Package: fdaconcur (via r-universe)

August 20, 2024

Type Package

Title Concurrent Regression and History Index Models for Functional Data

URL https://github.com/functionaldata/tFDAconcur

BugReports https://github.com/functionaldata/tFDAconcur/issues

Version 0.1.3

Encoding UTF-8

Date 2024-07-16

Maintainer Su I Iao <siao@ucdavis.edu>

Description Provides an implementation of concurrent or varying coefficient regression methods for functional data. The implementations are done for both dense and sparsely observed functional data. Pointwise confidence bands can be constructed for each case. Further, the influence of past predictor values are modeled by a smooth history index function, while the effects on the response are described by smooth varying coefficient functions, which are very useful in analyzing real data such as COVID data. References: Yao, F., Müller, H.G., Wang, J.L. (2005) <doi:10.1214/009053605000000660>. Sentürk, D., Müller, H.G. (2010) <doi:10.1198/jasa.2010.tm09228>.

License BSD_3_clause + file LICENSE

Imports Rcpp (>= 0.11.5), fdapace (>= 0.6.0), stats

LinkingTo Rcpp, RcppEigen

Suggests MASS, Matrix, pracma, numDeriv, testthat

RoxygenNote 7.3.1

NeedsCompilation yes

Author Su I Iao [aut, cre], Satarupa Bhattacharjee [aut], Yaqing Chen [aut], Changbo Zhu [aut], Han Chen [aut], Yidong Zhou [aut], Álvaro Gajardo [aut], Poorbita Kundu [aut], Hang Zhou [aut], Hans-Georg Müller [cph, ths, aut]

Repository CRAN

Date/Publication 2024-07-20 06:30:02 UTC

Contents

`ConcReg_Lag` *Functional Concurrent Regression with Lag Model*

Description

Functional concurrent regression model with lag for dense functional responses and dense functional predictors.

Usage

```
ConcReg_Lag(Y, X, Lag = NULL, optnsY = NULL, optnsX = NULL)
```

Arguments

`Y`	a list which contains functional responses in the form of a list LY and the time points LT at which they are observed (i.e., list(Ly = LY,Lt = LT)).
`X`	a list of lists which contains the observed functional predictors list Lxj and the time points list Ltj at which they are observed. It needs to be of the form `list(list(Ly = Lx1,Lt = Lxt1),list(Ly = Lx2,Lt = Lxt2),...)`.
`Lag`	a length `length(X)` vector denoting the lags for all predictors.
`optnsY`	a list of options control parameters for the response specified by `list(name=value)`. See 'Details' in `fdapace::FPCA`.
`optnsX`	a list of options control parameters for the predictors specified by `list(name=value)`. See 'Details' in `fdapace::FPCA`.

Details

The functional concurrent regression model with lag is defined as

$$E[Y(t)|X_1(t),\cdots,X_p(t)] = \beta_0(t) + \sum_{j=1}^{p} \beta_j(t) X_j(t - Lag[j])$$

For more details we refer to *Şentürk, D. and Müller, H.G., (2010). Functional varying coefficient models for longitudinal data. Journal of the American Statistical Association, 105(491), pp.1256-1264.*

Value

A list of the following:

`beta0`	a vector of `length(workGridY)` representing the fitted $\beta_0(t)$
`beta`	A matrix for the concurrent regression effects, where rows correspond to different predictors and columns to different time points.
`workGridY`	a vetor representing the working grid for the response.
`Lag`	a vector representing the lags for all predictors

References

Şentürk, D. and Müller, H.G., (2010). Functional varying coefficient models for longitudinal data. Journal of the American Statistical Association, 105(491), pp.1256-1264. Yao, F., Müller, H.G., Wang, J.L. (2005). Functional linear regression analysis for longitudinal data. Annals of Statistics 33, 2873–2903. Hall, P., Horowitz, J.L. (2007). Methodology and convergence rates for functional linear regression. The Annals of Statistics, 35(1), 70–91.

Examples

```
phi1 <- function(t) sin(pi*t / 5) / sqrt(5)
phi2 <- function(t) cos(pi*t / 5) / sqrt(5)
lambdaX <- c(10, 5)
n <- 50
N <- 101
Xi <- matrix(rnorm(2*n), nrow = n, ncol = 2)
denseLt <- list()
denseLy <- list()
t0 <- seq(0, 15, length.out = N)
for (i in 1:n) {
  denseLt[[i]] <- t0
  denseLy[[i]] <- lambdaX[1]*Xi[i, 1]*phi1(t0) + lambdaX[2]*Xi[i, 2]*phi2(t0)
}
denseX0 <- list(Ly = denseLy,Lt = denseLt)

Lag <- c(3)
t0_out <- t0[t0>=max(Lag)]
beta_1 <- function(t) 5*sin(pi*t/10)
beta_0 <- function(t) t^2/2
### functional response Y(t), t in t0_out ###
denseLt <- list()
denseLy <- list()
for (i in 1:n) {
  denseLt[[i]] <- t0_out
  denseLy[[i]] <- beta_0(t0_out) +
    denseX0$Ly[[i]][1:length(t0_out)]* beta_1(t0_out) +
    rnorm(length(t0_out), 0, 0.1)
```

```
}
denseY <- list(Ly = denseLy, Lt = denseLt)
model = ConcReg_Lag(Y=denseY, X=list(X1 = denseX0), Lag=Lag)

print(model$workGridY) # workGrid for Y between 6 to 15
plot(beta_1(model$workGridY)) # workGrid for X1 between 3 to 12
plot(beta_0(model$workGridY))

plot(model$beta[1,])
plot(model$beta0)
```

ConcurReg *Functional Concurrent Regression using 2D smoothing*

Description

Functional concurrent regression with dense or sparse functional data for scalar or functional dependent variables. Note: function-to-scalar regression can also be handled using the VCAM function in fdapace.

Usage

```
ConcurReg(
  vars,
  outGrid,
  userBwMu = NULL,
  userBwCov = NULL,
  kern = "gauss",
  measurementError = FALSE,
  diag1D = "all",
  useGAM = FALSE,
  returnCov = TRUE
)
```

Arguments

vars	A list of input functional/scalar covariates. Each field corresponds to a functional (a list) or scalar (a vector) covariate. The last entry is assumed to be the response if no entry is named 'Y'. If a field corresponds to a functional covariate, it should have two fields: 'Lt', a list of time points, and 'Ly', a list of functional values.
outGrid	A vector of output time points.
userBwMu	A scalar/vector bandwidth used for smoothing the mean function. Each entry in the vector represents the bandwidth used for the corresponding covariate in vars. For the scalar covariates, you can input 0 as a placeholder. If you only input a scalar, the function will use the same bandwidth to smooth all mean functions. — a scalar/vector of positive numeric - default: NULL — if no scalar/vector

value is provided, the bandwidth value for the smoothed mean function is chosen using 'GCV';

`userBwCov` A scalar/vector bandwidth used for smoothing the auto or cross-covariances. If you use 1D smoothing for the diagonal line of the covariance (diag1D="all"), only one scalar input is needed. If you use 2D smoothing for the covariance (diag1D="none"), a vector of bandwidth is required. Each entry in the vector represents the bandwidth used for the corresponding covariate in vars. For the scalar covariates, you can input 0 as a placeholder. — a scalar/vector of positive numeric - default: NULL — if no scalar/vector is provided, the bandwidth value for the smoothed cross-covariance function is chosen using 'GCV';

`kern` Smoothing kernel choice, common for mu and covariance; "rect", "gauss", "epan", "gausvar", "quar" - default: "gauss".

`measurementError` Assume measurement error in the data; logical - default: FALSE. If TRUE the diagonal raw covariance will be removed when smoothing.

`diag1D` A string specifying whether to use 1D smoothing for the diagonal line of the covariance. 'none': don't use 1D smoothing; 'all': use 1D for both auto- and cross-covariances. (default : 'all')

`useGAM` Use GAM smoothing instead of local linear smoothing (semi-parametric option); logical - default: FALSE.

`returnCov` Return the covariance surfaces, which is a four dimensional array. The first two dimensions correspond to outGrid and the last two correspond to the covariates and the response, i.e. (i, j, k, l) entry being Cov(X_k(t_i), X_l(t_j)); logical - default: FALSE.

Details

If measurement error is assumed, the diagonal elements of the raw covariance will be removed. This could result in highly unstable estimate if the design is very sparse, or strong seasonality presents. WARNING! For very sparse functional data, setting measurementError = TRUE is not recommended.

Value

A list containing the following fields:

`beta` A matrix for the concurrent regression effects, where rows correspond to different predictors and columns to different time points.

`beta0` A vector containing the time-varying intercept.

`outGrid` A vector of the output time points.

`cov` A 4-dimensional array for the (cross-)covariance surfaces, with the (i, j, k, l) entry being Cov(X_k(t_i), X_l(t_j))

`R2` A vector of the time-varying R2.

`n` The sample size.

References

- *Yao, F., Müller, H.G., Wang, J.L. "Functional Linear Regression Analysis for Longitudinal Data." Annals of Statistics 33, (2005): 2873-2903.(Dense data)*
- *Sentürk, D., Müller, H.G. "Functional varying coefficient models for longitudinal data." J. American Statistical Association, 10, (2010): 1256–1264.*
- *Sentürk, D., Nguyen, D.V. "Varying Coefficient Models for Sparse Noise-contaminated Longitudinal Data", Statistica Sinica 21(4), (2011): 1831-1856. (Sparse data)*

Examples

```
# Y(t) = beta_0(t) + beta_1(t) X_1(t) + beta_2(t) Z_2 + epsilon
#
# Settings
set.seed(1)
n <- 30
nGridIn <- 50
sparsity <- 5:10 # Sparse data sparsity
T <- round(seq(0, 1, length.out=nGridIn), 4) # Functional data support
bw <- 0.1
outGrid <- round(seq(min(T), 1, by=0.05), 2)
 outGrid <- seq(min(T), max(T), by=0.05)
# Simulate functional data
mu <- T * 2 # mean function for X_1
sigma <- 1

beta_0 <- 0
beta <- rbind(cos(T), 1.5 + sin(T))
beta_2 <- 1

Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(mu, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
Y <- matrix(NA, n, nGridIn)
for (i in seq_len(n)) {
  Y[i, ] <- beta_0 + beta[1,]*X_1[i, ] + beta[2,]*Z[i, 2] + epsilon[i]
}

# Sparsify functional data
set.seed(1)
X_1sp <- fdapace::Sparsify(X_1, T, sparsity)
set.seed(1)
Ysp <- fdapace::Sparsify(Y, T, sparsity)
vars <- list(X_1=X_1sp, Z_2=Z[, 2], Y=Ysp)
res2 <- ConcurReg(vars, outGrid, userBwMu=c(.5,0,.5), userBwCov=c(.5,0,.5), kern='gauss',
 measurementError=TRUE, diag1D='none', useGAM = FALSE, returnCov=TRUE)
```

---

fdaconcur *fdaconcur: Concurrent Regression and History Index Models for Functional Data*

Description

This package provides tools for functional concurrent regression and history index models.

Details

fdaconcur for Functional Concurrent Regression and History Index Models

Provides an implementation of concurrent or varying coefficient regression methods for functional data. The implementations are done for both dense and sparsely observed functional data. Pointwise confidence bands can be constructed for each case. Further, the influence of past predictor values are modeled by a smooth history index function, while the effects on the response are described by smooth varying coefficient functions, which are very useful in analyzing real data such as COVID data.

References: Yao, F., Müller, H.G., Wang, J.L. (2005) <doi: 10.1214/009053605000000660>. Sentürk, D., Müller, H.G. (2010) <doi: 10.1198/jasa.2010.tm09228>.

PACE is based on the idea that observed functional data are generated by a sample of underlying (but usually not fully observed) random trajectories that are realizations of a stochastic process. It does not rely on pre-smoothing of trajectories, which is problematic if functional data are sparsely sampled.

Maintainer: Su I Iao <siao@ucdavis.edu>

Author(s)

Satarupa Bhattacharjee Yaqing Chen Changbo Zhu Han Chen Yidong Zhou Álvaro Gajardo Poorbita Kundu Hang Zhou

Hans-Georg Müller <hgmueller@ucdavis.edu>

See Also

Useful links:

- https://github.com/functionaldata/tFDAconcur
- Report bugs at https://github.com/functionaldata/tFDAconcur/issues

---

`fitted_ptFCReg` *Fitted functional responses from a* `ptFCReg` *object.*

Description

Fitted functional responses from a `ptFCReg` object.

Usage

```
fitted_ptFCReg(object)
```

Arguments

object	An object of class ptFCReg, returned by ptFCReg or smPtFCRegCoef.

Value

An n-by-m matrix, where each row corresponds to one subject, and each column corresponds to a time point in object$tGrid.

Examples

```
set.seed(1)
n <- 50
nGridIn <- 101
tGrid <- seq(0, 1, length.out=nGridIn) # Functional data support
muX1 <- tGrid * 2 # mean function for X_1
sigma <- 1
beta0 <- 0
beta <- rbind(cos(tGrid), 1.5 + sin(tGrid))
Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(muX1, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
Y <- t(sapply(seq_len(n), function(i) {
  beta0 + beta[1,] * X_1[i, ] + beta[2,] * Z[i, 2] + epsilon[i]
}))
dat <- list(X1=X_1, Z1=Z[, 2], Y=Y)
res <- ptFCReg(tGrid = tGrid, dat = dat)
smres <- smPtFCRegCoef(res, bw = 2.5 / (nGridIn-1), kernel_type = 'epan')
fit_res <- stats::fitted(res)
fit_smres <- stats::fitted(smres)
```

GetCI_Dense — *Bootstrap pointwise confidence intervals for the coefficient functions in functional concurrent regression for densely observed data.*

Description

Bootstrap pointwise confidence intervals for the coefficient functions in functional concurrent regression for densely observed data.

Usage

```
GetCI_Dense(dat, tGrid, level = 0.95, R = 10, bw, kernel_type)
```

Arguments

dat	A list of input functional/scalar covariates. Each field corresponds to a functional (a matrix) or scalar (a vector) variable. The last entry is assumed to be the functional response if no entry is names 'Y'. If a field corresponds to a functional variable, it should be an n-by-m matrix, where each row holds the

	observations for one subject on the common grid `tGrid`. If a field corresponds to a scalar covariate, it should be a vector of length n.
`tGrid`	A vector of length m with the input time points.
`level`	A number taking values in [0,1] determing the confidence level. Default: 0.95.
`R`	An integer holding the number of bootstrap replicates. Default: 999.
`bw`	Scalar holding the bandwidth.
`kernel_type`	Character holding the kernel type (see `Lwls1D` for supported kernels).

Value

A list containing the following fields:

CI_beta0 CI for the intercept function — A data frame holding three variables: `CIgrid` — the time grid where the CIs are evaluated; `CI_beta0.lower` and `CI_beta0.upper` — the lower and upper bounds of the CIs for the intercept function on `CIgrid`.

CI_beta A list containing CIs for the slope functions — the length of the list is same as the number of covariates. Each list contains the following fields: A data frame holding three variables: `CIgrid` — the time grid where the CIs are evaluated, `CI_beta_j.lower` and `CI_beta_j.upper` — the lower and upper bounds of the CIs for the intercept function on `CIgrid` for $j = 1, 2, \ldots$.

CI_R2 CI the time-varying $R^2(t)$ — A data frame holding three variables: `CIgrid` — the time grid where the CIs are evaluated, `CI_R2.lower` and `CI_R2.upper` — the lower and upper bounds of the CIs for the time-varying $R^2(t)$ on `CIgrid`.

level The confidence level of the CIs.

Examples

```
set.seed(1)
n <- 50
nGridIn <- 101
tGrid <- seq(0, 1, length.out=nGridIn) # Functional data support
muX1 <- tGrid * 2 # mean function for X_1
sigma <- 1
beta0 <- 0
beta <- rbind(cos(tGrid), 1.5 + sin(tGrid))
Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(muX1, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
Y <- t(sapply(seq_len(n), function(i) {
  beta0 + beta[1,] * X_1[i, ] + beta[2,] * Z[i, 2] + epsilon[i]
}))
dat <- list(X1=X_1, Z1=Z[, 2], Y=Y)
res <- ptFCReg(tGrid = tGrid, dat = dat)
smres <- smPtFCRegCoef(res, bw =  2.5 / (nGridIn-1), kernel_type = 'epan')
res_CI = GetCI_Dense(dat, tGrid, level = 0.95, R = 10, bw = 2.5 / (nGridIn-1), kernel_type = 'epan')
beta1 = res_CI$CI_beta[[1]] ##extracting CI for beta1
beta1a = beta1$CI_beta1.lower
beta1b = beta1$CI_beta1.upper
true_beta = beta[1,]  ##extracting true coef beta1 in the simulation setting
est_beta = smres$beta[1,] ## ##extracting estimated coef beta1 from
```

```
###fitting the concurrent regression model
plot(beta1$CIgrid, beta1a, type= 'l', ylim = c(0,2)) ##plot of lower CI for beta1
lines(beta1$CIgrid, beta1b) ##plot of lower CI for beta1
lines(beta1$CIgrid, true_beta, col ='red')
##plot of true coef beta1 in the simulation setting
lines(beta1$CIgrid, est_beta, col ='blue')
##plot of estimated coef beta1 from fitting the concurrent regression model
```

`GetCI_Sparse` *Bootstrap pointwise confidence intervals for the coefficient functions in functional concurrent regression for sparsely observed data.*

Description

Bootstrap pointwise confidence intervals for the coefficient functions in functional concurrent regression for sparsely observed data.

Usage

```
GetCI_Sparse(
  vars,
  outGrid = NULL,
  level = 0.95,
  R = 999,
  userBwMu = NULL,
  userBwCov = NULL,
  kern = "gauss",
  measurementError = FALSE,
  diag1D = "all",
  useGAM = FALSE
)
```

Arguments

`vars`	A list of input functional/scalar covariates. Each field corresponds to a functional (a list) or scalar (a vector) covariate. The last entry is assumed to be the response if no entry is named 'Y'. If a field corresponds to a functional covariate, it should have two fields: 'Lt', a list of time points, and 'Ly', a list of functional values.
`outGrid`	A vector with the output time points, which need to be within 5% and 95% of the range of functional covariates. If NULL, outGrid will be generated from 5% to 95% of the range of functional covariates with 51 grids for fitting. Default: NULL
`level`	A number taking values in [0,1] determining the confidence level. Default: 0.95.
`R`	An integer holding the number of bootstrap replicates. Default: 999.

`userBwMu` A scalar/vector bandwidth used for smoothing the mean function. Each entry in the vector represents the bandwidth used for the corresponding covariate in vars. For the scalar covariates, you can input 0 as a placeholder. If you only input a scalar, the function will use the same bandwidth to smooth all mean functions. — a scalar/vector of positive numeric - default: NULL — if no scalar/vector value is provided, the bandwidth value for the smoothed mean function is chosen using 'GCV';

`userBwCov` A scalar/vector bandwidth used for smoothing the auto or cross-covariances. If you use 1D smoothing for the diagonal line of the covariance (diag1D="all"), only one scalar input is needed. If you use 2D smoothing for the covariance (diag1D="none"), a vector of bandwidth is required. Each entry in the vector represents the bandwidth used for the corresponding covariate in vars. For the scalar covariates, you can input 0 as a placeholder. — a scalar/vector of positive numeric - default: NULL — if no scalar/vector is provided, the bandwidth value for the smoothed cross-covariance function is chosen using 'GCV';

`kern` Smoothing kernel choice, common for mu and covariance; "rect", "gauss", "epan", "gausvar", "quar" (default: "gauss")

`measurementError` Assume measurement error in the data; logical - default: FALSE. If TRUE the diagonal raw covariance will be removed when smoothing.

`diag1D` A string specifying whether to use 1D smoothing for the diagonal line of the covariance. 'none': don't use 1D smoothing; 'all': use 1D for both auto- and cross-covariances. (default : 'all')

`useGAM` Use GAM smoothing instead of local linear smoothing (semi-parametric option); logical - default: FALSE.

Details

If measurement error is assumed, the diagonal elements of the raw covariance will be removed. This could result in highly unstable estimate if the design is very sparse, or strong seasonality presents. WARNING! For very sparse functional data, setting measurementError = TRUE is not recommended.

Value

A list containing the following fields:

CI_beta0 CI for the intercept function — A data frame holding three variables: `CIgrid` — the time grid where the CIs are evaluated, `CI_beta0.lower` and `CI_beta0.upper` — the lower and upper bounds of the CIs for the intercept function on `CIgrid`.

CI_beta A list containing CIs for the slope functions — the length of the list is same as the number of covariates. Each list contains the following fields: A data frame holding three variables: `CIgrid` — the time grid where the CIs are evaluated, `CI_beta_j.lower` and `CI_beta_j.upper` — the lower and upper bounds of the CIs for the coefficient functions on `CIgrid` for $j = 1, 2, \ldots$.

CI_R2 CI the time-varying $R^2(t)$ — A data frame holding three variables: `CIgrid` — the time grid where the CIs are evaluated, `CI_R2.lower` and `CI_R2.upper` — the lower and upper bounds of the CIs for the time-varying $R^2(t)$ on `CIgrid`.

level The confidence level of the CIs.

Examples

```
set.seed(1)
n <- 30
nGridIn <- 100
sparsity <- 5:10 # Sparse data sparsity
T <- round(seq(0, 1, length.out=nGridIn), 4) # Functional data support
bw <- 0.1
outGrid <- round(seq(min(T), 1, by=0.05), 2)
# Simulate functional data
mu <- T * 2 # mean function for X_1
sigma <- 1

beta_0 <- 0
beta <- rbind(cos(T), 1.5 + sin(T))
beta_2 <- 1

Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(mu, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
Y <- matrix(NA, n, nGridIn)
for (i in seq_len(n)) {
  Y[i, ] <- beta_0 + beta[1,] * X_1[i, ] + beta[2,] * Z[i, 2] + epsilon[i]
}

# Sparsify functional data
set.seed(1)
X_1sp <- fdapace::Sparsify(X_1, T, sparsity)
Ysp <- fdapace::Sparsify(Y, T, sparsity)
vars <- list(X_1=X_1sp, Z_2=Z[, 2], Y=Ysp)
res <-  GetCI_Sparse(vars, outGrid[-c(1,21)], level = 0.95, R = 2,
                     userBwMu = c(.1,.1,.1), userBwCov = c(.1,.1,.1),
                     kern='gauss', measurementError=TRUE, diag1D='none',
                     useGAM = FALSE)
```

---

historyIndexDense *Functional History Index Model*

Description

Functional history index model for dense functional responses and dense functional predictors.

Usage

```
historyIndexDense(Y, X, Lag = NULL, optnsY = NULL, optnsX = NULL)
```

Arguments

Y	a list which contains functional responses in the form of a list LY and the time points LT at which they are observed (i.e., list(Ly = LY,Lt = LT)).
X	a list of lists which contains the observed functional predictors list Lxj and the time points list Ltj at which they are observed. It needs to be of the form `list(list(Ly = Lx1,Lt = Lxt1),list(Ly = Lx2,Lt = Lxt2),...)`.
Lag	a length `length(X)` vector denoting the lags for all predictors.
optnsY	a list of options control parameters for the response specified by `list(name=value)`. See 'Details' in `fdapace::FPCA`.
optnsX	a list of options control parameters for the predictors specified by `list(name=value)`. See 'Details' in `fdapace::FPCA`.

Details

The functional history index model is defined as $E[Y(t)|X_1(t), \cdots, X_p(t)] = \beta_0(t) + \sum_{i=1}^{p} \beta_i(t) \int_0^{\Delta_i} \gamma_i(s) X_i(t-s) ds$ for $t \in [\max_i\{\Delta_i\}, T]$ with a suitable $T > 0$. Write $\alpha_i(t,s) = \beta_i(t)\gamma_i(s)$. It becomes $E[Y(t)|X_1(t), \cdots, X_p(t)] = \beta_0(t) + \sum_{i=1}^{p} \int_0^{\Delta_i} \alpha_i(t,s) X_i(t-s) ds$. For more details we refer to *Şentürk, D. and Müller, H.G., (2010). Functional varying coefficient models for longitudinal data. Journal of the American Statistical Association, 105(491), pp.1256-1264.*

Value

A list of the following:

beta0	a vector of `length(workGridY)` representing the fitted $\beta_0(t)$
alpha	a list of matrices with the i-th element representing the fitted $\alpha_i(t,s)$
yHat	an n by `length(workGridY)` matrix of fitted $Y_i(t)$'s.
workGridY	a vetor representing the working grid for the response.
workGridLag	a list of vectors with the i-th element representing the working grid in $[0, \Delta_i]$.

References

Şentürk, D. and Müller, H.G., (2010). Functional varying coefficient models for longitudinal data. Journal of the American Statistical Association, 105(491), pp.1256-1264. Yao, F., Müller, H.G., Wang, J.L. (2005). Functional linear regression analysis for longitudinal data. Annals of Statistics 33, 2873–2903. Hall, P., Horowitz, J.L. (2007). Methodology and convergence rates for functional linear regression. The Annals of Statistics, 35(1), 70–91.

Examples

```
set.seed(1)
### functional covariate X(t) ###
phi1 <- function(t) sin(pi*t / 5) / sqrt(5)
phi2 <- function(t) cos(pi*t / 5) / sqrt(5)
lambdaX <- c(10, 5)
n <- 150
N <- 101
```

```
Xi <- matrix(rnorm(2*n), nrow = n, ncol = 2)
denseLt <- list()
denseLy <- list()
t0 <- seq(0, 15, length.out = N)
for (i in 1:n) {
  denseLt[[i]] <- t0
  denseLy[[i]] <- lambdaX[1]*Xi[i, 1]*phi1(t0) + lambdaX[2]*Xi[i, 2]*phi2(t0)
}
denseX0 <- list(Ly = denseLy, Lt = denseLt)

### generate coefficient function gamma(u), beta(u) ###
Lag <- 5
u0 <- t0[t0<=Lag]
t0_out <- t0[t0>=Lag]
gamma_u <- function(u) sqrt(2/5) * cos(pi * u /5)
beta_1 <- function(t) 5*sin(pi*t/10)
beta_0 <- function(t) t^2/2

### functional response Y(t), t in t0_out ###
denseLt <- list()
denseLy <- list()
for (i in 1:n) {
  denseLt[[i]] <- t0_out
  Xt <- denseX0$Ly[[i]]
  Xtu <- t(sapply((1:N)[t0>=Lag], function(j){
    rev(Xt[(j-length(u0)+1):j])  #history index for X[t-u:t]
  }))
  IntGammaXtu <- apply(Xtu, 1, function(v){
    fdapace::trapzRcpp(u0, gamma_u(u0) * v)
  })
  #append 0 in the first length(u0)-1 element(useless info. in our modeling)
 denseLy[[i]] <- beta_0(t0_out) + IntGammaXtu * beta_1(t0_out) + rnorm(length(t0_out), 0, 0.1)
}
denseY <- list(Ly = denseLy, Lt = denseLt)

### functional predictor X(t) (adjust for t0_out) ###
denseLt <- list()
denseLy <- list()
for (i in 1:n){
  denseLt[[i]] <- t0_out
  denseLy[[i]] <- denseX0$Ly[[i]][t0>=Lag]
}
denseX <- list(Ly = denseLy,
               Lt = denseLt)
fit <- historyIndexDense(Y = denseY, X = list(X = denseX), Lag = Lag)
fit$beta0
fit$alpha[[1]]
```

`historyIndexSparse` *Functional History Index Model*

Description

Functional history index model for functional responses and functional predictors.

Usage

```
historyIndexSparse(Y, X, Lag = NULL, optnsY = NULL, optnsX = NULL)
```

Arguments

Y	A list which contains functional responses in the form of a list LY and the time points LT at which they are observed (i.e., list(Ly = LY,Lt = LT)).
X	A list of lists which contains the observed functional predictors list Lxj and the time points list Ltj at which they are observed. It needs to be of the form `list(list(Ly = Lx1,Lt = Lxt1),list(Ly = Lx2,Lt = Lxt2),...)`.
Lag	A length `length(X)` vector denoting the lags for all predictors.
optnsY	A list of options control parameters for the response specified by `list(name=value)`. See 'Details' in FPCA.
optnsX	A list of options control parameters for the predictors specified by `list(name=value)`. See 'Details' in FPCA.

Details

The functional history index model is defined as $E[Y(t)|X_1(t),\cdots,X_p(t)] = \beta_0(t)+\sum_{j=1}^{p}\beta_j(t)\int_0^{\Delta_j}\gamma_j(s)X_j(t-s)ds$ for $t \in [\max_j\{\Delta_j\},T]$ with a suitable $T > 0$. Write $\alpha_j(t,s) = \beta_j(t)\gamma_j(s)$. It becomes $E[Y(t)|X_1(t),\cdots,X_p(t)] = \beta_0(t)+\sum_{j=1}^{p}\int_0^{\Delta_j}\alpha_j(t,s)X_j(t-s)ds$. For more details we refer to *Şentürk, D. and Müller, H.G., (2010). Functional varying coefficient models for longitudinal data. Journal of the American Statistical Association, 105(491), pp.1256-1264.*

Value

A list of the following:

beta0	a vector of `length(workGridY)` representing the fitted $\beta_0(t)$.
beta1	a list of vectors with the j-th element representing the fitted $\beta_j(t)$.
gamma	a list of vectors with the j-th element representing the fitted $\gamma_j(s)$.
workGridY	a vetor representing the working grid for the response.
workGridLag	a list of vectors with the j-th element representing the working grid in $[0,\Delta_j]$.

References

Şentürk, D. and Müller, H.G., (2010). Functional varying coefficient models for longitudinal data. Journal of the American Statistical Association, 105(491), pp.1256-1264. Yao, F., Müller, H.G., Wang, J.L. (2005). Functional linear regression analysis for longitudinal data. Annals of Statistics 33, 2873–2903. Hall, P., Horowitz, J.L. (2007). Methodology and convergence rates for functional linear regression. The Annals of Statistics, 35(1), 70–91.

Examples

```
set.seed(1)
### functional covariate X(t) ###
phi1 <- function(t) sin(pi*t / 5) / sqrt(5)
phi2 <- function(t) cos(pi*t / 5) / sqrt(5)
lambdaX <- c(10, 5)
n <- 150
N <- 101
Xi <- matrix(rnorm(2*n), nrow = n, ncol = 2)
denseLt <- list()
denseLy <- list()
t0 <- seq(0, 15, length.out = N)
for (i in 1:n) {
  denseLt[[i]] <- t0
  denseLy[[i]] <- lambdaX[1]*Xi[i, 1]*phi1(t0) + lambdaX[2]*Xi[i, 2]*phi2(t0)
}
denseX0 <- list(Ly = denseLy,Lt = denseLt)

### generate coefficient function gamma(u), beta(u) ###
Lag <- 5
u0 <- t0[t0<=Lag]
t0_out <- t0[t0>=Lag]
gamma_u <- function(u) sqrt(2/5) * cos(pi * u /5)
beta_1 <- function(t) 5*sin(pi*t/10)
beta_0 <- function(t) t^2/2

### functional response Y(t), t in t0_out ###
denseLt <- list()
denseLy <- list()
for (i in 1:n) {
  denseLt[[i]] <- t0_out
  Xt <- denseX0$Ly[[i]]
  Xtu <- t(sapply((1:N)[t0>=Lag], function(j){
    rev(Xt[(j-length(u0)+1):j])  #history index for X[t-u:t]
  }))
  IntGammaXtu <- apply(Xtu, 1, function(v){
    fdapace::trapzRcpp(u0, gamma_u(u0) * v)
  })
  #append 0 in the first length(u0)-1 element(useless info. in our modeling)
 denseLy[[i]] <- beta_0(t0_out) + IntGammaXtu * beta_1(t0_out) + rnorm(length(t0_out), 0, 0.1)
}
denseY <- list(Ly = denseLy, Lt = denseLt)

### functional predictor X(t) (adjust for t0_out) ###
denseLt <- list()
denseLy <- list()
for (i in 1:n){
  denseLt[[i]] <- t0_out
  denseLy[[i]] <- denseX0$Ly[[i]][t0>=Lag]
}
denseX <- list(Ly = denseLy,
               Lt = denseLt)
```

```
### sparify the dense data ###
SparseY <- fdapace::Sparsify(samp = t(sapply(denseY$Ly, function(v) v)),
                    pts = t0_out,
                    sparsity = sample(10:20, n, replace = TRUE)
)
SparseX <- fdapace::Sparsify(samp = t(sapply(denseX$Ly, function(v) v)),
                    pts = t0_out,
                    sparsity = sample(10:20, n, replace = TRUE))

### model fitting for sparse case ###
Fit_result <- historyIndexSparse(Y = SparseY, X = list(X = SparseX), Lag = Lag)
Fit_result$beta0
Fit_result$beta1
Fit_result$gamma
```

`ptFCReg` *Functional concurrent regression using pointwise multiple linear regression.*

Description

Functional concurrent regression using pointwise multiple linear regression.

Usage

```
ptFCReg(tGrid, dat)
```

Arguments

`tGrid`	A vector of length m with the input time points.
`dat`	A list of input functional/scalar covariates. Each field corresponds to a functional (a matrix) or scalar (a vector) variable. The last entry is assumed to be the functional response if no entry is names `'Y'`. If a field corresponds to a functional variable, it should be an n-by-m matrix, where each row holds the observations for one subject on the common grid `tGrid`. If a field corresponds to a scalar covariate, it should be a vector of length n.

Value

A list containing the following fields:

beta0 A vector containing the time-varying intercept evaluated on `tGrid`.

beta A matrix for the concurrent regression effects, where rows correspond to different predictors and columns to different time points in `tGrid`.

tGrid The input `tGrid`.

R2 A vector of the time-varying $R^2(t)$, evaluated at t in `tGrid`.

Ldf A list holding the input data, each element of which is a data frame holding the data observed at one element of `tGrid`.

Examples

```
set.seed(1)
n <- 50
nGridIn <- 101
tGrid <- seq(0, 1, length.out=nGridIn) # Functional data support
muX1 <- tGrid * 2 # mean function for X_1
sigma <- 1
beta0 <- 0
beta <- rbind(cos(tGrid), 1.5 + sin(tGrid))
Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(muX1, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
Y <- t(sapply(seq_len(n), function(i) {
  beta0 + beta[1,] * X_1[i, ] + beta[2,] * Z[i, 2] + epsilon[i]
}))
dat <- list(X1=X_1, Z1=Z[, 2], Y=Y)
res <- ptFCReg(tGrid = tGrid, dat = dat)
```

---

smPtFCRegCoef *Smooth the concurrent effects functions in a* ptFCReg *object using local linear regression. The local linear regression is implemented using the function* Lwls1D.

Description

Smooth the concurrent effects functions in a ptFCReg object using local linear regression. The local linear regression is implemented using the function Lwls1D.

Usage

```
smPtFCRegCoef(object, bw, kernel_type)
```

Arguments

object	An object of class ptFCReg returned by the function ptFCReg.
bw	Scalar holding the bandwidth.
kernel_type	Character holding the kernel type (see Lwls1D for supported kernels).

Value

An object of class ptFCReg, where the fields beta0 and beta hold the smoothed intercept functions and concurrent effects functions, respectively. See ptFCReg for a complete list of the fields.

Examples

```
set.seed(1)
n <- 50
nGridIn <- 101
tGrid <- seq(0, 1, length.out=nGridIn) # Functional data support
muX1 <- tGrid * 2 # mean function for X_1
sigma <- 1
beta0 <- 0
beta <- rbind(cos(tGrid), 1.5 + sin(tGrid))
Z <- MASS::mvrnorm(n, rep(0, 2), diag(2))
X_1 <- Z[, 1, drop=FALSE] %*% matrix(1, 1, nGridIn) + matrix(muX1, n, nGridIn, byrow=TRUE)
epsilon <- rnorm(n, sd=sigma)
Y <- t(sapply(seq_len(n), function(i) {
  beta0 + beta[1,] * X_1[i, ] + beta[2,] * Z[i, 2] + epsilon[i]
}))
dat <- list(X1=X_1, Z1=Z[, 2], Y=Y)
res <- ptFCReg(tGrid = tGrid, dat = dat)
smres <- smPtFCRegCoef(res, bw = 2.5 / (nGridIn-1), kernel_type = 'epan')
```

Index