{\boldsymbol{Y}}\}^T\{\boldsymbol{Y}_{\ell'} - \bar{\boldsymbol{Y}}\}\{\boldsymbol{X}_{\ell'}(s') - \bar{\boldsymbol{X}}(s')\}^T/n^2$, $\hat{\boldsymbol{\Sigma}}(s,s') = \sum_{\ell=1}^{n}\{\boldsymbol{X}_\ell(s) - \bar{\boldsymbol{X}}(s)\}\{\boldsymbol{X}_\ell(s') - \bar{\boldsymbol{X}}(s')\}^T/n$, and penalty

$$P(\alpha) = \tau\{(1-\lambda)\sum_{i=1}^{p}\|\alpha_i\|_\eta^2 + \lambda(\sum_{i=1}^{p}\|\alpha_i\|_\eta)^2\}$$

where $\|\alpha_i\|_\eta^2 = \|\alpha_i\|^2 + \eta\|\alpha_i''\|^2$. Then we estimate $\{w_k(t), k > 0\}$ by regressing $\{\boldsymbol{Y}_\ell\}$ on $\{\hat{z}_{\ell,1}, ...\hat{z}_{\ell,K}\}$ using least square method. Here $\hat{z}_{\ell,k} = \int(\boldsymbol{X}_\ell(s) - \bar{\boldsymbol{X}}(s))^T\hat{\alpha}_k(s)ds$.

Value

An object of the "cv.msof.hd" class, which is used in the function `pred.msof.hd` for prediction.

`opt.K`	optimal number of components to be selected.
`opt.tau`	optimal value for tau.
`opt.lambda`	optimal value for $lambda$.
`opt.eta`	optimal value for $lambda$.
`min.error`	minimum CV error.
`errors`	list for CV errors.
`...`	for internal use

Author(s)

Ruiyan Luo and Xin Qi

References

Ruiyan Luo and Xin Qi (Submitted)

Examples

```
##############################################################################
# Example:
##############################################################################

#toy example

ptm <- proc.time()
library(FRegSigCom)
data(air)
Y=matrix(0, nrow(air[[1]]), 3)
Y[,1]=apply(air[[1]][,1:8],1,mean)
Y[,2]=apply(air[[1]][,(1:8)+8],1,mean)
Y[,3]=apply(air[[1]][,(1:8)+16],1,mean)

X=list()
for(i in 1:(length(air)-1))
{
  X[[i]]=air[[i+1]]
}

ntrain=100 # in paper, we use 80 observations as training data
t.x=seq(0,1,length.out=ncol(X[[1]]))
train.index=sample(1:nrow(X[[1]]), ntrain)
X.train <- X.test <- list()
t.x.list=list()
for(i in 1:length(X))
{
```

```
  t.x.list[[i]]=t.x
  X.train[[i]]=X[[i]][train.index,]
  X.test[[i]]=X[[i]][-(train.index),]
}
Y.train <- Y[train.index,]
Y.test <- Y[-(train.index),]

fit.cv=cv.msof.hd(X.train, Y.train, t.x.list, upper.comp=5)
# in practice, use the default values (or larger) for
# "upper.comp" and "n.basis".

Y.pred=pred.msof.hd(fit.cv, X.test)
pred.error=mean((Y.pred-Y.test)^2)
print(c("pred.error=",pred.error))

print(proc.time()-ptm)
```

`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, $\{F_i(x, s, t), 1 \leq i \leq p\}$ are all unspecified nonlinear functions of x, s, t, $\{X_i(s), 1 \leq i \leq p\}$ 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.

Usage

```
cv.nonlinear(X, Y, t.x.list, t.y, s.n.basis = 40, x.n.basis = 40,
        t.n.basis = 40, K.cv = 5, upper.comp = 10, thresh = 0.01)
```

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 observation 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 B-spline basis functions for the argument s. Default is 40.
x.n.basis	the number of B-spline basis functions for x. Default is 40.
t.n.basis	the number of B-spline basis functions for t. Default is 4.
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 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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.

Details

This method estimates a special decomposition:

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

We first estimate $\boldsymbol{G}_k = (G_{1k}(x_1,s),...,G_{pk}(x_p,s))^T$ for each $k > 0$ by solving a generalized penalized functional eigenvalue problem

$$max_{\boldsymbol{G}} \frac{\hat{\boldsymbol{\Lambda}}(\boldsymbol{G},\boldsymbol{G})}{\hat{\boldsymbol{\Sigma}}(\boldsymbol{G},\boldsymbol{G}) + P(\boldsymbol{G})}$$

$$\text{s.t.} \quad \hat{\boldsymbol{\Sigma}}(\boldsymbol{G}_{k'},\boldsymbol{G}) = 0$$

$$\text{and} \quad \int\int \boldsymbol{G}(s)^T \hat{\Sigma}(s,s')\boldsymbol{G}_{k'}(s')dsds' = 0 \quad \text{for} \quad k' < k$$

where

$$\hat{\boldsymbol{\Lambda}}(\boldsymbol{G},\boldsymbol{G}) = \int \left[\sum_{\ell=1}^{n}\sum_{i=1}^{p} \int \{G_i(x_{\ell,i}(s),s) - \bar{G}_i(s)\}ds\{y_\ell(t) - \bar{y}(t)\}\right]^2 dt/n^2,$$

$$\hat{\boldsymbol{\Sigma}}(\boldsymbol{G},\tilde{\boldsymbol{G}}) = \sum_{\ell=1}^{n}\left[\sum_{i=1}^{p} \int \{G_i(x_{\ell,i}(s),s) - \bar{G}_i(s)\}ds\right]\left[\sum_{j=1}^{p} \int \{\tilde{G}_j(x_{\ell,j}(s),s) - \bar{\tilde{G}}_j(s)\}ds\right]/n,$$

penlaty

$$P(G) = \lambda\sum_{j=1}^{p}\{\|G_j\|^2 + \tau(\|\partial_{xx}G_j\|^2 + \|\partial_{xs}G_j\|^2 + \|\partial_{ss}G_j\|^2)\},$$

and $< G, \tilde{G} >_{H^2} = < G, \tilde{G} >_{L^2} + < \partial_{xx} G, \partial_{xx} \tilde{G} >_{L^2} + < \partial_{xs} G, \partial_{xs} \tilde{G} >_{L^2} + < \partial_{ss} G, \partial_{ss} \tilde{G} >_{L^2}$ with $< G, \tilde{G} >_{L^2} = \int \int G(x,s)\tilde{G}(x,s)dxds$. Then we estimate $\{w_k(t), k > 0\}$ by regressing $\{y_\ell(t)\}_{\ell=1}^n$ on $\{\hat{z}_{\ell,1}, ...\hat{z}_{\ell,K}\}_{\ell=1}^n$ with penalty $\kappa \sum_{k=0}^K \|w_k''\|^2$ tuned by the smoothness parameter κ. Here $\hat{z}_{\ell,k} = \sum_{i=1}^p \int (G_{ik}(x_{\ell,i}(s), s) - \bar{G}_{ik}(s))ds$, and $\bar{G}_{ik}(s) = \sum_{\ell=1}^n G_{ik}(X_{\ell,i}(s), s)/n$.

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.K`	optimal number of components to be selected.
`opt.lambda`	optimal value for λ.
`opt.tau`	optimal value for τ.
`opt.kappa`	optimal value for κ, the smoothness tuning parameter for $w_k(t)$.
`...`	other output for internal use.

Author(s)

Xin Qi and Ruiyan Luo

References

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

Examples

```
##############################################################################
# Example: Nonlinear function-on-function model
##############################################################################

ptm <- proc.time()
library(FRegSigCom)
library(refund)
data(DTI)
I=which(is.na(apply(DTI$cca,1,mean)))
X=DTI$cca[-I,] # functional response
Y=DTI$rcst[-I,-(1:12)] #functional predictor
t.x <- list(seq(0,1,length=dim(X)[2]))
t.y <- seq(0,1,length=dim(Y)[2])
# randomly split all the observations into a training set with 200 observations
# and a test set.
train.id=sample(1:nrow(Y), 30)
X.train.list <- list(X[train.id,])
Y.train <- Y[train.id, ]
X.test.list <- list(X[-(train.id),])
Y.test <- Y[-(train.id), ]
fit.cv=cv.nonlinear(X.train.list, Y.train, t.x, t.y, upper.comp=3,
```

```
s.n.basis=20, x.n.basis=20,t.n.basis=20)
           # in practice, use the default values (or larger) for
           # "upper.comp", "s.n.basis","x.n.basis", and "t.n.basis".
Y.pred=pred.nonlinear(fit.cv, X.test.list)
nonlinear.error= mean((Y.pred-Y.test)^2)
print(c("prediction error=",nonlinear.error))
print(proc.time()-ptm)
```

`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 with or without 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. When Z =NULL, there is no scalar predictor. The $\{X_i(s), 1 \le i \le p\}$ are p functional predictors and $\{\beta_i(s,t), 1 \le i \le p\}$ are their corresponding coefficient functions, where p is a positive integer. When p is large (e.g., greater than 6), in addition to smoothness penalty, one may want to consider to impose 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.

Usage

```
cv.sigcom(X, Y, t.x, t.y, Z = NULL, s.n.basis = 50, t.n.basis = 50,
          K.cv = 5, upper.comp = 20, thresh = 0.001,
          basis.type.x="Bspline", basis.type.y="Bspline")
```

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 $X_i(s)$, $1 \le i \le p$.
`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 20.
thresh	a number between 0 and 1 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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 0.001.
basis.type.x	the type of basis functions $\psi_{ik}(s)$. Only "BSpline" (default) and "Fourier" are supported.
basis.type.y	the type of basis functions $w_k(t)$. Only "BSpline" (default) and "Fourier" are supported.

Details

We consider the model with $p = 1$ and Z=NULL. Details for the model with scalar and multiple functional predictors can be found in the reference and its supplementary material. We consider a special decomposition of the coefficient function:

$$\beta(s,t) = \sum_{k=1}^{\infty} \psi_k(s) w_k(t).$$

Let $\{(x_1(s), y_1(t)), ..., (x_n(s), y_n(t))\}$ denote the samples. We first estimate $\psi_k(s)$ for each $k > 0$ by solving a penalized generalized functional eigenvalue problem

$$max_{\psi} \frac{\int\int \psi(s)\hat{B}(s,s')\psi(s')dsds'}{\int\int \psi(s)\hat{\Sigma}(s,s')\psi(s')dsds' + P(\psi)}$$

$$\text{s.t.} \quad \int\int \psi(s)\hat{\Sigma}(s,s')\psi(s')dsds' = 1$$

$$\text{and} \quad \int\int \psi(s)\hat{\Sigma}(s,s')\psi_{k'}(s')dsds' = 0 \quad \text{for} \quad k' < k$$

where $\hat{B}(s,s') = \sum_{\ell=1}^{n}\sum_{\ell'=1}^{n}\{x_\ell(s) - \bar{x}(s)\}\int\{y_\ell(t) - \bar{y}(t)\}\{y_{\ell'}(t) - \bar{y}(t)\}dt\{x_{\ell'}(s') - \bar{x}(s')\}/n^2$, $\hat{\Sigma}(s,s') = \sum_{\ell=1}^{n}\{x_\ell(s) - \bar{x}(s)\}\{x_\ell(s') - \bar{x}(s')\}/n$, and penalty

$$P(\psi) = \lambda\{||\psi_k||^2 + \tau||\psi_k''||^2\}.$$

Then we estimate $\{w_k(t), k > 0\}$ by regressing $\{y_\ell(t)\}_{\ell=1}^n$ on $\{\hat{z}_{\ell 1}, ...\hat{z}_{\ell K}\}_{\ell=1}^n$ with penalty $\kappa\sum_{k=0}^{K}\|w_k''\|^2$ tuned by the smoothness parameter κ. Here $\hat{z}_{\ell k} = \int(x_\ell(s) - \bar{x}(s))\hat{\psi}_k(s)ds$.

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: linear function-on-function regresion with one predictor
#             curve and two scalar predictors
#############################################################################
ptm <- proc.time()
library(FRegSigCom)
library(refund)
data(DTI)
tmp=1*(DTI$sex=="male")
Z.0=cbind(DTI$case, tmp)

I=which(is.na(apply(DTI$cca,1,mean)))
Y=DTI$cca[-I,] # functional response
X=list(DTI$rcst[-I,-(1:12)]) #functional predictor
```

```
Z=Z.0[-I,] # scalar predictor

t.x <- list(seq(0,1,length=dim(X[[1]])[2]))
t.y <- seq(0,1,length=dim(Y)[2])
# randomly split all the observations into a training set with 200 observations
# and a test set.
train.id=sample(1:nrow(Y), 100)
X.train <- list(X[[1]][train.id,])
Y.train <- Y[train.id, ]
Z.train <- Z[train.id, ]
X.test <- list(X[[1]][-(train.id),])
Y.test <- Y[-(train.id), ]
Z.test <- Z[-(train.id), ]
fit.cv=cv.sigcom(X.train, Y.train, t.x, t.y, Z=Z.train)
Y.pred=pred.sigcom(fit.cv, X.test, Z.test=Z.test)
error<- mean((Y.pred-Y.test)^2)
print(c(" prediction error=", error))
coef.est.list=getcoef.sigcom(fit.cv)
mu.est=coef.est.list[[1]] # intercept curve
beta.est=coef.est.list[[2]] # coefficient functions of functional predictors
gamma.est=coef.est.list[[3]] # coefficient functions of scalar predictors
print(proc.time()-ptm)



#############################################################################
# Example 2: linear function-on-function regresion with four predictor curves
#############################################################################


ptm <- proc.time()
library(FRegSigCom)
data(ocean)
Y=ocean[[1]]
Y.train=Y[1:80,]
Y.test=Y[-(1:80),]
t.y=seq(0,1, length.out = ncol(Y))
X.list=list()
X.train.list=list()
X.test.list=list()
t.x.list=list()
for(i in 1:4)
{
  X.list[[i]]=ocean[[i+1]]
  X.train.list[[i]]=X.list[[i]][1:80,]
  X.test.list[[i]]=X.list[[i]][-(1:80),]
  t.x.list[[i]]=seq(0,1, length.out = ncol(X.list[[i]]))
}
fit.cv=cv.sigcom(X.train.list, Y.train, t.x.list, t.y)
Y.pred=pred.sigcom(fit.cv, X.test.list)
error<- mean((Y.pred-Y.test)^2)
print(c(" prediction error=", error))
coef.list=getcoef.sigcom(fit.cv)
```

```
mu.est=coef.list[[1]] # intercept curve
beta.est=coef.list[[2]] # coefficient functions of functional predictors
print(proc.time()-ptm)
```

`cv.sof.spike` *Cross-validation for linear scalar-on-function regression for highly densely observed spiky functional data*

Description

This function is used to perform cross-validation and build the final model for highly densely observed spiky data using the signal compression approach for the following linear scalar-on-function regression model:

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

where μ is the intercept. The $\{X_i(s), 1 \leq i \leq p\}$ are p functional predictors and $\{\beta_i(s), 1 \leq i \leq p\}$ are their corresponding coefficient functions, where p is a positive integer. The ϵ is the random noise.

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.

Usage

```
cv.sof.spike(X, Y, t.x, K.cv = 10, upper.level = 10)
```

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`	an n dimensional vector of the observed values for the response, where n is the sample size.
`t.x`	a list of length p. Its i-th element is the vector of obesrvation time points of the i-th functional predictor $X_i(s)$, $1 \leq i \leq p$.
`K.cv`	the number of CV folds. Default is 10.
`upper.level`	the upper bound of the maximum resolution level. The optimal maximum resolution level is chosen between 1 and "upper.level", together with other tuning parameters, by cross-validation.

Details

This method uses wavelet basis to expand $X_i(s)$ and $\beta_i(s)$, $(1 \leq i \leq p)$, and estimates the expansion coefficients of $\beta_i(s)$'s by penalized least squares method with penalty

$$\lambda \sum_{i=1}^{p} \{ \sum_{j=0}^{J_1} \{ 2^{-2\alpha e^{-(j-\tau)/\alpha}} 2^{2\alpha j} ||b_{ij}||^2 + \kappa ||b_i||^2 \} \},$$

where b_{ij} denotes the vector of wavelet coefficient for $\beta_i(s)$ at the jth level, and b_i is the vector concatenating all b_{ij}, $(0 \leq j \leq J_1)$.

Value

An object of the "cv.sof.spike" class, which is used in the function `pred.sof.spike` for prediction.

`mu`	the estimated intercept.
`coef`	a list of p vectors, where the i-th vector contains the estimated values of the slope coefficient function $beta_i(s)$ at $t.x$.
`...`	optimal tuning parameters

Author(s)

Xin Qi and Ruiyan Luo,

References

Xin Qi and Ruiyan Luo, (manuscript) Functional regression for highly densely observed functional data with novel regularity.

Examples

```
##############################################################################
# Example: scalar-on-function for highly-densely observed curves
##############################################################################


ptm <- proc.time()
library(FRegSigCom)
data(Pork)
X=Pork$X
Y=Pork$Y
ntrain=40 # in paper, we use 80 observations as training data
xtrange=c(0,1) # the range of t in x(t).
t.x.list=list(seq(0,1,length.out=dim(X)[2]))
train.index=sample(1:dim(X)[1], ntrain)
X.train <- X.test <- list()

X.train[[1]]=X[train.index,]
X.test[[1]]=X[-(train.index),]
```

```
Y.train <- Y[train.index]
Y.test <- Y[-(train.index)]

fit.cv=cv.sof.spike(X.train, Y.train, t.x.list)
Y.pred=pred.sof.spike(fit.cv, X.test)
pred.error=mean((Y.pred-Y.test)^2)

print(c("pred.error=",pred.error))

print(proc.time()-ptm)
```

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

Description

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

Usage

```
getcoef.ff.interaction(fit.obj, t.x.coef=NULL, t.y.coef=NULL)
```

Arguments

`fit.obj`	the output object of `cv.ff.interaction`, or `step.ff.interaction`.
`t.x.coef`	a list of length p of vectors providing the observation time points of predictors on which coefficient functions will be evaluated. If `t.x.coef`=NULL (default), `t.x` in `cv.ff.interaction` or `step.ff.interaction` will be used.
`t.y.coef`	a vector of observation time points of response function on which the coefficient functions will be evaluated. If `t.y.coef`=NULL (default), `t.y` in `cv.ff.interaction` or `step.ff.interaction` will be used.

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.coef` 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.coef[[i]] and t.y.coef, 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.coef[[i]], t.x.coef[[j]] and t.y.coef, where the pair i,j is in inter_effects.

Author(s)

Xin Qi and Ruiyan Luo

References

Ruiyan Luo and Xin Qi (2018) Interaction model and model selection for function-on-function regression, Journal of Computational and Graphical Statistics. https://doi.org/10.1080/10618600.2018.1514310

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 coefficient functions for linear function-on-function models with a large numbe of predictor curves*

Description

This function is used to 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` which is the vector of observation time points for the response function used in `cv.hd`.
`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 two-dimensional grid created by `t.x.list[[i]]` and `t.y` used in `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 in nonlinear function-on-function model*

Description

This function is used to calculate the estimates for $\mu(t), F_i(x,s,t)'s$ based on the object obtained from `cv.nonlinear`.

Usage

```
getcoef.nonlinear(fit.cv, n.x.grid = 50)
```

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 50.

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

See Also

`cv.nonlinear`.

Examples

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

`getcoef.sigcom` *Get the estimated intercept and coefficient functions for linear function-on-function models*

Description

This function is used to calculate the estimates for $\mu(t), \alpha(t), \beta_i(s, t)$ based on the object obtained from `cv.sigcom`.

Usage

```
getcoef.sigcom(fit.obj, t.x.coef=NULL, t.y.coef=NULL)
```

Arguments

`fit.obj`	the object obtained from `cv.sigcom`.
`t.x.coef`	a list of length p of vectors providing the observation time points of predictors on which coefficient functions will be evaluated. If `t.x.coef`=NULL (default), `t.x` in `cv.sigcom` will be used.
`t.y.coef`	a vector of observation time points of response function on which the coefficient functions will be evaluated. If `t.y.coef`=NULL (default), `t.y` in `cv.sigcom` will be used.

Value

a list containing

`mu`	the vector of the estimated values of $\mu(t)$ at the grid of observation points of the response function ($t.y.coef$ 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.coef[[i]]` and `t.y` for `cv.sigcom`. The columns correspond to different observation points for response varible ($t.y.coef$).

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)
```

`Pork` *Pork fat samples*

Description

The spectrum curves of Raman spectroscopy and the percentages of the unsaturated fatty acid obtained from 105 samples of the fat taken from the daily production stock of a slaughterhouse.

Usage

```
data(Pork)
```

Format

A list of two matrix:

X a matrix with 105 rows and 566 columns with each row representing the spectrum curve of Raman spectroscopy from one fat sample.

Y a vector of length 105, giving the percentages of the unsaturated fatty acid of the 105 samples.

Source

http://www.models.life.ku.dk/RAMANporkfat

References

Lotte Boge Lyndgaard, Klavs Martin Sorensen, Frans van den Berg and Soren Balling Engelsen (2012): Depth profiling of porcine adipose tissue by Raman spectroscopy, Journal of Raman Spectroscopy, 43, 482-489.

Xin Qi and Ruiyan Luo. (Manuscript) Functional regression for highly densely observed functional data with novel regularity.

See Also

`cv.sof.spike`

Examples

```
data(Pork)
str(Pork)
```

`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, t.y.test=NULL)
```

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`.
`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.ff.interaction` or `step.ff.interaction` will be used.

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.fof.spike` *Prediction for linear function-on-function regression on highly sensely observed spiky data*

Description

Make predition for functional response from the CV object obtained by `cv.fof.spike`.

Usage

```
pred.fof.spike(fit.cv, X.test)
```

Arguments

`fit.cv`	the CV object obtained by `cv.fof.spike`.
`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.

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`.

Author(s)

Xin Qi and Ruiyan Luo

See Also

`cv.fof.spike`

Examples

```
#See the examples in cv.fof.spike().
```

`pred.fof.wv` *Prediction for linear function-on-function regression using signal compression*

Description

Make predition for functional response from the CV object obtained by `cv.fof.wv`.

Usage

```
pred.fof.wv(cv.obj, X.test)
```

Arguments

`cv.obj`	the CV object obtained by `cv.fof.wv`.
`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.

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`.

Author(s)

Ruiyan Luo and Xin Qi

See Also

`cv.fof.wv`

Examples

```
#See the examples in cv.fof.wv().
```

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.msof` *Prediction for linear multivariate scalar-on-function regression*

Description

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

Usage

```
pred.msof(fit.obj, X.test)
```

Arguments

`fit.obj`	the CV object obtained by `cv.msof`.
`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.

Value

A matrix which contains the predicted values of the multivariate response. Each row is the prediction for each new observation.

Author(s)

Ruiyan Luo and Xin Qi

See Also

`cv.msof`

Examples

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

`pred.msof.hd` *Prediction for high dimensional linear multivariate scalar-on-function regression*

Description

Make predition for multivariate scalar response based on new observations of large number of predictor curves from the CV object obtained by `cv.msof.hd`.

Usage

```
pred.msof.hd(fit.cv, X.test)
```

Arguments

`fit.cv`	the CV object obtained by `cv.msof.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.

Value

A matrix which contains the predicted values of the multivariate response. Each row is the prediction for each new observation.

Author(s)

Ruiyan Luo and Xin Qi

See Also

`cv.msof.hd`

Examples

```
#See the examples in cv.msof.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.obj, X.test, t.y.test=NULL, Z.test = NULL)
```

Arguments

fit.obj	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().
```

`pred.sof.spike` *Prediction for linear function-on-function regression on highly sensely observed spiky data*

Description

Make predition for the scalar response from the CV object obtained by `cv.sof.spike`.

Usage

```
pred.sof.spike(fit.cv, X.test)
```

Arguments

`fit.cv`	the CV object obtained by `cv.sof.spike`.
`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.

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)

Xin Qi and Ruiyan Luo

See Also

`cv.sof.spike`

Examples

```
#See the examples in cv.sof.spike().
```

`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, adaptive=FALSE, s.n.basis=40, t.n.basis=40,
          inter.n.basis=20, basis.type.x="Bspline", basis.type.y="Bspline",
          K.cv=5, 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 $Y(t)$.
`adaptive`	a logic value indicating whether using adaptive penalty that uses different smoothness tuning parameters for different target functions. Default is FALSE.
`s.n.basis`	the number of basis functions used for estimating the functions $\psi_{ik}(s)$ (see details in `cv.ff.interaction`). 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.x` the type of basis functions $\psi_{ik}(s)$. Only "BSpline" (default) and "Fourier" are supported.

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

`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 8.

`thresh` a number between 0 and 1 used to determine the maximum number of components we need to calculate. The maximum number is between one and the "upp.comp" above. The optimal number of components will be chosen between 1 and this maximum number, together with other tuning parameters by cross-validation. A smaller thresh value leads to a larger maximum number of components 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 (2018) Interaction model and model selection for function-on-function regression, Journal of Computational and Graphical Statistics. https://doi.org/10.1080/10618600.2018.1514310

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