Package 'goffda'

August 26, 2022

Type Package

Title Goodness-of-Fit Tests for Functional Data

Version 0.1.1

Date 2022-08-26

Description Implementation of several goodness-of-fit tests for functional data. Currently, mostly related with the functional linear model with functional/scalar response and functional/scalar predictor. The package allows for the replication of the data applications considered in García-Portugués, Álvarez-Liébana, Álvarez-Pérez and González-Manteiga (2021) <doi:10.1111/sjos.12486>.

License GPL-3

LazyData true

Depends R (>= 3.6.0), Rcpp

Imports fda.usc, glmnet, ks

Suggests microbenchmark, knitr, viridisLite, rmarkdown

LinkingTo Rcpp, RcppArmadillo

URL https://github.com/egarpor/goffda

BugReports https://github.com/egarpor/goffda

Encoding UTF-8

RoxygenNote 7.2.0

NeedsCompilation yes

Author Eduardo García-Portugués [aut, cre] (<https://orcid.org/0000-0002-9224-4111>), Javier Álvarez-Liébana [aut] (<https://orcid.org/0000-0003-0671-3856>), Gonzalo Álvarez-Pérez [ctb], Manuel Febrero-Bande [ctb]

Maintainer Eduardo García-Portugués <edgarcia@est-econ.uc3m.es>

Repository CRAN

Date/Publication 2022-08-26 19:42:02 UTC

R topics documented:

---

goffda-package **goffda** – *Goodness-of-Fit Tests for Functional Data*

Description

Implementation of several goodness-of-fit tests for functional data. Currently, mostly related with the functional linear model with functional/scalar response and functional/scalar predictor. The package allows for the replication of the data applications considered in García-Portugués, Álvarez-Liébana, Álvarez-Pérez and González-Manteiga (2021) <doi:10.1111/sjos.12486>.

Author(s)

Eduardo García-Portugués and Javier Álvarez-Liébana.

References

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

García-Portugués, E., González-Manteiga, W. and Febrero-Bande, M. (2014). A goodness-of-fit test for the functional linear model with scalar response. *Journal of Computational and Graphical Statistics*, 23(3):761–778. doi: 10.1080/10618600.2013.812519

`aemet_temp` *AEMET daily temperatures during 1974–2013*

Description

Series of daily temperatures of 73 Spanish weather stations during the 40-year period 1974–2013.

Usage

```
aemet_temp
```

Format

A list with the following entries:

temp an `fdata` with 2892 temperature (in Celsius degrees) curves, discretized on 365 equispaced grid points (days) on $[0.5, 364.5]$. Each curve corresponds to the yearly records of a weather station.

df a dataframe with metadata for each curve:

- `ind`: identifier of the weather station.
- `name`: name of the weather station.
- `year`: year of the observation.

Details

For consistency with the `fda.usc-package`'s `aemet` dataset, the names and identifiers of the 73 weather stations are the same as in that dataset. Only a minor fix has been applied to the "A CORUÑA/ALVEDRO" station, whose identifier was the same as the "A CORUÑA" station, `"1387"`. The former was set to `"1387A"`.

Information about the province, altitude, longitude, and latitude of each weather station can be retrieved in `df` from the `fda.usc-package`'s `aemet` dataset.

The dataset is a curated version of a larger database of 115 stations. It excludes stations with inconsistent records or that were relocated, closed, or opened during the 40-year period. There are 9 stations with missing years. The total of missing years is 28.

In leap years, the daily-average temperature is computed as the average of February 28th and 29th.

Author(s)

Original data processing scripts by Manuel Febrero-Bande and Manuel Oviedo de la Fuente. Adaptations by Eduardo García-Portugués.

Source

The data was retrieved from the FTP of the Meteorological State Agency of Spain (AEMET) in 2014 using a processing script by the authors of the `fda.usc-package`.

References

Febrero-Bande, M. and Oviedo de la Fuente, M. (2012). Statistical Computing in Functional Data Analysis: The R Package fda.usc. *Journal of Statistical Software*, 51(4):1–28. doi: 10.18637/jss.v051.i04

Examples

```
## Data splitting

# Load raw data
data("aemet_temp")

# Partition the dataset in the first and last 20 years
with(aemet_temp, {
  ind_pred <- which((1974 <= df$year) & (df$year <= 1993))
  ind_resp <- which((1994 <= df$year) & (df$year <= 2013))
  aemet_temp_pred <<- list("df" = df[ind_pred, ], "temp" = temp[ind_pred])
  aemet_temp_resp <<- list("df" = df[ind_resp, ], "temp" = temp[ind_resp])
})

# Average the temperature on each period
mean_aemet <- function(x) {
  m <- tapply(X = 1:nrow(x$temp$data), INDEX = x$df$ind,
              FUN = function(i) colMeans(x$temp$data[i, , drop = FALSE],
                                         na.rm = TRUE))
 x$temp$data <- do.call(rbind, m)
 return(x$temp)
}

# Build predictor and response fdatas
aemet_temp_pred <- mean_aemet(aemet_temp_pred)
aemet_temp_resp <- mean_aemet(aemet_temp_resp)

# Plot
old_par <- par(mfrow = c(1, 2))
plot(aemet_temp_pred)
plot(aemet_temp_resp)
par(old_par)

# Average daily temperatures
day_avg_pred <- func_mean(aemet_temp_pred)
day_avg_resp <- func_mean(aemet_temp_resp)

# Average yearly temperatures
avg_year_pred <- rowMeans(aemet_temp_pred$data)
avg_year_resp <- rowMeans(aemet_temp_resp$data)

## Test the linear model with functional response and predictor

(comp_flmfr <- flm_test(X = aemet_temp_pred, Y = aemet_temp_resp,
                        est_method = "fpcr_l1s"))
beta0 <- diag(rep(1, length(aemet_temp_pred$argvals)))
```

```
(simp_flmfr <- flm_test(X = aemet_temp_pred, Y = aemet_temp_resp,
                        beta0 = beta0, est_method = "fpcr_l1s"))

# Visualize estimation
filled.contour(x = aemet_temp_pred$argvals, y = aemet_temp_resp$argvals,
               z = comp_flmfr$fit_flm$Beta_hat,
               color.palette = viridisLite::viridis, nlevels = 20)

## Test the linear model with scalar response and functional predictor

(comp_flmsr <- flm_test(X = aemet_temp_pred, Y = avg_year_resp,
                        est_method = "fpcr_l1s"))
(simp_flmsr <- flm_test(X = aemet_temp_pred, Y = avg_year_resp,
                        beta0 = 1 / 365, est_method = "fpcr_l1s"))

# Visualize estimation
plot(aemet_temp_pred$argvals, comp_flmsr$fit_flm$Beta_hat, type = "l",
     ylim = c(0, 30 / 365))
abline(h = 1 / 365, col = 2)

## Test the linear model with functional response and scalar predictor

(comp_frsp <- flm_test(X = avg_year_pred, Y = aemet_temp_resp))
(simp_frsp <- flm_test(X = avg_year_pred, Y = aemet_temp_resp, beta0 = 1))

## Test the linear model with scalar response and predictor

(comp_srsp <- flm_test(X = avg_year_pred, Y = avg_year_resp))
(simp_srsp <- flm_test(X = avg_year_pred, Y = avg_year_resp, beta0 = 1))
```

---

cv_glmnet *Fitting of regularized linear models*

Description

Convenience function for fitting multivariate linear models with multivariate response by relying on cv.glmnet from the glmnet-package. The function fits the multivariate linear model

$$\mathbf{Y} = \mathbf{XB} + \mathbf{E},$$

where $\mathbf{X}$ is a p-dimensional vector, $\mathbf{Y}$ and $\mathbf{E}$ are two q-dimensional vectors, and $\mathbf{B}$ is a $p \times q$ matrix.

If $\mathbf{X}$ and $\mathbf{Y}$ are *centered* (i.e., have zero-mean columns), the function estimates $\mathbf{B}$ by solving, for the sample $(\mathbf{X}_1, \mathbf{Y}_1), \ldots, (\mathbf{X}_n, \mathbf{Y}_n)$, the elastic-net optimization problem

$$\min_{\mathbf{B} \in R^{q \times p}} \frac{1}{2n} \sum_{i=1}^{n} \|\mathbf{Y}_i - \mathbf{X}_i \mathbf{B}\|^2 + \lambda \left[(1-\alpha) \|\mathbf{B}\|_{\mathrm{F}}^2 / 2 + \alpha \sum_{j=1}^{p} \|\mathbf{B}_j\|_2 \right],$$

where $\|\mathbf{B}\|_{\mathrm{F}}$ stands for the Frobenious norm of the matrix $\mathbf{B}$ and $\|\mathbf{B}_j\|_2$ for the Euclidean norm of the j-th *row* of $\mathbf{B}$. The choice $\alpha = 0$ in the elastic-net penalization corresponds to ridge regression,

whereas $\alpha = 1$ yields a lasso-type estimator. The unpenalized least-squares estimator is obtained with $\lambda = 0$.

Usage

```
cv_glmnet(x, y, alpha = c("lasso", "ridge")[1], lambda = NULL,
  intercept = TRUE, thresh = 1e-10, cv_1se = TRUE, cv_nlambda = 50,
  cv_folds = NULL, cv_grouped = FALSE, cv_lambda = 10^seq(2, -3,
  length.out = cv_nlambda), cv_second = TRUE, cv_tol_second = 0.025,
  cv_log10_exp = c(-0.5, 3), cv_thresh = 1e-05, cv_parallel = FALSE,
  cv_verbose = FALSE, ...)
```

Arguments

x	input matrix of size c(n, p), or a vector of length n.
y	response matrix of size c(n, q), or a vector of length n.
alpha	elastic net mixing argument in glmnet, with $0 \leq \alpha \leq 1$. Alternatively, a character string indicating whether the "ridge" ($\alpha = 0$) or "lasso" ($\alpha = 1$) fit is to be performed.
lambda	scalar giving the regularization parameter λ. If NULL (default), the optimal λ is searched by cross-validation. If lambda is provided, then cross-validation is skipped and the fit is performed for the given lambda.
intercept	flag passed to the intercept argument in glmnet to indicate if the intercept should be fitted (default; does not assume that the data is centered) or set to zero (the optimization problem above is solved as-is). Defaults to TRUE.
thresh	convergence threshold of the coordinate descending algorithm, passed to the thresh argument in glmnet. Defaults to 1e-10.
cv_1se	shall the *optimal* lambda be the lambda.1se, as returned by cv.glmnet? This favors sparser fits. If FALSE, then the optimal lambda is lambda.min, the minimizer of the cross-validation loss. Defaults to TRUE.
cv_nlambda	the length of the sequence of λ values, passed to the nlambda argument in cv.glmnet for the cross-validation search. Defaults to 50.
cv_folds	number of folds to perform cross-validation. If NULL (the default), then cv_folds <- n internally, that is, leave-one-out cross-validation is performed. Passed to the nfolds argument in cv.glmnet.
cv_grouped	passed to the grouped argument in cv.glmnet. Defaults to FALSE.
cv_lambda	passed to the lambda argument in cv.glmnet. Defaults to 10^seq(2, -3, length.out = cv_nlambda).
cv_second	flag to perform a second cross-validation search if the optimal λ was found at the extremes of the first λ sequence (indicating that the minimum may not be reliable). Defaults to TRUE.
cv_tol_second	tolerance for performing a second search if second = TRUE. If the minimum is found at the 100 * cv_tol_second lower/upper percentile of search interval, then the search interval is expanded for a second search. Defaults to 0.025.

cv_log10_exp	expansion of the λ sequence if the minimum is found close to its *upper* extreme. If that is the case, the sequence for the is set as 10^(log10(lambda_min) + cv_log10_exp), where lambda_min is the minimum obtained in the first sequence. If the minimum is found close to the lower extreme of the sequence, then -rev(cv_log10_exp) is considered. Defaults to c(-0.5, 5).
cv_thresh	convergence threshold used during cross-validation in cv.glmnet. Defaults to 1e-5.
cv_parallel	passed to the parallel argument in cv.glmnet. Defaults to FALSE.
cv_verbose	flag to display information about the cross-validation search with plots and messages. More useful if second = TRUE. Defaults to FALSE.
...	further parameters to be passed to glmnet to perform the final model fit.

Details

If $\alpha = 1$, then the lasso-type fit shrinks to zero, *simultaneously*, all the elements of certain rows of $\mathbf{B}$, thus helping the selection of the p most influential variables in $\mathbf{X}$ for explaining/predicting $\mathbf{Y}$.

The function first performs a cross-validation search for the optimal λ if lambda = NULL (using cv_thresh to control the convergence threshold). After the optimal penalization parameter is determined, a second fit (now with convergence threshold thresh) using the default λ sequence in glmnet is performed. The final estimate is obtained via predict.glmnet from the optimal λ determined in the first step.

Due to its cross-validatory nature, cv_glmnet can be computationally demanding. Approaches for reducing the computation time include: considering a smaller number of folds than n, such as cv_folds = 10 (but will lead to random partitions of the data); decrease the tolerance of the coordinate descending algorithm by increasing cv_thresh; reducing the number of candidate λ values with nlambda; setting second = FALSE to avoid a second cross-validation; or considering cv_parallel = TRUE to use a parallel backend (must be registered before hand; see examples).

By default, the λ sequence is used with *standardized* $\mathbf{X}$ and $\mathbf{Y}$ (both divided by their columnwise variances); see glmnet and the standardized argument. Therefore, the optimal selected λ value assumes standardization and must be used with care if the variables are not standardized. For example, when using the ridge analytical solution, a prior change of scale that depends on q needs to be done. See the examples for the details.

Value

A list with the following entries:

beta_hat	the estimated $\mathbf{B}$, a matrix of size c(p, q).
lambda	the optimal λ obtained by cross-validation and according to cv_1se.
cv	if lambda = NULL, the result of the cross-validation search for the optimal λ. Otherwise, NULL.
fit	the glmnet fit, computed with thresh threshold and with an automatically chosen λ sequence.

Author(s)

Eduardo García-Portugués. Initial contributions by Gonzalo Álvarez-Pérez.

References

Friedman, J., Hastie, T. and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22. doi: 10.18637/jss.v033.i01

Examples

```
## Quick example for multivariate linear model with multivariate response

# Simulate data
n <- 100
p <- 10; q <- 5
set.seed(123456)
x <- matrix(rnorm(n * p, sd = rep(1:p, each = n)), nrow = n, ncol = p)
e <- matrix(rnorm(n * q, sd = rep(q:1, each = n)), nrow = n, ncol = q)
beta <- matrix(((1:p - 1) / p)^2, nrow = p, ncol = q)
y <- x %*% beta + e

# Fit lasso (model with intercept, the default)
cv_glmnet(x = x, y = y, alpha = "lasso", cv_verbose = TRUE)$beta_hat

## Multivariate linear model with multivariate response

# Simulate data
n <- 100
p <- 10; q <- 5
set.seed(123456)
x <- matrix(rnorm(n * p, sd = rep(1:p, each = n)), nrow = n, ncol = p)
e <- matrix(rnorm(n * q, sd = rep(q:1, each = n)), nrow = n, ncol = q)
beta <- matrix(((1:p - 1) / p)^2, nrow = p, ncol = q)
y <- x %*% beta + e

# Fit ridge
cv0 <- cv_glmnet(x = x, y = y, alpha = "ridge", intercept = FALSE,
                 cv_verbose = TRUE)
cv0$beta_hat

# Same fit for the chosen lambda
cv_glmnet(x = x, y = y, alpha = "ridge", lambda = cv0$lambda,
          intercept = FALSE)$beta_hat

# Fit lasso (model with intercept, the default)
cv1 <- cv_glmnet(x = x, y = y, alpha = "lasso", cv_verbose = TRUE)
cv1$beta_hat

# Use cv_1se = FALSE
cv1_min <- cv_glmnet(x = x, y = y, alpha = "lasso", cv_verbose = TRUE,
                     cv_1se = FALSE)

# Compare it with ridge analytical solution. Observe the factor in lambda,
# necessary since lambda is searched for the standardized data. Note also
# that, differently to the case q = 1, no standardarization with respect to
# y happens
```

```
sd_x <- apply(x, 2, function(x) sd(x)) * sqrt((n - 1) / n)
cv_glmnet(x = x, y = y, alpha = "ridge", lambda = cv0$lambda,
          thresh = 1e-20, intercept = FALSE)$beta_hat
solve(crossprod(x) + diag(cv0$lambda * sd_x^2 * n)) %*% t(x) %*% y

# If x is standardized, the match between glmnet and usual ridge
# analytical expression does not require scaling of lambda
x_std <- scale(x, scale = sd_x, center = TRUE)
cv_glmnet(x = x_std, y = y, alpha = "ridge", lambda = cv0$lambda,
          intercept = FALSE, thresh = 1e-20)$beta_hat
solve(crossprod(x_std) + diag(rep(cv0$lambda * n, p))) %*% t(x_std) %*% y

## Simple linear model

# Simulate data
n <- 100
p <- 1; q <- 1
set.seed(123456)
x <- matrix(rnorm(n * p), nrow = n, ncol = p)
e <- matrix(rnorm(n * q), nrow = n, ncol = q)
beta <- 2
y <- x * beta + e

# Fit by ridge (model with intercept, the default)
cv0 <- cv_glmnet(x = x, y = y, alpha = "ridge", cv_verbose = TRUE)
cv0$beta_hat
cv0$intercept

# Comparison with linear model with intercept
lm(y ~ 1 + x)$coefficients

# Fit by ridge (model without intercept)
cv0 <- cv_glmnet(x = x, y = y, alpha = "ridge", cv_verbose = TRUE,
                 intercept = FALSE)
cv0$beta_hat

# Comparison with linear model without intercept
lm(y ~ 0 + x)$coefficients

# Same fit for the chosen lambda (and without intercept)
cv_glmnet(x = x, y = y, alpha = "ridge", lambda = cv0$lambda,
          intercept = FALSE)$beta_hat

# Same for lasso (model with intercept, the default)
cv1 <- cv_glmnet(x = x, y = y, alpha = "lasso")
cv1$beta_hat

## Multivariate linear model (p = 3, q = 1)

# Simulate data
n <- 50
p <- 10; q <- 1
set.seed(123456)
```

```
x <- matrix(rnorm(n * p, mean = 1, sd = rep(1:p, each = n)),
            nrow = n, ncol = p)
e <- matrix(rnorm(n * q), nrow = n, ncol = q)
beta <- ((1:p - 1) / p)^2
y <- x %*% beta + e

# Fit ridge (model without intercept)
cv0 <- cv_glmnet(x = x, y = y, alpha = "ridge", intercept = FALSE,
                 cv_verbose = TRUE)
cv0$beta_hat

# Same fit for the chosen lambda
cv_glmnet(x = x, y = y, alpha = "ridge", lambda = cv0$lambda,
          intercept = FALSE)$beta_hat

# Compare it with ridge analytical solution. Observe the factor in lambda,
# necessary since lambda is searched for the standardized data
sd_x <- apply(x, 2, function(x) sd(x)) * sqrt((n - 1) / n)
sd_y <- sd(y) * sqrt((n - 1) / n)
cv_glmnet(x = x, y = y, alpha = "ridge", lambda = cv0$lambda,
          intercept = FALSE, thresh = 1e-20)$beta_hat
solve(crossprod(x) + diag(cv0$lambda * sd_x^2 / sd_y * n)) %*% t(x) %*% y

# If x and y are standardized, the match between glmnet and usual ridge
# analytical expression does not require scaling of lambda
x_std <- scale(x, scale = sd_x, center = TRUE)
y_std <- scale(y, scale = sd_y, center = TRUE)
cv_glmnet(x = x_std, y = y_std, alpha = "ridge", lambda = cv0$lambda,
          intercept = FALSE, thresh = 1e-20)$beta_hat
solve(crossprod(x_std) + diag(rep(cv0$lambda * n, p))) %*% t(x_std) %*% y_std

# Fit lasso (model with intercept, the default)
cv1 <- cv_glmnet(x = x, y = y, alpha = "lasso", cv_verbose = TRUE)
cv1$beta_hat

# ## Parallelization
#
# # Parallel
# doMC::registerDoMC(cores = 2)
# microbenchmark::microbenchmark(
# cv_glmnet(x = x, y = y, nlambda = 100, cv_parallel = TRUE),
# cv_glmnet(x = x, y = y, nlambda = 100, cv_parallel = FALSE),
# times = 10)
```

`elem-flmfr` *Covariate, error, and kernel of a functional linear model with functional response*

Description

Simulation of X, a random variable in the Hilbert space of square-integrable functions in $[a, b]$, $L^2([a, b])$, and ε, a random variable in $L^2([c, d])$. Together with the bivariate kernel β, they are the necessary elements for sampling a Functional Linear Model with Functional Response (FLMFR):

$$Y(t) = \int_a^b X(s)\beta(s,t)ds + \varepsilon(t).$$

The next functions sample X and ε, and construct β, using different proposals in the literature:

- `r_cm2013_flmfr` is based on the numerical example given in Section 3 of Crambes and Mas (2013). Termed as S1 in Section 2 of García-Portugués et al. (2021).
- `r_ik2018_flmfr` is based on the numerical example given in Section 4 of Imaizumi and Kato (2018), but zeroing the first Functional Principal Components (FPC) coefficients of β (so the first FPC are not adequate for estimation). S3 in Section 2 of García-Portugués et al. (2021).
- `r_gof2021_flmfr` gives a numerical example in Section 2 of García-Portugués et al. (2021), denoted therein as S2.

Usage

```
r_cm2013_flmfr(n, s = seq(0, 1, len = 101), t = seq(0, 1, len = 101),
  std_error = 0.15, n_fpc = 50, concurrent = FALSE)

r_ik2018_flmfr(n, s = seq(0, 1, l = 101), t = seq(0, 1, l = 101),
  std_error = 1.5, parameters = c(1.75, 0.8, 2.4, 0.25), n_fpc = 50,
  concurrent = FALSE)

r_gof2021_flmfr(n, s = seq(0, 1, len = 101), t = seq(0, 1, len = 101),
  std_error = 0.35, concurrent = FALSE)
```

Arguments

`n`	number of trajectories to sample.
`s, t`	grid points where functional covariates and responses are valued, respectively.
`std_error`	standard deviation of the random variables involved in the generation of the functional error `error_fdata`. Defaults to `0.15`.
`n_fpc`	number of FPC to be taken into account for the data generation. Must be greater than `4` when `r_ik2018_flmfr` is applied, since the first 4 FPC are null. Defaults to `50`.
`concurrent`	flag to consider a concurrent FLMFR (degenerate case). Defaults to `FALSE`.
`parameters`	vector of parameters, only required for `r_ik2018_flmfr`. Defaults to `c(1.75, 0.8, 2.4, 0.25)`.

Details

Descriptions of the processes X and ε, and of β can be seen in the references.

Value

A list with the following elements:

`X_fdata`	functional covariates, an `fdata` object of length `n`.
`error_fdata`	functional errors, an `fdata` object of length `n`.
`beta`	either the matrix with $\beta(s,t)$ evaluated at the `argvals` of `X_fdata` and `Y_fdata` (if `concurrent = FALSE`) or a vector with $\beta(t)$ evaluated at the `argvals` of `X_fdata` (if `concurrent = TRUE`).

Author(s)

Javier Álvarez-Liébana.

References

Cardot, H. and Mas, A. (2013). Asymptotics of prediction in functional linear regression with functional outputs. *Bernoulli*, 19(5B):2627–2651. doi: 10.3150/12BEJ469

Imaizumi, M. and Kato, K. (2018). PCA-based estimation for functional linear regression with functional responses. *Journal of Multivariate Analysis*, 163:15–36. doi: 10.1016/j.jmva.2017.10.001

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

Examples

```
# FLMFR based on Imaizumi and Kato (2018) adopting different Hilbert spaces
s <- seq(0, 1, l = 201)
t <- seq(2, 4, l = 301)
r_ik2018 <- r_ik2018_flmfr(n = 50, s = s, t = t, std_error = 1.5,
                           parameters = c(1.75, 0.8, 2.4, 0.25), n_fpc = 50)
plot(r_ik2018$X_fdata)
plot(r_ik2018$error_fdata)
image(x = s, y = t, z = r_ik2018$beta, col = viridisLite::viridis(20))

# FLMFR based on Cardot and Mas (2013) adopting different Hilbert spaces
r_cm2013 <- r_cm2013_flmfr(n = 50, s = s, t = t, std_error = 0.15,
                           n_fpc = 50)
plot(r_cm2013$X_fdata)
plot(r_cm2013$error_fdata)
image(x = s, y = t, z = r_cm2013$beta, col = viridisLite::viridis(20))

# FLMFR in García-Portugués et al. (2021) adopting different Hilbert spaces
r_gof2021 <- r_gof2021_flmfr(n = 50, s = s, t = t, std_error = 0.35,
                             concurrent = FALSE)
plot(r_gof2021$X_fdata)
plot(r_gof2021$error_fdata)
image(x = s, y = t, z = r_gof2021$beta, col = viridisLite::viridis(20))

# Concurrent model in García-Portugués et al. (2021)
r_gof2021 <- r_gof2021_flmfr(n = 50, s = s, t = s, std_error = 0.35,
```

```
                                concurrent = TRUE)
plot(r_gof2021$X_fdata)
plot(r_gof2021$error_fdata)
plot(r_gof2021$beta)
```

`flm_est` *Estimation of functional linear models*

Description

Estimation of the linear operator relating a functional predictor X with a functional response Y in the linear model

$$Y(t) = \int_a^b \beta(s,t)X(s)\,\mathrm{d}s + \varepsilon(t),$$

where X is a random variable in the Hilbert space of square-integrable functions in $[a,b]$, $L^2([a,b])$, Y and ε are random variables in $L^2([c,d])$, and $s \in [a,b]$ and $t \in [c,d]$.

The linear, Hilbert–Schmidt, integral operator is parametrized by the bivariate kernel $\beta \in L^2([a,b]) \otimes L^2([c,d])$. Its estimation is done through the truncated expansion of β in the tensor product of the data-driven bases of the Functional Principal Components (FPC) of X and Y, and through the fitting of the resulting multivariate linear model. The FPC basis for X is truncated in p components, while the FPC basis for Y is truncated in q components. Automatic selection of p and q is detailed below.

The particular cases in which either X or Y are *constant* functions give either a scalar predictor or response. The simple linear model arises if both X and Y are scalar, for which β is a constant.

Usage

```
flm_est(X, Y, est_method = "fpcr_l1s", p = NULL, q = NULL,
  thre_p = 0.99, thre_q = 0.99, lambda = NULL, X_fpc = NULL,
  Y_fpc = NULL, compute_residuals = TRUE, centered = FALSE,
  int_rule = "trapezoid", cv_verbose = FALSE, ...)
```

Arguments

`X, Y`	samples of functional/scalar predictors and functional/scalar response. Either `fdata` objects (for functional variables) or vectors of length `n` (for scalar variables).
`est_method`	either `"fpcr"` (Functional Principal Components Regression; FPCR), `"fpcr_l2"` (FPCR with ridge penalty), `"fpcr_l1"` (FPCR with lasso penalty) or `"fpcr_l1s"` (FPCR with lasso-selected FPC). If `X` is scalar, `flm_est` only considers `"fpcr"` as estimation method. See details below. Defaults to `"fpcr_l1s"`.
`p, q`	index vectors indicating the specific FPC to be considered for the truncated bases expansions of `X` and `Y`, respectively. If a single number for `p` is provided, then `p <- 1:max(p)` internally (analogously for `q`) and the first `max(p)` FPC are considered. If `NULL` (default), then a data-driven selection of `p` and `q` is done. See details below.

`thre_p, thre_q` thresholds for the *proportion* of variance that is explained, *at least*, by the first p and q FPC of `X` and `Y`, respectively. These thresholds are employed for an (initial) automatic selection of p and q. Default to `0.99`. `thre_p` (`thre_q`) is ignored if `p` (`q`) is provided.

`lambda` regularization parameter λ for the estimation methods `"fpcr_l2"`, `"fpcr_l1"`, and `"fpcr_l1s"`. If `NULL` (default), it is chosen with `cv_glmnet`.

`X_fpc, Y_fpc` FPC decompositions of `X` and `Y`, as returned by `fpc`. Computed if not provided.

`compute_residuals` whether to compute the fitted values `Y_hat` and its `Y_hat_scores`, and the `residuals` of the fit and its `residuals_scores`. Defaults to `TRUE`.

`centered` flag to indicate if `X` and `Y` have been centered or not, in order to avoid their recentering. Defaults to `FALSE`.

`int_rule` quadrature rule for approximating the definite unidimensional integral: trapezoidal rule (`int_rule = "trapezoid"`) and extended Simpson rule (`int_rule = "Simpson"`) are available. Defaults to `"trapezoid"`.

`cv_verbose` flag to display information about the estimation procedure (passed to `cv_glmnet`). Defaults to `FALSE`.

`...` further parameters to be passed to `cv_glmnet` (and then to `cv.glmnet`) such as `cv_1se`, `cv_nlambda` or `cv_parallel`, among others.

Details

`flm_est` deals seamlessly with either functional or scalar inputs for the predictor and response. In the case of scalar inputs, the corresponding dimension-related arguments (`p`, `q`, `thre_p` or `thre_q`) will be ignored as in these cases either $p = 1$ or $q = 1$.

The function translates the functional linear model into a multivariate model with multivariate response and then estimates the $p \times q$ matrix of coefficients of β in the tensor basis of the FPC of `X` and `Y`. The following estimation methods are implemented:

- `"fpcr"`: Functional Principal Components Regression (FPCR); see details in Ramsay and Silverman (2005).
- `"fpcr_l2"`: FPCR, with ridge penalty on the associated multivariate linear model.
- `"fpcr_l1"`: FPCR, with lasso penalty on the associated multivariate linear model.
- `"fpcr_l1s"`: FPCR, with FPC selected by lasso regression on the associated multivariate linear model.

The last three methods are explained in García-Portugués et al. (2021).

The p FPC of `X` and q FPC of `Y` are determined as follows:

- If `p = NULL`, then `p` is set as `p_thre <- 1:j_thre`, where `j_thre` is the j-th FPC of `X` for which the cumulated proportion of explained variance is greater than `thre_p`. If `p != NULL`, then `p_thre <- p`.
- If `q = NULL`, then the same procedure is followed with `thre_q`, resulting `q_thre`.

Once p_thre and q_thre have been obtained, the methods "fpcr_l1" and "fpcr_l1s" perform a second selection of the FPC that are effectively considered in the estimation of β. This subset of FPC (of p_thre) is encoded in p_hat. No further selection of FPC is done for the methods "fpcr" and "fpcr_l2".

The flag compute_residuals controls if Y_hat, Y_hat_scores, residuals, and residuals_scores are computed. If FALSE, they are set to NULL. Y_hat equals $\hat{Y}_i(t) = \int_a^b \hat{\beta}(s,t)X_i(s)\,\mathrm{d}s$ and residuals stands for $\hat{\varepsilon}_i(t) = Y_i(t) - \hat{Y}_i(t)$, both for $i = 1, \ldots, n$. Y_hat_scores and residuals_scores are the $n \times q$ matrices of coefficients (or scores) of these functions in the FPC of Y.

Missing values on X and Y are automatically removed.

Value

A list with the following entries:

Beta_hat	estimated β, a matrix with values $\hat{\beta}(s,t)$ evaluated at the grid points for X and Y. Its size is c(length(X$argvals), length(Y$argvals)).
Beta_hat_scores	the matrix of coefficients of Beta_hat (resulting from projecting it into the tensor basis of X_fpc and Y_fpc), with dimension c(p_hat, q_thre).
H_hat	hat matrix of the associated fitted multivariate linear model, a matrix of size c(n, n). NULL if est_method = "fpcr_l1", since lasso estimation does not provide it explicitly.
p_thre	index vector indicating the FPC of X considered for estimating the model. Chosen by thre_p or equal to p if given.
p_hat	index vector of the FPC considered by the methods "fpcr_l1" and "fpcr_l1s" methods after further selection of the FPC considered in p_thre. For methods "fpcr" and "fpcr_l2", p_hat equals p_thre.
q_thre	index vector indicating the FPC of Y considered for estimating the model. Chosen by thre_q or equal to q if given. Note that zeroing by lasso procedure only affects in the rows.
est_method	the estimation method employed.
Y_hat	fitted values, either an fdata object or a vector, depending on Y.
Y_hat_scores	the matrix of coefficients of Y_hat, with dimension c(n, q_thre).
residuals	residuals of the fitted model, either an fdata object or a vector, depending on Y.
residuals_scores	the matrix of coefficients of residuals, with dimension c(n, q_thre).
X_fpc, Y_fpc	FPC of X and Y, as returned by fpc with n_fpc = n.
lambda	regularization parameter λ used for the estimation methods "fpcr_l2", "fpcr_l1", and "fpcr_l1s".
cv	cross-validation object returned by cv_glmnet.

Author(s)

Eduardo García-Portugués and Javier Álvarez-Liébana.

References

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

Ramsay, J. and Silverman, B. W. (2005). *Functional Data Analysis*. Springer-Verlag, New York.

Examples

```
## Quick example of functional response and functional predictor

# Generate data
set.seed(12345)
n <- 50
X_fdata <- r_ou(n = n, t = seq(0, 1, l = 201), sigma = 2)
epsilon <- r_ou(n = n, t = seq(0, 1, l = 201), sigma = 0.5)
Y_fdata <- 2 * X_fdata + epsilon

# Lasso-selection FPCR (p and q are estimated)
flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s")

## Functional response and functional predictor

# Generate data
set.seed(12345)
n <- 50
X_fdata <- r_ou(n = n, t = seq(0, 1, l = 201), sigma = 2)
epsilon <- r_ou(n = n, t = seq(0, 1, l = 201), sigma = 0.5)
Y_fdata <- 2 * X_fdata + epsilon

# FPCR (p and q are estimated)
fpcr_1 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr")
fpcr_1$Beta_hat_scores
fpcr_1$p_thre
fpcr_1$q_thre

# FPCR (p and q are provided)
fpcr_2 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr",
                  p = c(1, 5, 2, 7), q = 2:1)
fpcr_2$Beta_hat_scores
fpcr_2$p_thre
fpcr_2$q_thre

# Ridge FPCR (p and q are estimated)
l2_1 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l2")
l2_1$Beta_hat_scores
l2_1$p_hat

# Ridge FPCR (p and q are provided)
l2_2 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l2",
                p = c(1, 5, 2, 7), q = 2:1)
l2_2$Beta_hat_scores
```

```
l2_2$p_hat

# Lasso FPCR (p and q are estimated)
l1_1 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1")
l1_1$Beta_hat_scores
l1_1$p_thre
l1_1$p_hat

# Lasso estimator (p and q are provided)
l1_2 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1",
                p = c(1, 5, 2, 7), q = 2:1)
l1_2$Beta_hat_scores
l1_2$p_thre
l1_2$p_hat

# Lasso-selection FPCR (p and q are estimated)
l1s_1 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s")
l1s_1$Beta_hat_scores
l1s_1$p_thre
l1s_1$p_hat

# Lasso-selection FPCR (p and q are provided)
l1s_2 <- flm_est(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s",
                 p = c(1, 5, 2, 7), q = 1:4)
l1s_2$Beta_hat_scores
l1s_2$p_thre
l1s_2$p_hat

## Scalar response

# Generate data
set.seed(12345)
n <- 50
beta <- r_ou(n = 1, t = seq(0, 1, l = 201), sigma = 0.5, x0 = 3)
X_fdata <- fdata_cen(r_ou(n = n, t = seq(0, 1, l = 201), sigma = 2))
epsilon <- rnorm(n, sd = 0.25)
Y <- drop(inprod_fdata(X_fdata1 = X_fdata, X_fdata2 = beta)) + epsilon

# FPCR
fpcr_4 <- flm_est(X = X_fdata, Y = Y, est_method = "fpcr")
fpcr_4$p_hat

# Ridge FPCR
l2_4 <- flm_est(X = X_fdata, Y = Y, est_method = "fpcr_l2")
l2_4$p_hat

# Lasso FPCR
l1_4 <- flm_est(X = X_fdata, Y = Y, est_method = "fpcr_l1")
l1_4$p_hat

# Lasso-selection FPCR
l1s_4 <- flm_est(X = X_fdata, Y = Y, est_method = "fpcr_l1s")
l1s_4$p_hat
```

```
## Scalar predictor

# Generate data
set.seed(12345)
n <- 50
X <- rnorm(n)
epsilon <- r_ou(n = n, t = seq(0, 1, l = 201), sigma = 0.5)
beta <- r_ou(n = 1, t = seq(0, 1, l = 201), sigma = 0.5, x0 = 3)
beta$data <- matrix(beta$data, nrow = n, ncol = ncol(beta$data),
                    byrow = TRUE)
Y_fdata <- beta * X + epsilon

# FPCR
fpcr_4 <- flm_est(X = X, Y = Y_fdata, est_method = "fpcr")
plot(beta, col = 2)
lines(beta$argvals, drop(fpcr_4$Beta_hat))
```

`flm_stat` *Projected Cramér–von Mises test statistic for the goodness-of-fit test of functional linear models*

Description

Computation of the Projected Cramér–von Mises (PCvM) test statistic and its associated $\mathbf{A}_\bullet$ matrix. For a sample of functional covariates $X_1, \ldots, X_n$, the test statistic is computed from $\mathbf{x}_{1,p}, \ldots, \mathbf{x}_{n,p}$, the coefficients (scores) of the sample in a p-truncated basis expansion, such as Functional Principal Components (FPC).

The PCvM statistic is defined as

$$\mathrm{PCvM}_{n,p,q} = c \cdot \mathrm{tr}(\hat{\mathbf{E}}_q' \mathbf{A}_\bullet \hat{\mathbf{E}}_q)$$

where

$$c = 2\pi^{(p+q)/2-1}/(q\Gamma(p/2)\Gamma(q/2)n^2),$$

$\hat{\mathbf{E}}_q$ is the $n \times q$ matrix of multivariate residuals, and $\mathbf{A}_\bullet$ is a $n \times n$ matrix whose ij-th element is $\sum_{r=1}^{n} A_{ijr}$, for A_{ijr} depending on $(\mathbf{x}_{i,p}, \mathbf{x}_{j,p}, \mathbf{x}_{r,p})$. Its exact expression can be seen in Escanciano (2006) and García-Portugués et al. (2021).

Usage

```
flm_stat(E, p, Adot_vec, constant = TRUE)

Adot(X)
```

Arguments

`E`	the matrix of multivariate residuals, with dimension `c(n, q)`. A vector if $q = 1$.
`p`	dimension of the covariates space. Must be a positive integer.
`Adot_vec`	output from `Adot`. A vector of length `n * (n - 1) / 2 + 1`. This corresponds to the most expensive computation in the test statistic.
`constant`	whether to include the constant of the PCvM test statistic, c, in its computation. Defaults to `TRUE`.
`X`	a matrix of size `c(n, p)` containing the coefficients (scores) of the functional data in a `p`-truncated *orthonormal* basis expansion, such as FPC. Must not contain repeated rows.

Details

`Adot` assumes that `X` does not have repeated rows or otherwise `NaN`s will be present in the result. If `X` has repeated rows, `Adot` will throw a warning.

The implementation of the PCvM test statistic for scalar response is addressed in García-Portugués et al. (2014), whereas García-Portugués et al. (2021) presents its multivariate extension and shows that $\mathbf{A}_\bullet$ induces a weighted quadratic norm (if there are no repetitions in the sample). The PCvM statistic is rooted in the proposal by Escanciano (2006).

Both `flm_stat` and `A_dot` are coded in C++.

Value

- `flm_stat`: the value of the test statistic, a scalar.
- `A_dot`: a vector of length `n * (n - 1) / 2 + 1`. The first entry contains the common diagonal element of $\mathbf{A}_\bullet$. The remaining entries are the upper triangular matrix (excluding the diagonal) of $\mathbf{A}_\bullet$, stacked by columns.

Author(s)

Eduardo García-Portugués.

References

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

Escanciano, J. C. (2006) A consistent diagnostic test for regression models using projections. *Econometric Theory*, 22(6):1030—1051. doi: 10.1017/S0266466606060506

García-Portugués, E., González-Manteiga, W. and Febrero-Bande, M. (2014). A goodness-of-fit test for the functional linear model with scalar response. *Journal of Computational and Graphical Statistics*, 23(3):761–778. doi: 10.1080/10618600.2013.812519

Examples

```
## flm_stat

# Generate data
n <- 200
q <- 2
p <- 3
E <- matrix(rnorm(n * q), nrow = n, ncol = q)
X_fdata <- r_ou(n = n, t = seq(0, 1, l = 101))

# Compute FPC
X_fpc <- fpc(X_fdata)

# Adot
Adot_vec <- Adot(X = X_fpc[["scores"]])

# Check equality
constant <- n^(-2) * 2 * pi^((p / 2) - 1) / gamma(p / 2)
constant * .Fortran("pcvm_statistic", n = as.integer(n),
                    Adot_vec = Adot_vec, residuals = E[, 2],
                    statistic = 0)$statistic
flm_stat(E = E[, 2, drop = FALSE], p = p, Adot_vec = Adot_vec,
         constant = FALSE)

## Adot

# Generate data
n <- 200
X_fdata <- r_ou(n = n, t = seq(0, 1, l = 101))

# Compute FPC
X_fpc <- fpc(X_fdata)

# Using inprod_fdata and Adot
Adot_vec <- Adot(X = X_fpc[["scores"]])

# Check with fda.usc::Adot with adequate inprod
head(drop(Adot_vec))
head(fda.usc::Adot(X_fdata))

# Obtention of the entire Adot matrix
Ad <- diag(rep(Adot_vec[1], n))
Ad[upper.tri(Ad, diag = FALSE)] <- Adot_vec[-1]
head(Ad <- t(Ad) + Ad - diag(diag(Ad)))

# Positive definite
eigen(Ad)$values

# # Warning if X contains repeated observations
# Adot(X = rbind(1:3, 1:3, 3:5))

# Comparison with .Fortran("adot", PACKAGE = "fda.usc")
```

```
n <- as.integer(n)
a <- as.double(rep(0, (n * (n - 1) / 2 + 1)))
inprod <- X_fpc[["scores"]] %*% t(X_fpc[["scores"]])
inprod <- inprod[upper.tri(inprod, diag = TRUE)]
X <- X_fpc[["scores"]]
microbenchmark::microbenchmark(
  .Fortran("adot", n = n, inprod = inprod, Adot_vec = a,
           PACKAGE = "fda.usc"),
  Adot(X = X),
  times = 50, control = list(warmup = 10))
```

`flm_term` *Functional linear model term with bivariate kernel*

Description

Computation of the functional linear term

$$\int_a^b \beta(s,t)X(s)\,\mathrm{d}s,$$

of a Functional Linear Model with Functional Response (FLMFR), where X is a random variable in the Hilbert space of square-integrable functions in $[a,b]$, $L^2([a,b])$, β is the bivariate kernel of the FLMFR, and ε is a random variable in $L^2([c,d])$.

Usage

```
flm_term(X_fdata, beta, t, int_rule = "trapezoid", equispaced = NULL,
  concurrent = FALSE)
```

Arguments

`X_fdata`	sample of functional data as an `fdata` object of length `n`.
`beta`	matrix containing the values $\beta(s,t)$, for each grid point s in $[a,b]$ and t in $[c,d]$. If `concurrent = TRUE`, a row/column vector must be introduced, valued in the same grid as `error_fdata`, with the same length as `length(X_fdata$argvals)`.
`t`	grid points where responses are valued.
`int_rule`	quadrature rule for approximating the definite unidimensional integral: trapezoidal rule (`int_rule = "trapezoid"`) and extended Simpson rule (`int_rule = "Simpson"`) are available. Defaults to `"trapezoid"`.
`equispaced`	flag to indicate if `X_fdata$data` is valued in an equispaced grid or not. Defaults to `FALSE`.
`concurrent`	flag to consider a concurrent FLMFR (degenerate case). Defaults to `FALSE`.

Value

Functional linear model term as the integral (in `s`) between `X_fdata` and `beta`, as an `fdata` object of length `n`.

Author(s)

Javier Álvarez-Liébana.

References

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

Examples

```
## Generate a sample of functional responses via FLMFR

# Bivariate kernel beta(s,t) as an egg carton shape
s <- seq(0, 1, l = 101)
t <- seq(0, 1, l = 201)
beta <- outer(s, t, FUN = function(s, t) sin(6 * pi * s) + cos(6 * pi * t))

# Functional data as zero-mean Gaussian process with exponential variogram
X_fdata <- fda.usc::rproc2fdata(n = 50, t = s, sigma = "vexponential",
                                list = list(scale = 2.5))

# Functional error as an OU process with variance 0.075
sigma <- sqrt(0.075) * 2
error_fdata <- r_ou(n = 50, t = t, sigma = sigma)
Y_fdata <- flm_term(X_fdata = X_fdata, beta = beta, t = t) + error_fdata
plot(Y_fdata)

## Generate a sample of functional responses via concurrent model

# Function beta(t)
s <- seq(1, 3, l = 201)
t <- seq(2, 5, l = 201)
beta <- sin(pi * t) + cos(pi * t)

# Functional data as zero-mean Gaussian process with exponential variogram
X_fdata <- fda.usc::rproc2fdata(n = 50, t = s, sigma = "vexponential",
                                list = list(scale = 2.5))

# Functional error as an OU process with variance 0.075
sigma <- sqrt(0.075) * 2
error_fdata <- r_ou(n = 50, t = t, sigma = sigma)
Y_fdata <- flm_term(X_fdata = X_fdata, beta = beta, t = t,
                    concurrent = TRUE) + error_fdata
plot(Y_fdata)
```

`flm_test` *Goodness-of-fit test for functional linear models*

Description

Goodness-of-fit test of a functional linear model with functional response $Y \in L^2([c,d])$ and functional predictor $X \in L^2([a,b])$, where $L^2([a,b])$ is the Hilbert space of square-integrable functions in $[a,b]$.

The goodness-of-fit test checks the *linearity* of the regression model $m : L^2([a,b]) \to L^2([c,d])$ that relates Y and X by

$$Y(t) = m(X) + \varepsilon(t),$$

where ε is a random variable in $L^2([c,d])$ and $t \in [c,d]$. The check is formalized as the test of the composite hypothesis

$$H_0 : m \in \{m_\beta : \beta \in L^2([a,b]) \otimes L^2([c,d])\},$$

where

$$m_\beta(X(s))(t) = \int_a^b \beta(s,t)X(s)\,\mathrm{d}s$$

is the linear, Hilbert–Schmidt, integral operator parametrized by the bivariate kernel β. Its estimation is done by the truncated expansion of β in the tensor product of the data-driven bases of *Functional Principal Components* (FPC) of X and Y. The FPC basis for X is truncated in p components, while the FPC basis for Y is truncated in q components.

The particular cases in which either X or Y are *constant* functions give either a scalar predictor or response. The simple linear model arises if both X and Y are scalar, for which β is a constant.

Usage

```
flm_test(X, Y, beta0 = NULL, B = 500, est_method = "fpcr", p = NULL,
  q = NULL, thre_p = 0.99, thre_q = 0.99, lambda = NULL,
  boot_scores = TRUE, verbose = TRUE, plot_dens = TRUE,
  plot_proc = TRUE, plot_max_procs = 100, plot_max_p = 2,
  plot_max_q = 2, save_fit_flm = TRUE, save_boot_stats = TRUE,
  int_rule = "trapezoid", refit_lambda = FALSE, ...)
```

Arguments

`X, Y`	samples of functional/scalar predictors and functional/scalar response. Either `fdata` objects (for functional variables) or vectors of length `n` (for scalar variables).
`beta0`	if provided (defaults to `NULL`), the *simple* null hypothesis $H_0 : m = m_{\beta_0}$ is tested. `beta0` must be a matrix of size `c(length(X$argvals), length(Y$argvals))`. If `X` or `Y` are scalar, `beta0` can be also an `fdata` object, with the same `argvals` as `X` or `Y`. Can also be a constant (understood as a shorthand for a matrix with *all* its entries equal to the constant).
`B`	number of bootstrap replicates. Defaults to `500`.
`est_method`	either `"fpcr"` (Functional Principal Components Regression; FPCR), `"fpcr_l2"` (FPCR with ridge penalty), `"fpcr_l1"` (FPCR with lasso penalty) or `"fpcr_l1s"` (FPCR with lasso-selected FPC). If `X` is scalar, `flm_est` only considers `"fpcr"` as estimation method. See details below. Defaults to `"fpcr_l1s"`.

`p, q` either index vectors indicating the specific FPC to be considered for the truncated bases expansions of `X` and `Y`, respectively. If a single number for `p` is provided, then `p <- 1:max(p)` internally (analogously for `q`) and the first `max(p)` FPC are considered. If `NULL` (default), then a data-driven selection of `p` and `q` is done. See details below.

`thre_p, thre_q` thresholds for the *proportion* of variance that is explained, *at least*, by the first p and q FPC of `X` and `Y`, respectively. These thresholds are employed for an (initial) automatic selection of p and q. Default to `0.99`. `thre_p` (`thre_q`) is ignored if `p` (`q`) is provided.

`lambda` regularization parameter λ for the estimation methods `"fpcr_l2"`, `"fpcr_l1"`, and `"fpcr_l1s"`. If `NULL` (default), it is chosen with `cv_glmnet`.

`boot_scores` flag to indicate if the bootstrap shall be applied to the scores of the residuals, rather than to the functional residuals. This improves the computational expediency notably. Defaults to `TRUE`.

`verbose` flag to show information about the testing progress. Defaults to `TRUE`.

`plot_dens` flag to indicate if a kernel density estimation of the bootstrap statistics shall be plotted. Defaults to `TRUE`.

`plot_proc` whether to display a graphical tool to identify the degree of departure from the null hypothesis. If `TRUE` (default), the residual marked empirical process, projected in several FPC directions of `X` and `Y`, is shown, together with bootstrap analogues. The FPC directions are ones selected at the estimation stage.

`plot_max_procs` maximum number of bootstrapped processes to plot in the graphical tool. Set as the minimum of `plot_max_procs` and `B`. Defaults to `100`.

`plot_max_p, plot_max_q` maximum number of FPC directions to be considered in the graphical tool. They limit the resulting plot to be at most of size `c(plot_max_p, plot_max_q)`. Default to `2`.

`save_fit_flm, save_boot_stats` flag to return `fit_flm` and `boot_*`. If `FALSE`, these memory-expensive objects are set to `NA`. Default to `TRUE`.

`int_rule` quadrature rule for approximating the definite unidimensional integral: trapezoidal rule (`int_rule = "trapezoid"`) and extended Simpson rule (`int_rule = "Simpson"`) are available. Defaults to `"trapezoid"`.

`refit_lambda` flag to reselect $lambda$ in each bootstrap replicate, incorporating its variability in the bootstrap calibration. Much more time consumig. Defaults to `FALSE`.

`...` further parameters to be passed to `cv_glmnet` (and then to `cv.glmnet`) such as `cv_1se`, `cv_nlambda` or `cv_parallel`, among others.

Details

The function implements the bootstrap-based goodness-of-fit test for the functional linear model with functional/scalar response and functional/scalar predictor, as described in Algorithm 1 in García-Portugués et al. (2021). The specifics are detailed there.

By default `cv_1se = TRUE` for `cv_glmnet` is considered, unless it is changed via `...`. This is the recommended choice for conducting the goodness-of-fit test based on regularized estimators, as

the oversmoothed estimate of the regression model under the null hypothesis notably facilitates the calibration of the test (see García-Portugués et al., 2021).

The graphical tool obtained with `plot_proc = TRUE` is based on an extension of the tool described in García-Portugués et al. (2014).

Repeated observations on `X` are internally removed, as otherwise they would cause `NaNs` in `Adot`. Missing values on `X` and `Y` are also automatically removed.

Value

An object of the `htest` class with the following elements:

`statistic`	test statistic.
`p.value`	p-value of the test.
`boot_statistics`	the bootstrapped test statistics, a vector of length `B`.
`method`	information on the type of test performed.
`parameter`	a vector with the dimensions p and q considered in the test statistic. These are the lengths of the outputs `p` and `q`.
`p`	the index of the FPC considered for `X`.
`q`	the index of the FPC considered for `Y`.
`fit_flm`	the output resulted from calling `flm_est`.
`boot_lambda`	bootstrapped $lambda$.
`boot_p`	a list with the bootstrapped indexes of the FPC considered for `X`.
`data.name`	name of the value of `data`.

Author(s)

Eduardo García-Portugués.

References

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

García-Portugués, E., González-Manteiga, W. and Febrero-Bande, M. (2014). A goodness-of-fit test for the functional linear model with scalar response. *Journal of Computational and Graphical Statistics*, 23(3):761–778. doi: 10.1080/10618600.2013.812519

Examples

```
## Quick example for functional response and predictor

# Generate data under H0
n <- 100
set.seed(987654321)
X_fdata <- r_ou(n = n, t = seq(0, 1, l = 101), sigma = 2)
```

```
epsilon <- r_ou(n = n, t = seq(0, 1, l = 101), sigma = 0.5)
Y_fdata <- epsilon

# Test the FLMFR
flm_test(X = X_fdata, Y = Y_fdata)

# Simple hypothesis
flm_test(X = X_fdata, Y = Y_fdata, beta0 = 0)

# Generate data under H1
n <- 100
set.seed(987654321)
sample_frm_fr <- r_frm_fr(n = n, scenario = 3, s = seq(0, 1, l = 101),
                          t = seq(0, 1, l = 101), nonlinear = "quadratic")
X_fdata <- sample_frm_fr[["X_fdata"]]
Y_fdata <- sample_frm_fr[["Y_fdata"]]

# Test the FLMFR
flm_test(X = X_fdata, Y = Y_fdata)

## Functional response and predictor

# Generate data under H0
n <- 50
B <- 100
set.seed(987654321)
t <- seq(0, 1, l = 201)
X_fdata <- r_ou(n = n, t = t, sigma = 2)
epsilon <- r_ou(n = n, t = t, sigma = 0.5)
Y_fdata <- epsilon

# With boot_scores = TRUE
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l2", B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s", B = B)

# With boot_scores = FALSE
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l2",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s",
         boot_scores = FALSE, B = B)

# Simple hypothesis
flm_test(X = X_fdata, Y = Y_fdata, beta0 = 2, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y_fdata, beta0 = 0, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y_fdata, beta0 = 0, est_method = "fpcr_l1s", B = B)

# Generate data under H1
n <- 50
```

```
B <- 100
set.seed(987654321)
sample_frm_fr <- r_frm_fr(n = n, scenario = 3, s = t, t = t,
                          nonlinear = "quadratic")
X_fdata <- sample_frm_fr$X_fdata
Y_fdata <- sample_frm_fr$Y_fdata

# With boot_scores = TRUE
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l2", B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s", B = B)

# With boot_scores = FALSE
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l2",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y_fdata, est_method = "fpcr_l1s",
         boot_scores = FALSE, B = B)

## Scalar response and functional predictor

# Generate data under H0
n <- 50
B <- 100
set.seed(987654321)
t <- seq(0, 1, l = 201)
X_fdata <- r_ou(n = n, t = t, sigma = 2)
beta <- r_ou(n = 1, t = t, sigma = 0.5, x0 = 2)
epsilon <- rnorm(n = n)
Y <- drop(inprod_fdata(X_fdata1 = X_fdata, X_fdata2 = beta) + epsilon)

# With boot_scores = TRUE
flm_test(X = X_fdata, Y = Y, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l2", B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l1s", B = B)

# With boot_scores = FALSE
flm_test(X = X_fdata, Y = Y, est_method = "fpcr",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l2",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l1",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l1s",
         boot_scores = FALSE, B = B)

# Simple hypothesis
flm_test(X = X_fdata, Y = Y, beta0 = beta, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y, beta0 = 0, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y, beta0 = 0, est_method = "fpcr_l1s", B = B)
```

```
# Generate data under H1
n <- 50
B <- 100
set.seed(987654321)
X_fdata <- r_ou(n = n, t = t, sigma = 2)
beta <- r_ou(n = 1, t = t, sigma = 0.5)
epsilon <- rnorm(n = n)
Y <- drop(exp(inprod_fdata(X_fdata1 = X_fdata^2, X_fdata2 = beta)) + epsilon)

# With boot_scores = TRUE
flm_test(X = X_fdata, Y = Y, est_method = "fpcr", B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l2", B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l1s", B = B)

# With boot_scores = FALSE
flm_test(X = X_fdata, Y = Y, est_method = "fpcr",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l2",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l1",
         boot_scores = FALSE, B = B)
flm_test(X = X_fdata, Y = Y, est_method = "fpcr_l1s",
         boot_scores = FALSE, B = B)

## Functional response and scalar predictor

# Generate data under H0
n <- 50
B <- 100
set.seed(987654321)
X <- rnorm(n)
t <- seq(0, 1, l = 201)
beta <- r_ou(n = 1, t = t, sigma = 0.5, x0 = 3)
beta$data <- matrix(beta$data, nrow = n, ncol = ncol(beta$data),
                    byrow = TRUE)
epsilon <- r_ou(n = n, t = t, sigma = 0.5)
Y_fdata <- X * beta + epsilon

# With boot_scores = TRUE
flm_test(X = X, Y = Y_fdata, est_method = "fpcr", B = B)

# With boot_scores = FALSE
flm_test(X = X, Y = Y_fdata, est_method = "fpcr", boot_scores = FALSE, B = B)

# Simple hypothesis
flm_test(X = X, Y = Y_fdata, beta0 = beta[1], est_method = "fpcr", B = B)
flm_test(X = X, Y = Y_fdata, beta0 = 0, est_method = "fpcr", B = B)

# Generate data under H1
n <- 50
B <- 100
set.seed(987654321)
```

```
X <- rexp(n)
beta <- r_ou(n = 1, t = t, sigma = 0.5, x0 = 3)
beta$data <- matrix(beta$data, nrow = n, ncol = ncol(beta$data),
                    byrow = TRUE)
epsilon <- r_ou(n = n, t = t, sigma = 0.5)
Y_fdata <- log(X * beta) + epsilon

# With boot_scores = TRUE
flm_test(X = X, Y = Y_fdata, est_method = "fpcr", B = B)

# With boot_scores = FALSE
flm_test(X = X, Y = Y_fdata, est_method = "fpcr", boot_scores = FALSE, B = B)
```

---

fpc *Computation of functional principal components*

Description

Computation of Functional Principal Components (FPC) for equispaced and non equispaced functional data.

Usage

```
fpc(X_fdata, n_fpc = 3, centered = FALSE, int_rule = "trapezoid",
  equispaced = FALSE, verbose = FALSE)
```

Arguments

X_fdata	sample of functional data as an fdata object of length n.
n_fpc	number of FPC to be computed. If n_fpc > n, n_fpc is set to n. Defaults to 3.
centered	flag to indicate if X_fdata is centered or not. Defaults to FALSE.
int_rule	quadrature rule for approximating the definite unidimensional integral: trapezoidal rule (int_rule = "trapezoid") and extended Simpson rule (int_rule = "Simpson") are available. Defaults to "trapezoid".
equispaced	flag to indicate if X_fdata$data is valued in an equispaced grid or not. Defaults to FALSE.
verbose	whether to show or not information about the fpc procedure. Defaults to FALSE.

Details

The FPC are obtained by performing the single value decomposition

$$\mathbf{X}\mathbf{W}^{1/2} = \mathbf{U}\mathbf{D}(\mathbf{V}'\mathbf{W}^{1/2})$$

where $\mathbf{X}$ is the matrix of discretized functional data, $\mathbf{W}$ is a diagonal matrix of weights (computed by w_integral1D according to int_rule), $\mathbf{D}$ is the diagonal matrix with singular values (standard deviations of FPC), $\mathbf{U}$ is a matrix whose columns contain the left singular vectors, and $\mathbf{V}$ is a matrix whose columns contain the right singular vectors (FPC).

Value

An "fpc" object containing the following elements:

d	standard deviations of the FPC (i.e., square roots of eigenvalues of the empirical autocovariance estimator).
rotation	orthonormal eigenfunctions (loadings or functional principal components), as an fdata class object.
scores	rotated samples: inner products. between X_fdata and eigenfunctions in rotation.
l	vector of index of FPC.
equispaced	equispaced flag.

Author(s)

Javier Álvarez-Liébana and Gonzalo Álvarez-Pérez.

References

Jolliffe, I. T. (2002). Principal Component Analysis. Springer-Verlag, New York.

Examples

```
## Computing FPC for equispaced data

# Sample data
X_fdata1 <- r_ou(n = 200, t = seq(2, 4, l = 201))

# FPC with trapezoid rule
X_fpc1 <- fpc(X_fdata = X_fdata1, n_fpc = 50, equispaced = TRUE,
              int_rule = "trapezoid")

# FPC with Simpsons's rule
X_fpc2 <- fpc(X_fdata = X_fdata1, n_fpc = 50, equispaced = TRUE,
               int_rule = "Simpson")

# Check if FPC are orthonormal
norms1 <- rep(0, length(X_fpc1$l))
for (i in X_fpc1$l) {

  norms1[i] <- integral1D(fx = X_fpc1$rotation$data[i, ]^2,
                          t = X_fdata1$argvals)

}
norms2 <- rep(0, length(X_fpc2$l))
for (i in X_fpc2$l) {

  norms2[i] <- integral1D(fx = X_fpc2$rotation$data[i, ]^2,
                          t = X_fdata1$argvals)

}
```

```
## Computing FPC for non equispaced data

# Sample data
X_fdata2 <- r_ou(n = 200, t = c(seq(0, 0.5, l = 201), seq(0.51, 1, l = 301)))

# FPC with trapezoid rule
X_fpc3 <- fpc(X_fdata = X_fdata2, n_fpc = 5, int_rule = "trapezoid",
              equispaced = FALSE)

# Check if FPC are orthonormal
norms3 <- rep(0, length(X_fpc3$l))
for (i in X_fpc3$l) {

  norms3[i] <- integral1D(fx = X_fpc3$rotation$data[i, ]^2,
                          t = X_fdata2$argvals)

}

## Efficiency comparisons

# fpc() vs. fda.usc::fdata2pc()
data(phoneme, package = "fda.usc")
mlearn <- phoneme$learn[1:10, ]
res1 <- fda.usc::fdata2pc(mlearn, ncomp = 3)
res2 <- fpc(X_fdata = mlearn, n_fpc = 3)
plot(res1$x[, 1:3], col = 1)
points(res2$scores, col = 2)

microbenchmark::microbenchmark(fda.usc::fdata2pc(mlearn, ncomp = 3),
                               fpc(X_fdata = mlearn, n_fpc = 3), times = 1e3,
                               control = list(warmup = 20))
```

`fpc_utils` *Utilities for functional principal components*

Description

Computation of coefficients and reconstructions based on Functional Principal Components (FPC). The function `fpc_coefs` allows to project a functional data sample into a basis of FPC; the reconstruction of the sample from its projections and the FPC is done with `fpc_to_fdata`. The functions `beta_fpc_coefs` and `fpc_to_beta` do analogous operations but for the bivariate kernel β and the tensor product of two FPC bases.

Usage

```
fpc_coefs(X_fdata, X_fpc, ind_X_fpc = 1:3, int_rule = "trapezoid")

beta_fpc_coefs(beta, X_fpc, Y_fpc, ind_X_fpc = 1:3, ind_Y_fpc = 1:3,
```

```
  int_rule = "trapezoid")

fpc_to_fdata(coefs, X_fpc, ind_coefs = 1:ncol(coefs))

fpc_to_beta(beta_coefs, X_fpc, Y_fpc, ind_coefs_X = 1:nrow(beta_coefs),
  ind_coefs_Y = 1:ncol(beta_coefs))
```

Arguments

`X_fdata`	sample of functional data as an `fdata` object of length `n`.
`X_fpc, Y_fpc`	`"fpc"` objects as resulted from calling `fpc`.
`ind_X_fpc, ind_Y_fpc`	vectors giving the FPC indexes for whom the coefficients are computed. Their lengths must be smaller than the number of FPC in `X_fpc` and `Y_fpc`, respectively. Default to `1:3`.
`int_rule`	quadrature rule for approximating the definite unidimensional integral: trapezoidal rule (`int_rule = "trapezoid"`) and extended Simpson rule (`int_rule = "Simpson"`) are available. Defaults to `"trapezoid"`.
`beta`	a matrix containing the bivariate kernel β evaluated on a grid. Must be of size `c(length(X_fpc$rotation$argvals),length(Y_fpc$rotation$argvals))`.
`coefs`	a vector of coefficients to combine linearly the FPC. Its length must be smaller than the number of FPC in `X_fpc`.
`ind_coefs, ind_coefs_X, ind_coefs_Y`	indexes of FPC to associate to the provided coefficients. By default, from the first FPC to the sizes of `coefs` or `beta_coefs`.
`beta_coefs`	a matrix of coefficients to combine linearly the tensor products of FPC. Its size must be smaller than the number of FPC in `X_fpc` and `Y_fpc`.

Value

`fpc_coefs`	a vector of the same length as `coefs` containing the coefficients of `X_fdata` in the FPC of `X_fpc`.
`beta_fpc_coefs`	a matrix of the same size as `beta_coefs` containing the coefficients of β in the tensor product of the FPC in `X_fpc` and `Y_fpc`.
`fpc_to_fdata`	an `fdata` object of the same type as `X_fpc$rotation`.
`fpc_to_beta`	a matrix with the reconstructed kernel and size `c(length(X_fpc$rotation$argvals), length(Y_fpc$rotation$argvals))`.

Author(s)

Eduardo García-Portugués.

References

Jolliffe, I. T. (2002). Principal Component Analysis. Springer-Verlag, New York.

Examples

```
## Compute FPC coefficients and reconstruct data

# Sample data
X_fdata <- r_ou(n = 200, t = seq(2, 4, l = 201))

# Compute FPC
X_fpc <- fpc(X_fdata = X_fdata, n_fpc = 50)

# FPC coefficients are the same if the data is centered
fpc_coefs(X_fdata = fdata_cen(X_fdata), X_fpc = X_fpc)[1:4, ]
X_fpc$scores[1:4, 1:3]

# Reconstruct the first two curves for an increasing number of FPC
plot(X_fdata[1:2, ], col = 1)
n_fpc <- c(2, 5, 10, 25, 50)
for (j in 1:5) {
  lines(fpc_to_fdata(X_fpc = X_fpc,
                     coefs = X_fpc$scores[, 1:n_fpc[j]])[1:2, ], col = j + 1)
}

## Project and reconstruct beta

# Surface
beta_fun <- function(s, t) sin(6 * pi * s) + cos(6 * pi * t)
s <- seq(0, 1, l = 101)
t <- seq(0, 1, l = 201)
beta_surf <- outer(s, t, FUN = beta_fun)

# Functional data as zero-mean Gaussian process with exponential variogram
X_fdata <- fda.usc::rproc2fdata(n = 100, t = s, sigma = "vexponential",
                                list = list(scale = 2.5))
Y_fdata <- flm_term(X_fdata = X_fdata, beta = beta_surf, t = t) +
  r_ou(n = 100, t = t, sigma = sqrt(0.075) * 2)

# FPC
X_fpc <- fpc(X_fdata = X_fdata, n_fpc = 50)
Y_fpc <- fpc(X_fdata = Y_fdata, n_fpc = 50)

# Coefficients
beta_coefs <- beta_fpc_coefs(beta = beta_surf, X_fpc = X_fpc, Y_fpc = Y_fpc,
                             ind_X_fpc = 1:50, ind_Y_fpc = 1:50)

# Reconstruction
beta_surf1 <- fpc_to_beta(beta_coefs = beta_coefs[1:2, 1:5],
                          X_fpc = X_fpc, Y_fpc = Y_fpc)
beta_surf2 <- fpc_to_beta(beta_coefs = beta_coefs[1:15, 1:10],
                          X_fpc = X_fpc, Y_fpc = Y_fpc)
beta_surf3 <- fpc_to_beta(beta_coefs = beta_coefs[1:50, 1:50],
                          X_fpc = X_fpc, Y_fpc = Y_fpc)

# Show reconstructions
```

```
old_par <- par(mfrow = c(2, 2))
col <- viridisLite::viridis(20)
image(s, t, beta_surf, col = col, zlim = c(-2.5, 2.5), main = "Original")
image(s, t, beta_surf1, col = col, zlim = c(-2.5, 2.5), main = "2 x 5")
image(s, t, beta_surf2, col = col, zlim = c(-2.5, 2.5), main = "15 x 10")
image(s, t, beta_surf3, col = col, zlim = c(-2.5, 2.5), main = "50 x 50")
par(old_par)
```

`ontario` *Ontario temperature and electricity consumption during 2010–2014*

Description

Real dataset employed Benatia et al. (2017). Contains the hourly electricity consumption and air temperature curves in the province of Ontario (Canada). It features a set of daily curves during the summer months of 2010–2014.

Usage

```
ontario
```

Format

A list with the following entries:

temp an `fdata` with 368 smoothed daily temperature (in Celsius degrees) curves of the Ontario province, discretized on 73 equispaced grid points on $[-24, 48]$ (see examples).

elec an `fdata` with the daily electricity consumption (in gigawatts) curves of the Ontario province. Discretized on 25 equispaced grid points on $[0, 24]$.

df a dataframe with time metadata for each curve:

- `date`: the date of the observation, a `POSIXct` object.
- `weekday`: the weekday of the observation.

Details

The summer months correspond to June 1st to September 15th. Weekend days and holidays are disregarded.

The smoothed temperature curves are constructed by a weighted average of the temperatures of 41 Ontarian cities that is afterwards smoothed with a local polynomial regression. The curves correspond to a 3-days window of the temperature (see examples). The temperature is standardized such that its original minimum, 6 ºC, is subtracted.

The electricity consumption curves are discretized on the interval $[0, 24]$. That means that the last observation of the i-th curve is the same as the first observation of the $(i + 1)$-th curve *if* the curves correspond to consecutive days.

See more details about the construction of the dataset in Benatia et al. (2017).

Author(s)

Data gathered and processed by David Benatia, Marine Carrasco, and Jean-Pierre Florens. Javier Álvarez-Liébana and Eduardo García-Portugués imported the dataset and added temporal metadata.

Source

The dataset comes from the companion data to Benatia et al. (2017), which was retrieved from the first author's website. The source of the electricity consumption data is the System operator's website. The source of the preprocessed temperature values is the Environment Canada's website.

References

Benatia, D., Carrasco, M. and Florens, J. P. (2017) Functional linear regression with functional response. *Journal of Econometrics*, 201(2):269–291. doi: 10.1016/j.jeconom.2017.08.008

Examples

```
## Show data

# Load data
data("ontario")

# Plot
old_par <- par(mfrow = c(1, 2))
plot(ontario$temp)
plot(ontario$elec)
par(old_par)

# Observe the 3-day windows for each observation
plot(ontario$temp$argvals, ontario$temp$data[2, ], type = "o",
     xlim = c(-48, 72), ylim = c(7, 13), xlab = "Hours",
     ylab = "Electricity consumption", pch = 16)
points(ontario$temp$argvals - 24, ontario$temp$data[1, ], col = 3, pch = 2)
points(ontario$temp$argvals + 24, ontario$temp$data[3, ], col = 2, cex = 1.5)
abline(v = 24 * -2:3, lty = 2)
legend("top", legend = c("Curve 1", "Curve 2", "Curve 3"), col = c(3, 1, 2),
       pt.cex = c(1, 1, 1.5), pch = c(2, 16, 1))

# If the days are not consecutive, then the electricity consumptions at the
# end of one day and the beginning of the next do not match
head(abs(ontario$elec$data[-368, 25] - ontario$elec$data[-1, 1]))
head(diff(ontario$df$date))

## Test the linear model with functional response and predictor

(comp_flmfr <- flm_test(X = ontario$temp, Y = ontario$elec,
                        est_method = "fpcr_l1s"))
(simp_flmfr <- flm_test(X = ontario$temp, Y = ontario$elec,
                        beta0 = 0, est_method = "fpcr_l1s"))

# Visualize estimation
```

```
filled.contour(x = ontario$temp$argvals, y = ontario$elec$argvals,
               z = comp_flmfr$fit_flm$Beta_hat,
               color.palette = viridisLite::viridis, nlevels = 20)
```

---

r_ou *Simulation of an Ornstein–Uhlenbeck process*

Description

Simulation of trajectories of the Ornstein–Uhlenbeck process $\{X_t\}$. The process is the solution to the stochastic differential equation

$$\mathrm{d}X_t = \alpha(X_t - \mu)\mathrm{d}t + \sigma \mathrm{d}W_t,$$

whose stationary distribution is $N(\mu, \sigma^2/(2\alpha))$, for $\alpha, \sigma > 0$ and $\mu \in R$.

Given an initial point x_0 and the evaluation times $t_1, \ldots, t_m$, a sample trajectory $X_{t_1}, \ldots, X_{t_m}$ can be obtained by sampling the joint Gaussian distribution of $(X_{t_1}, \ldots, X_{t_m})$.

Usage

```
r_ou(n, t = seq(0, 1, len = 201), mu = 0, alpha = 1, sigma = 1,
  x0 = rnorm(n, mean = mu, sd = sigma/sqrt(2 * alpha)))
```

Arguments

n	number of trajectories to sample.
t	evaluation times for the trajectories, a vector.
mu	mean of the process, a scalar.
alpha	strength of the drift, a positive scalar.
sigma	diffusion coefficient, a positive scalar.
x0	a vector of length n giving the initial values of the Ornstein–Uhlenbeck trajectories. By default, n points are sampled from the stationary distribution. If a single scalar is passed, then the same x0 is employed for all the trajectories.

Value

Random trajectories, an fdata object of length n and t as argvals.

Author(s)

Eduardo García-Portugués.

Examples

```
# Same initial point
plot(r_ou(n = 20, x0 = 5), col = viridisLite::viridis(20))

# Different initial points
plot(r_ou(n = 100, alpha = 2, sigma = 4, x0 = 1:100),
     col = viridisLite::viridis(100))
```

`sim-frmfr` *Sampling functional regression models with functional responses*

Description

Simulation of a Functional Regression Model with Functional Response (FRMFR) comprised of an additive mix of a linear and nonlinear terms:

$$Y(t) = \int_a^b X(s)\beta(s,t)ds + \Delta(X)(t) + \varepsilon(t),$$

where X is a random variable in the Hilbert space of square-integrable functions in $[a, b]$, $L^2([a, b])$, β is the bivariate kernel of the FRMFR, ε is a random variable in $L^2([c, d])$, and $\Delta(X)$ is a nonlinear term.

In particular, the scenarios considered in García-Portugués et al. (2021) can be easily simulated.

Usage

```
r_frm_fr(n, scenario = 3, X_fdata = NULL, error_fdata = NULL,
  beta = NULL, s = seq(0, 1, l = 101), t = seq(0, 1, l = 101),
  std_error = 0.15, nonlinear = NULL, concurrent = FALSE,
  int_rule = "trapezoid", n_fpc = 50, verbose = FALSE, ...)

nl_dev(X_fdata, t = seq(0, 1, l = 101), nonlinear = NULL,
  int_rule = "trapezoid", equispaced = equispaced, verbose = FALSE)
```

Arguments

`n`	sample size, only required when `scenario` is given.
`scenario`	an index from 1 to 3 (default) denoting one of the scenarios (S1, S2 or S3) simulated in García-Portugués et al. (2021) (see details below). If `scenario = NULL`, `X_fdata`, `error_fdata`, and `beta` have to be provided. Otherwise, `X_fdata`, `error_fdata`, and `beta` will be ignored.
`X_fdata`	sample of functional covariates $X(s)$ as `fdata` objects of length `n`, with s in $[a, b]$. Defaults to `NULL`.
`error_fdata`	sample of functional errors $\varepsilon(t)$ as `fdata` objects of length `n`, with t in $[c, d]$. If `concurrent = TRUE`, `X_fdata` and `error_fdata` must be valued in the same grid. Defaults to `NULL`.

beta	matrix containing the values $\beta(s,t)$, for each grid point s in $[a,b]$ and t in $[c,d]$. If concurrent = TRUE (see details below), a row/column vector must be introduced, valued in the same grid as error_fdata. If beta = NULL (default), scenario != NULL is required.
s, t	grid points. If X_fdata, error_fdata and beta are provided, s and t are ignored. Default to s = seq(0, 1, l = 101) and t = seq(0, 1, l = 101), respectively.
std_error	standard deviation of the random variables involved in the generation of the functional error error_fdata. Defaults to 0.15.
nonlinear	nonlinear term. Either a character string ("exp", "quadratic" or "sin") or an fdata object of length n, valued in the same grid as error_fdata. If nonlinear = NULL (default), the nonlinear part is set to zero.
concurrent	flag to consider a concurrent FLRFR (degenerate case). Defaults to FALSE.
int_rule	quadrature rule for approximating the definite unidimensional integral: trapezoidal rule (int_rule = "trapezoid") and extended Simpson rule (int_rule = "Simpson") are available. Defaults to "trapezoid".
n_fpc	number of components to be considered for the generation of functional variables. Defaults to 50.
verbose	flag to display information about the sampling procedure. Defaults to FALSE.
...	further parameters passed to r_cm2013_flmfr, r_gof2021_flmfr and r_ik2018_flmfr, depending on the chosen scenario.
equispaced	flag to indicate if X_fdata$data is valued in an equispaced grid or not. Defaults to FALSE.

Details

- r_frm_fr samples the above regression model, where the nonlinear term $\Delta(X)$ is computed by nl_dev. Functional covariates, errors, and β are generated automatically from the scenarios in García-Portugués et al. (2021) when scenario != NULL (see the documentation of r_gof2021_flmfr). If scenario = NULL, covariates, errors and β must be provided.

  When concurrent = TRUE, the concurrent FRMFR

  $$Y(t) = X(t)\beta(t) + \Delta(X)(t) + \varepsilon(t)$$

  is considered.

- nl_dev computes a nonlinear deviation $\Delta(X)$: $\exp(\sqrt{X(a+(t-c)((b-a)/(d-c)))})$ (for "exp"), $(X^2(a+(t-c)((b-a)/(d-c)))-1)$ ("quadratic") or $(\sin(2\pi t)-\cos(2\pi t))\|X\|^2$ ("sin"). Also, $\Delta(X)$ can be manually set as an fdata object of length n and valued in the same grid as error_fdata.

Value

A list with the following elements:

X_fdata	functional covariates, an fdata object of length n.
Y_fdata	functional responses, an fdata object of length n.

error_fdata	functional errors, an fdata object of length n.
beta	either the matrix with $\beta(s,t)$ evaluated at the argvals of X_fdata and Y_fdata (if concurrent = FALSE) or a vector with $\beta(t)$ evaluated at the argvals of X_fdata (if concurrent = TRUE).
nl_dev	nonlinear term, an fdata object of length n.

Author(s)

Javier Álvarez-Liébana.

References

García-Portugués, E., Álvarez-Liébana, J., Álvarez-Pérez, G. and Gonzalez-Manteiga, W. (2021). A goodness-of-fit test for the functional linear model with functional response. *Scandinavian Journal of Statistics*, 48(2):502–528. doi: 10.1111/sjos.12486

Examples

```
## Generate samples for the three scenarios

# Equispaced grids and Simpson's rule

s <- seq(0, 1, l = 101)
samp <- list()
old_par <- par(mfrow = c(3, 5))
for (i in 1:3) {
  samp[[i]] <- r_frm_fr(n = 100, scenario = i, s = s, t = s,
                        int_rule = "Simpson")
  plot(samp[[i]]$X_fdata)
  plot(samp[[i]]$error_fdata)
  plot(samp[[i]]$Y_fdata)
  plot(samp[[i]]$nl_dev)
  image(x = s, y = s, z = samp[[i]]$beta, col = viridisLite::viridis(20))
}
par(old_par)

## Linear term as a concurrent model

# The grids must be have the same number of grid points for a given
# nonlinear term and a given beta function

s <- seq(1, 2, l = 101)
t <- seq(0, 1, l = 101)
samp_c_1 <- r_frm_fr(n = 100, scenario = 3, beta = sin(t) - exp(t),
                     s = s, t = t, nonlinear = fda.usc::fdata(mdata =
                       t(matrix(rep(sin(t), 100), nrow = length(t))),
                       argvals = t),
                     concurrent = TRUE)
old_par <- par(mfrow = c(3, 2))
plot(samp_c_1$X_fdata)
plot(samp_c_1$error_fdata)
```

```
plot(samp_c_1$Y_fdata)
plot(samp_c_1$nl_dev)
plot(samp_c_1$beta)
par(old_par)

## Sample for given X_fdata, error_fdata, and beta

# Non equispaced grids with sinusoidal nonlinear term and intensity 0.5
s <- c(seq(0, 0.5, l = 50), seq(0.51, 1, l = 101))
t <- seq(2, 4, len = 151)
X_fdata <- r_ou(n = 100, t = s, alpha = 2, sigma = 4, x0 = 1:100)
error_fdata <- r_ou(n = 100, t = t, alpha = 1, sigma = 1, x0 = 1:100)
beta <- r_gof2021_flmfr(n = 100, s = s, t = t)$beta
samp_Xeps <- r_frm_fr(scenario = NULL, X_fdata = X_fdata,
                      error_fdata = error_fdata, beta = beta,
                      nonlinear = "exp", int_rule = "trapezoid")
old_par <- par(mfrow = c(3, 2))
plot(samp_Xeps$X_fdata)
plot(samp_Xeps$error_fdata)
plot(samp_Xeps$Y_fdata)
plot(samp_Xeps$nl_dev)
image(x = s, y = t, z = beta, col = viridisLite::viridis(20))
par(old_par)
```

Index