Package: funcharts (via r-universe)

November 1, 2026

Type Package

Title Functional Control Charts

Version 1.8.1

Description Provides functional control charts for statistical process monitoring of functional data, using the methods of Capezza et al. (2020) <doi:10.1002/asmb.2507>, Centofanti et al. (2021) <doi:10.1080/00401706.2020.1753581>, Capezza et al. (2024) <doi:10.1080/00224065.2024.2383674>, Capezza et al. (2024) <doi:10.1080/00401706.2024.2327346>, Centofanti et al. (2025) <doi:10.1080/00224065.2024.2430978>, Capezza et al. (2025) <doi:10.48550/arXiv.2410.20138>. The package is thoroughly illustrated in the paper of Capezza et al (2023) <doi:10.1080/00224065.2023.2219012>.

Depends R (>= 3.6.0), robustbase

Imports dplyr, ggplot2, patchwork, parallel, tidyr, Rcpp, fda, fda.usc, roahd, rrcov, Rfast, mgcv, scam, fdapace, RSpectra, MASS, rofanova, spatstat.univar, methods

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Suggests covr, knitr, rmarkdown, testthat, sn

VignetteBuilder knitr

URL https://github.com/unina-sfere/funcharts

BugReports https://github.com/unina-sfere/funcharts/issues

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Christian Capezza [cre, aut], Fabio Centofanti [aut], Davide Forcina [aut], Antonio Lepore [aut], Biagio Palumbo [aut], Alessandra Menafoglio [ctb], Simone Vantini [ctb]

Maintainer Christian Capezza <christian.capezza@unina.it>

Repository https://cran.r-universe.dev

Date/Publication 2026-01-18 23:40:09 UTC

RemoteUrl https://github.com/cran/funcharts

RemoteRef HEAD

RemoteSha 1bf52bc4f0b0fe2ea486245f78935cea7e12a329

Contents

[.mfd . . . 4
abline_mfd . . . 5
air . . . 6
AMFCC_PhaseI . . . 7
AMFCC_PhaseII . . . 9
AMFEWMA_PhaseI . . . 12
AMFEWMA_PhaseII . . . 15
cbind_mfd . . . 17
cont_plot . . . 18
control_charts_pca . . . 19
control_charts_pca_mfd_real_time . . . 22
control_charts_sof_pc . . . 24
control_charts_sof_pc_real_time . . . 26
cor_mfd . . . 28
cov_mfd . . . 29
data_sim_mfd . . . 30
estimate_mixture . . . 31
FMRCC_PhaseI . . . 32
FMRCC_PhaseII . . . 34
fof_pc . . . 36
fof_pc_real_time . . . 39
FRTM_PhaseI . . . 40
FRTM_PhaseII . . . 43
functional_filter . . . 44
get_mfd_array . . . 46
get_mfd_array_real_time . . . 47
get_mfd_df . . . 48
get_mfd_df_real_time . . . 51
get_mfd_fd . . . 52
get_mfd_list . . . 53
get_mfd_list_real_time . . . 55
get_ooc . . . 56
get_outliers_mfd . . . 57
get_sof_pc_outliers . . . 58
inprod_mfd . . . 59
inprod_mfd_diag . . . 60
is.mfd . . . 61

lines.mfd . . . 61
lines_mfd . . . 62
mean.mfd . . . 63
mfd . . . 64
mFPCA . . . 65
minus_mfd . . . 67
mixregfit_multivariate . . . 68
nbasis . . . 69
nobs.mfd . . . 69
norm.mfd . . . 70
nvar . . . 70
OEBFDTW . . . 71
par.FDTW . . . 73
par.mFPCA . . . 75
par.rtr . . . 76
pca_mfd . . . 77
pca_mfd_real_time . . . 78
plot.AMFCC_PhaseI . . . 79
plot.FRTM_PhaseI . . . 80
plot.mfd . . . 82
plot.mFPCA . . . 82
plot_bifd . . . 83
plot_bootstrap_sof_pc . . . 84
plot_control_charts . . . 85
plot_control_charts_real_time . . . 86
plot_mfd . . . 87
plot_mon . . . 88
plot_pca_mfd . . . 90
plus_mfd . . . 90
predict.pca_mfd . . . 92
predict_fof_pc . . . 93
predict_sof_pc . . . 94
rbind_mfd . . . 95
regr_cc_fof . . . 96
regr_cc_fof_real_time . . . 98
regr_cc_sof . . . 100
regr_cc_sof_real_time . . . 102
RoAMFEWMA_PhaseI . . . 104
RoAMFEWMA_PhaseII . . . 107
RoMFCC_PhaseI . . . 109
RoMFCC_PhaseI_casewise . . . 111
RoMFCC_PhaseII . . . 113
RoMFCC_PhaseII_casewise . . . 114
RoMFDI . . . 116
rpca_mfd . . . 117
scale_mfd . . . 119
sim_funcharts . . . 120
simulate_data_fmrcc . . . 121

simulate_data_FRTM 124
simulate_data_RoMFCC 125
simulate_mfd 127
sof_pc 130
sof_pc_real_time 132
tensor_product_mfd 134
times_mfd 135
which_ooc 136

Index **138**

`[.mfd` *Extract observations and/or variables from* `mfd` *objects.*

Description

Extract observations and/or variables from `mfd` objects.

Usage

```
## S3 method for class 'mfd'
mfdobj[i = TRUE, j = TRUE]
```

Arguments

`mfdobj`	An object of class `mfd`.
`i`	Index specifying functional observations to extract or replace. They can be numeric, character, or logical vectors or empty (missing) or NULL. Numeric values are coerced to integer as by as.integer (and hence truncated towards zero). The can also be negative integers, indicating functional observations to leave out of the selection. Logical vectors indicate TRUE for the observations to select. Character vectors will be matched to the argument `fdnames[[2]]` of `mfdobj`, i.e. to functional observations' names.
`j`	Index specifying functional variables to extract or replace. They can be numeric, logical, or character vectors or empty (missing) or NULL. Numeric values are coerced to integer as by as.integer (and hence truncated towards zero). The can also be negative integers, indicating functional variables to leave out of the selection. Logical vectors indicate TRUE for the variables to select. Character vectors will be matched to the argument `fdnames[[3]]` of `mfdobj`, i.e. to functional variables' names.

Details

This function adapts the `fda::"[.fd"` function to be more robust and suitable for the `mfd` class. In fact, whatever the number of observations or variables you want to extract, it always returns a `mfd` object with a three-dimensional coef array. In other words, it behaves as you would always use the argument `drop=FALSE`. Moreover, you can extract observations and variables both by index numbers and by names, as you would normally do when using `` `[` `` with standard vector/matrices.

Value

a `mfd` object with selected observations and variables.

Examples

```
library(funcharts)
library(fda)

# In the following, we extract the first one/two observations/variables
# to see the difference with `[.fd`.
mfdobj <- data_sim_mfd()
fdobj <- fd(mfdobj$coefs, mfdobj$basis, mfdobj$fdnames)

# The argument `coef` in `fd`
# objects is converted to a matrix when possible.
dim(fdobj[1, 1]$coef)
# Not clear what is the second dimension:
# the number of replications or the number of variables?
dim(fdobj[1, 1:2]$coef)
dim(fdobj[1:2, 1]$coef)

# The argument `coef` in `mfd` objects is always a three-dimensional array.
dim(mfdobj[1, 1]$coef)
dim(mfdobj[1, 1:2]$coef)
dim(mfdobj[1:2, 1]$coef)

# Actually, `[.mfd` works as `[.fd` when passing also `drop = FALSE`
dim(fdobj[1, 1, drop = FALSE]$coef)
dim(fdobj[1, 1:2, drop = FALSE]$coef)
dim(fdobj[1:2, 1, drop = FALSE]$coef)
```

---

abline_mfd *Add reference lines to all panels of the current multi-panel plot*

Description

Calls `abline` in every panel actually used by the most recent call to `plot.mfd`.

Usage

```
abline_mfd(a = NULL, b = NULL, h = NULL, v = NULL, ...)
```

Arguments

a, b	the intercept and slope, single values.
h	the y-value(s) for horizontal line(s).
v	the x-value(s) for vertical line(s).
...	Further graphical parameters (e.g., `col`, `lty`, `lwd`).

Details

The function relies on `plot.mfd` having stored the number of variables in `options("last_mfd_nvar")`. It then loops over exactly that many panels in the current layout.

Calls `abline` in every panel actually used by the most recent call to `plot.mfd`. Vertical and horizontal lines span the full x- or y-range of each panel, even when scales differ.

`air` *Air quality data*

Description

This data set has been included from the R package `FRegSigCom`. The original .RData file is available at https://github.com/cran/FRegSigCom/blob/master/data/air.RData.

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

Usage

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

Format

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

NO2 Hourly observation of concentration level of NO2 in 355 days

CO Hourly observation of concentration level of CO in 355 days

NMHC Hourly observation of concentration level of NMHC in 355 days

NOx Hourly observation of concentration level of NOx in 355 days

C6H6 Hourly observation of concentration level of C6H6 in 355 days

temperature Hourly observation of concentration level of temperature in 355 days

humidity Hourly observation of concentration level of humidity in 355 days

Source

`https://archive.ics.uci.edu/ml/datasets/Air+quality`

References

De Vito, S., Massera E., Piga M., Martinotto L. and Di Francia G. (2008). On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario *Sensors and Actuators B: Chemical*, 129: 50-757. `doi:10.1016/j.snb.2007.09.060`

Xin Qi and Ruiyan Luo (2019). Nonlinear function on function additive model with multiple predictor curves. *Statistica Sinica*, 29:719-739. `doi:10.5705/ss.202017.0249`

`AMFCC_PhaseI` *Phase I of the Adaptive Multivariate Functional Control Chart (AMFCC).*

Description

This function implements the design phase (Phase I) of the Adaptive Multivariate Functional Control Chart.

Usage

```
AMFCC_PhaseI(
  data_tra,
  data_tun = NULL,
  grid,
  q = 30,
  par_seq_list = list(10^seq(-7, 2, l = 10), c(0.5, 0.7, 0.8, 0.9, 0.99)),
  alpha_diagn = 0.05,
  alpha_mon = 0.05,
  ncores = 1
)
```

Arguments

`data_tra`	a data frame with the training data with the following columns: • var: vector of the variable indexes. • curve: vector of the curve indexes. • timeindex: vector of the time indexes corresponding to given elements of `grid`. • x: concatenated vector of the observed curves.
`data_tun`	a data frame with the tuning data with the same structure as `data_tra`. If NULL, `data_tun` is set to `data_tra`.
`grid`	The vector of time points where the curves are sampled.
`q`	The dimension of the set of B-spline functions.
`par_seq_list`	a list with two elements. The first element is a sequence of values for the regularization parameter λ and the second element is a sequence of percentages of the total variability to select L.
`alpha_diagn`	Type I error probability for the diagnostic.
`alpha_mon`	Type I error probability for the monitoring.
`ncores`	number of cores to use for parallel computing

Value

A list containing the following arguments:

- `statistics_IC`: A matrix with the values of the Hotelling T^2-type statistics for each observation and parameter combination.
- `p_values_combined`: A list with two elements containing the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- `CL`: The control limits for the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- `contributions_IC`: A list where each element corresponds to a variable and is a matrix with the contributions to the Hotelling T^2-type statistics for each observation and parameter combination.
- `p_values_combined_cont`: A list where each element corresponds to a variable and is a list of two elements containing the contribution to the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- `CL_cont`: The control limits for the contribution to the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- `par_seq_list`: The list of the sequences of the tuning parameters.
- `q`: The dimension of the set of B-spline functions.
- `basis`: The basis functions used for the functional data representation.
- `grid`: The vector of time points where the curves are sampled.
- `comb_list_tot`: The matrix with all the parameter combinations.
- `mod_pca_list`: The list of the MFPCA models for each value of `lambda_s`.

References

Centofanti, F., A. Lepore, and B. Palumbo (2025). An Adaptive Multivariate Functional Control Chart. Accepted for publication in *Technometrics*.

Examples

```
library(funcharts)
N <- 10
l_grid <- 10
p <- 2
grid <- seq(0, 1, l = l_grid)


Xall_tra <- funcharts::simulate_mfd(
  nobs = N,
  p = p,
  ngrid = l_grid,
  correlation_type_x = c("Bessel", "Gaussian")
)
X_tra <-
  data.frame(
    x = c(Xall_tra$X_list[[1]], Xall_tra$X_list[[2]]),
```

```
    timeindex = rep(rep(1:l_grid, each = (N)), p),
    curve = rep(1:(N), l_grid * p),
    var = rep(1:p, each = l_grid * N)
  )

Xall_II <- funcharts::simulate_mfd(
  nobs = N,
  p = p,
  ngrid = l_grid,
  shift_type_x = list("A", "B"),
  d_x = c(10, 10),
  correlation_type_x = c("Bessel", "Gaussian")
)

X_II <-
  data.frame(
    x = c(Xall_II$X_list[[1]], Xall_II$X_list[[2]]),
    timeindex = rep(rep(1:l_grid, each = (N)), p),
    curve = rep(1:(N), l_grid * p),
    var = rep(1:p, each = l_grid * N)
  )

# AMFCC ------------------------------------------------------------------
print("AMFCC")

mod_phaseI_AMFCC <- AMFCC_PhaseI(
  data_tra = X_tra,
  data_tun =
    NULL,
  grid = grid,
  ncores = 1
)

mod_phaseII_AMFCC <- AMFCC_PhaseII(data = X_II,
mod_Phase_I = mod_phaseI_AMFCC,
ncores = 1)

plot(mod_phaseII_AMFCC)
plot(mod_phaseII_AMFCC,type='cont',ind_obs=1)
```

`AMFCC_PhaseII` *Phase II of the Adaptive Multivariate Functional Control Chart (AMFCC).*

Description

This function implements the monitoring phase (Phase II) of the Adaptive Multivariate Functional Control Chart.

Usage

```
AMFCC_PhaseII(data = NULL, mod_Phase_I, ncores = 1)
```

Arguments

data	a data frame with the testing data with the following columns: var: vector of the variable indexes. curve: vector of the curve indexes. timeindex: vector of the time indexes corresponding to given elements of \code{grid x: concatenated vector of the observed curves.
mod_Phase_I	a list with the output of the Phase I.
ncores	number of cores to use for parallel computing

Value

A list containing the following arguments:

- ARL: The average run length (ARL) for the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- ARL_cont: The average run length for the contribution to the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- statistics: A matrix with the values of the Hotelling T^2-type statistics for each observation and parameter combination.
- contributions: A list where each element is a matrix with the contributions to the Hotelling T^2-type statistics for each observation and parameter combination.
- p_values_combined: A list with two elements containing the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- p_values_combined_cont: A list where each element is a list of two elements containing the contribution to the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- CL: The control limits for the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.
- CL_cont: The control limits for the contribution to the monitoring statistics obtained with the Fisher omnibus and Tippett combining functions.

References

Centofanti, F., A. Lepore, and B. Palumbo (2025). An Adaptive Multivariate Functional Control Chart. Accepted for publication in *Technometrics*.

Examples

```
library(funcharts)
N <- 10
l_grid <- 10
p <- 2
grid <- seq(0, 1, l = l_grid)


Xall_tra <- funcharts::simulate_mfd(
  nobs = N,
  p = p,
  ngrid = l_grid,
  correlation_type_x = c("Bessel", "Gaussian")
)
X_tra <-
  data.frame(
    x = c(Xall_tra$X_list[[1]], Xall_tra$X_list[[2]]),
    timeindex = rep(rep(1:l_grid, each = (N)), p),
    curve = rep(1:(N), l_grid * p),
    var = rep(1:p, each = l_grid * N)
  )

Xall_II <- funcharts::simulate_mfd(
  nobs = N,
  p = p,
  ngrid = l_grid,
  shift_type_x = list("A", "B"),
  d_x = c(10, 10),
  correlation_type_x = c("Bessel", "Gaussian")
)

X_II <-
  data.frame(
    x = c(Xall_II$X_list[[1]], Xall_II$X_list[[2]]),
    timeindex = rep(rep(1:l_grid, each = (N)), p),
    curve = rep(1:(N), l_grid * p),
    var = rep(1:p, each = l_grid * N)
  )

# AMFCC ------------------------------------------------------------------
print("AMFCC")

mod_phaseI_AMFCC <- AMFCC_PhaseI(
  data_tra = X_tra,
  data_tun =
    NULL,
  grid = grid,
  ncores = 1
)

mod_phaseII_AMFCC <- AMFCC_PhaseII(data = X_II,
mod_Phase_I = mod_phaseI_AMFCC,
```

```
ncores = 1)

plot(mod_phaseII_AMFCC)
plot(mod_phaseII_AMFCC,type='cont',ind_obs=1)
```

`AMFEWMA_PhaseI` *Adaptive Multivariate Functional EWMA control chart - Phase I*

Description

This function performs Phase I of the Adaptive Multivariate Functional EWMA (AMFEWMA) control chart proposed by Capezza et al. (2024)

Usage

```
AMFEWMA_PhaseI(
  mfdobj,
  mfdobj_tuning,
  lambda = NULL,
  k = NULL,
  ARL0 = 200,
  bootstrap_pars = list(n_seq = 200, l_seq = 2000),
 optimization_pars = list(lambda_grid = c(0.1, 0.2, 0.3, 0.5, 1), k_grid = c(1, 2, 3,
    4), epsilon = 0.1, sd_small = 0.25, sd_big = 2),
  discrete_grid_length = 25,
  score_function = "huber",
  fev = 0.9,
  n_skip = 100
)
```

Arguments

`mfdobj`	An object of class `mfd` containing the Phase I multivariate functional data set, to be used to train the multivariate functional principal component analysis model.
`mfdobj_tuning`	An object of class `mfd` containing the Phase I multivariate functional data set, to be used as tuning data set to estimate the AMFEWMA control chart limit.
`lambda`	lambda parameter to be used in the score function. See Equation (7) or (8) of Capezza et al. (2024). If it is provided, it must be a number between zero and one. If NULL, it is chosen through the selected according to the optimization procedure presented in Section 2.4 of Capezza et al. (2024). In this case, it is chosen among the values of `optimization_pars$lambda_grid`. Default value is NULL.
`k`	k parameter to be used in the score function. See Equation (7) or (8) of Capezza et al. (2024). If it is provided, it must be a number greater than zero. If NULL, it is chosen through the selected according to the optimization procedure presented in Section 2.4 of Capezza et al. (2024). In this case, it is chosen among the values of `optimization_pars$k_grid`. Default value is NULL.

`ARL0` The nominal in-control average run length. Default value is 200.

`bootstrap_pars` Parameters of the bootstrap procedure described in Section 2.4 of Capezza et al. (2024) for the estimation of the control chart limit. It must be a list with two arguments. `n_seq` is the number of bootstrap sequences to be generated. `l_seq` is the length of each bootstrap sequence, i.e., the number of observations to be sampled with replacement from the tuning set. Default value is `list(n_seq = 200, l_seq = 2000)`.

`optimization_pars`
Parameters to be used in the optimization procedure described in Section 2.4 of Capezza et al. (2024) for the selection of the parameters lambda and k. It must be a list of the following parameters. `lambda_grid` contains the possible values of the parameter `lambda`. `k_grid` contains the possible values of the parameter `k`. `epsilon` is the parameter used in Equation (10) of Capezza et al. (2024). When performing the parameter optimization, first the parameters lambda and k are selected to minimize the ARL with respect to a large shift, then the same parameters are chosen to minimize the ARL with respect to a small shift, given that the resulting ARL with respect to the previous large shift does not increase, in percentage, more than `epsilon`*100. Default value is 0.1. `sd_small` is a positive constant that multiplies the standard deviation function to define the small shift delta_1 in Section 2.4 of Capezza et al. (2024). In fact, the small shift is defined as delta_1(t) = mu_0(t) + `sd_small` * sigma(t), where mu_0(t) is the estimated in-control mean function and sigma(t) is the estimated standard deviation function. Default value is 0.25. `sd_big` is a positive constant that multiplies the standard deviation function to define the large shift delta_2 in Section 2.4 of Capezza et al. (2024). In fact, the large shift is defined as delta_2(t) = mu_0(t) + `sd_large` * sigma(t), where mu_0(t) is the estimated in-control mean function and sigma(t) is the estimated standard deviation function. Default value is 2.

`discrete_grid_length`
The number of equally spaced argument values at which the `mfd` objects are discretized. Default value is 25.

`score_function` Score function to be used in Equation (7) or (8) of Capezza et al. (2024), to calculate the weighting parameter of the EWMA statistic for each observation of the sequence. Two values are possible. If "huber", it uses the score function (7) inspired by the Huber's function. If "tukey", it uses the score function (8) inspired by the Tukey's bisquare function.

`fev` Number between 0 and 1 denoting the fraction of variability that must be explained by the principal components to be selected after applying multivariate functional principal component analysis on `mfdobj`. Default is 0.9.

`n_skip` The upper control limit of the AMFEWMA control chart is set to achieve a desired in-control ARL, evaluated after the monitoring statistic has reached steady state. A monitoring statistic is in a steady state if the process has been in control long enough for the effect of the starting value to become negligible (Lucas and Saccucci, 1990). In this regard, the first `n_skip` observations are excluded from the calculation of the run length. Default value is 100.

Value

A list with the following elements. `lambda` is the selected lambda parameter. `k` is the selected k parameter. `mod_1` contains the estimated Phase I model. It is a list with the following elements.

- `mfdobj` the `mfdobj` object passed as input to this function,
- `mfdobj_tuning` the `mfdobj_tuning` object passed as input to this function,
- `inv_sigmaY_reg`: the matrix containing the discretized version of the function K^*(s,t) defined in Equation (9) of Capezza et al. (2024),
- `mean_mfdobj`: the estimated mean function,
- `h`: the calculated upper control limit of the AMFEWMA control chart,
- `ARL0`: the estimated in-control ARL, which should be close to the nominal value passed as input to this function,
- `lambda`: the lambda parameter selected by the optimization procedure described in Section 2.4 of Capezza et al. (2024).
- `k`: The function C_j(t)=k sigma_j(t) appearing in the score functions (7) and (8) of Capezza et al. (2024).
- `grid_points`: the grid containing the points over which the functional data are discretized before computing the AMFEWMA monitoring statistic and estimating all the model parameters.
- `V2_mat`: the `n_seqXl_seq` matrix containing, in each column, the AMFEWMA monitoring statistic values of each bootstrap sequence. This matrix is used to set the control chart limit `h` to ensure that the desired average run length is achieved.
- `n_skip`: the `n_skip` input parameter passed to this function,
- `huber`: if the input parameter `score_function` is `"huber"`, this is TRUE, else is FALSE,
- `vectors`: the discretized eigenfunctions psi_l(t) of the covariance function, appearing in Equation (9) of Capezza et al. (2024).
- `values`: the eigenvalues rho_l of the covariance function, appearing in Equation (9) of Capezza et al. (2024).

References

Capezza, C., Capizzi, G., Centofanti, F., Lepore, A., Palumbo, B. (2025) An Adaptive Multivariate Functional EWMA Control Chart. *Journal of Quality Technology*, 57(1):1–15, doi:https://doi.org/10.1080/00224065.2024.23

Lucas, J. M., Saccucci, M. S. (1990) Exponentially weighted moving average control schemes: properties and enhancements. *Technometrics*, 32(1), 1-12.

Examples

```
set.seed(0)
library(funcharts)
dat_I <- simulate_mfd(nobs = 200,
                      correlation_type_x = c("Bessel", "Bessel", "Bessel"),
                      sd_x = c(0.3, 0.3, 0.3))
dat_tun <- simulate_mfd(nobs = 200,
                        correlation_type_x = c("Bessel", "Bessel", "Bessel"),
```

```
                        sd_x = c(0.3, 0.3, 0.3))
dat_II <- simulate_mfd(nobs = 20,
                       correlation_type_x = c("Bessel", "Bessel", "Bessel"),
                       shift_type_x = c("C", "C", "C"),
                       d_x = c(2, 2, 2),
                       sd_x = c(0.3, 0.3, 0.3))
mfdobj_I <- get_mfd_list(dat_I$X_list, lambda = 1e-2)
mfdobj_tun <- get_mfd_list(dat_tun$X_list, lambda = 1e-2)
mfdobj_II <- get_mfd_list(dat_II$X_list, lambda = 1e-2)

# p <- plot_mfd(mfdobj_I[1:100])
# lines_mfd(p, mfdobj_II, col = "red")


mod <- AMFEWMA_PhaseI(mfdobj = mfdobj_I,
                      mfdobj_tuning = mfdobj_tun,
                      lambda = 0.1,
                      k = c(1, 2))

cc <- AMFEWMA_PhaseII(mfdobj_2 = rbind_mfd(mfdobj_I[1:100], mfdobj_II),
                      mod_1 = mod)
plot_control_charts(cc$cc, nobsI = 100)
```

`AMFEWMA_PhaseII` *Adaptive Multivariate Functional EWMA control chart - Phase II*

Description

This function performs Phase II of the Adaptive Multivariate Functional EWMA (AMFEWMA) control chart proposed by Capezza et al. (2024)

Usage

```
AMFEWMA_PhaseII(mfdobj_2, mod_1, n_seq_2 = 1, l_seq_2 = 2000)
```

Arguments

`mfdobj_2`	An object of class `mfd` containing the Phase II multivariate functional data set, to be monitored with the AMFEWMA control chart.
`mod_1`	The output of the Phase I achieved through the `AMFEWMA_PhaseI` function.
`n_seq_2`	If it is 1, the Phase II monitoring statistic is calculated on the data sequence. If it is an integer number larger than 1, a number `n_seq_2` of bootstrap sequences are sampled with replacement from `mfdobj_2` to allow uncertainty quantification on the estimation of the run length. Default value is 1.
`l_seq_2`	If `n_seq_2` is larger than 1, this parameter sets the length of each bootstrap sequence to be generated. Default value is 2000 (which is ignored if the default value

Value

A list with the following elements.

- ARL_2: the average run length estimated over the bootstrap sequences. If n_seq_2 is 1, it is simply the run length observed over the Phase II sequence, i.e., the number of observations up to the first alarm,
- RL: the run length observed over the Phase II sequence, i.e., the number of observations up to the first alarm,
- V2: a list with length n_seq_2, containing the AMFEWMA monitoring statistic in Equation (8) of Capezza et al. (2024), calculated in each bootstrap sequence, until the first alarm.
- cc: a data frame with the information needed to plot the AMFEWMA control chart in Phase II, with the following columns. id contains the id of each multivariate functional observation, amfewma_monitoring_statistic contains the AMFEWMA monitoring statistic values calculated on the Phase II sequence, amfewma_monitoring_statistic_lim is the upper control limit.

References

Capezza, C., Capizzi, G., Centofanti, F., Lepore, A., Palumbo, B. (2025) An Adaptive Multivariate Functional EWMA Control Chart. *Journal of Quality Technology*, 57(1):1–15, doi:https://doi.org/10.1080/00224065.2024.23

Examples

```
set.seed(0)
library(funcharts)
dat_I <- simulate_mfd(nobs = 200,
                      correlation_type_x = c("Bessel", "Bessel", "Bessel"),
                      sd_x = c(0.3, 0.3, 0.3))
dat_tun <- simulate_mfd(nobs = 200,
                        correlation_type_x = c("Bessel", "Bessel", "Bessel"),
                        sd_x = c(0.3, 0.3, 0.3))
dat_II <- simulate_mfd(nobs = 20,
                       correlation_type_x = c("Bessel", "Bessel", "Bessel"),
                       shift_type_x = c("C", "C", "C"),
                       d_x = c(2, 2, 2),
                       sd_x = c(0.3, 0.3, 0.3))
mfdobj_I <- get_mfd_list(dat_I$X_list, lambda = 1e-2)
mfdobj_tun <- get_mfd_list(dat_tun$X_list, lambda = 1e-2)
mfdobj_II <- get_mfd_list(dat_II$X_list, lambda = 1e-2)

# p <- plot_mfd(mfdobj_I[1:100])
# lines_mfd(p, mfdobj_II, col = "red")


mod <- AMFEWMA_PhaseI(mfdobj = mfdobj_I,
                      mfdobj_tuning = mfdobj_tun,
                      lambda = 0.1,
                      k = c(1, 2))

cc <- AMFEWMA_PhaseII(mfdobj_2 = rbind_mfd(mfdobj_I[1:100], mfdobj_II),
```

```
                        mod_1 = mod)
plot_control_charts(cc$cc, nobsI = 100)
```

---

cbind_mfd *Bind variables of two Multivariate Functional Data Objects*

Description

Bind variables of two Multivariate Functional Data Objects

Usage

```
cbind_mfd(mfdobj1, mfdobj2)
```

Arguments

mfdobj1	An object of class mfd, with the same number of replications of mfdobj2 and different variable names with respect to mfdobj2.
mfdobj2	An object of class mfd, with the same number of replications of mfdobj1, and different variable names with respect to mfdobj1.

Value

An object of class mfd, whose replications are the same of mfdobj1 and mfdobj2 and whose functional variables are the union of the functional variables in mfdobj1 and mfdobj2.

Examples

```
library(funcharts)
mfdobj1 <- data_sim_mfd(nvar = 3)
mfdobj2 <- data_sim_mfd(nvar = 2)
dimnames(mfdobj2$coefs)[[3]] <- mfdobj2$fdnames[[3]] <- c("var10", "var11")

plot_mfd(mfdobj1)
plot_mfd(mfdobj2)
mfdobj_cbind <- cbind_mfd(mfdobj1, mfdobj2)
plot_mfd(mfdobj_cbind)
```

cont_plot *Produce contribution plots*

Description

This function produces a contribution plot from functional control charts for a given observation of a phase II data set, using ggplot.

Usage

```
cont_plot(cclist, id_num, which_plot = c("T2", "spe"), print_id = FALSE)
```

Arguments

cclist	A data.frame produced by control_charts_pca, control_charts_sof_pc regr_cc_fof, or regr_cc_sof.
id_num	An index number giving the observation in the phase II data set to be plotted, i.e. 1 for the first observation, 2 for the second, and so on.
which_plot	A character vector. Each value indicates which contribution you want to plot: "T2" indicates contribution to the Hotelling's T2 statistic, "spe" indicates contribution to the squared prediction error statistic.
print_id	A logical value, if TRUE, it prints also the id of the observation in the title of the ggplot. Default is FALSE.

Value

A ggplot containing the contributions of functional variables to the monitoring statistics. Each plot is a bar plot, with bars corresponding to contribution values and horizontal black segments denoting corresponding (empirical) upper limits. Bars are coloured by red if contributions exceed their limit.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[201:300, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:60]
y_tuning <- y[61:90]
y2 <- y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod <- sof_pc(y1, mfdobj_x1)
```

```
cclist <- regr_cc_sof(object = mod,
                      y_new = y2,
                      mfdobj_x_new = mfdobj_x2,
                      y_tuning = y_tuning,
                      mfdobj_x_tuning = mfdobj_x_tuning,
                      include_covariates = TRUE)
get_ooc(cclist)
cont_plot(cclist, 3)
```

`control_charts_pca` *T2 and SPE control charts for multivariate functional data*

Description

This function builds a data frame needed to plot the Hotelling's T2 and squared prediction error (SPE) control charts based on multivariate functional principal component analysis (MFPCA) performed on multivariate functional data, as Capezza et al. (2020) for the multivariate functional covariates. The training data have already been used to fit the model. An optional tuning data set can be provided to estimate the control chart limits. A phase II data set contains the observations to be monitored with the control charts.

Usage

```
control_charts_pca(
  pca,
  components = NULL,
  tuning_data = NULL,
  newdata,
  alpha = 0.05,
  limits = "standard",
  seed,
  nfold = 5,
  ncores = 1,
  tot_variance_explained = 0.9,
  single_min_variance_explained = 0,
  absolute_error = FALSE
)
```

Arguments

`pca`	An object of class `pca_mfd` obtained by doing MFPCA on the training set of multivariate functional data.
`components`	A vector of integers with the components over which to project the multivariate functional data. If this is not NULL, the arguments `single_min_variance_explained` and `tot_variance_explained` are ignored. If NULL, components are selected such that the total fraction of variance explained by them is at least equal to the

argument tot_variance_explained, where only components explaining individually a fraction of variance at least equal to the argument single_min_variance_explained are considered to be retained. Default is NULL.

tuning_data An object of class mfd containing the tuning set of the multivariate functional data, used to estimate the T2 and SPE control chart limits. If NULL, the training data, i.e. the data used to fit the MFPCA model, are also used as the tuning data set, i.e. tuning_data=pca$data. Default is NULL.

newdata An object of class mfd containing the phase II set of the multivariate functional data to be monitored.

alpha If it is a number between 0 and 1, it defines the overall type-I error probability and the Bonferroni correction is applied by setting the type-I error probability in the two control charts equal to alpha/2. If you want to set manually the Type-I error probabilities in the two control charts, then the argument alpha must be a named list with two elements, named T2 and spe, respectively, each containing the desired Type I error probability of the corresponding control chart. Default value is 0.05.

limits A character value. If "standard", it estimates the control limits on the tuning data set. If "cv", the function calculates the control limits only on the training data using cross-validation using calculate_cv_limits. Default is "standard".

seed If limits=="cv", since the split in the k groups is random, you can fix a seed to ensure reproducibility. Deprecated: use set.seed() before calling the function for reproducibility.

nfold If limits=="cv", this gives the number of groups k used for k-fold cross-validation. If it is equal to the number of observations in the training data set, then we have leave-one-out cross-validation. Otherwise, this argument is ignored.

ncores If limits=="cv", if you want perform the analysis in the k groups in parallel, give the number of cores/threads. Otherwise, this argument is ignored.

tot_variance_explained
The minimum fraction of variance that has to be explained by the set of multivariate functional principal components retained into the MFPCA model fitted on the functional covariates. Default is 0.9.

single_min_variance_explained
The minimum fraction of variance that has to be explained by each multivariate functional principal component such that it is retained into the MFPCA model. Default is 0.

absolute_error If FALSE, the SPE statistic, which monitors the principal components not retained in the MFPCA model, is calculated as the sum of the integrals of the squared prediction error functions, obtained as the difference between the actual functions and their approximation after projection over the selected principal components. If TRUE, the SPE statistic is calculated by replacing the square of the prediction errors with the absolute value, as proposed by Capizzi and Masarotto (2018). Default value is FALSE.

Value

A `data.frame` with as many rows as the number of multivariate functional observations in the phase II data set and the following columns:

- one `id` column identifying the multivariate functional observation in the phase II data set,
- one `T2` column containing the Hotelling T2 statistic calculated for all observations,
- one column per each functional variable, containing its contribution to the T2 statistic,
- one `spe` column containing the SPE statistic calculated for all observations,
- one column per each functional variable, containing its contribution to the SPE statistic,
- `T2_lim` gives the upper control limit of the Hotelling's T2 control chart,
- one `contribution_T2_*_lim` column per each functional variable giving the limits of the contribution of that variable to the Hotelling's T2 statistic,
- `spe_lim` gives the upper control limit of the SPE control chart
- one `contribution_spe*_lim` column per each functional variable giving the limits of the contribution of that variable to the SPE statistic.

References

Capezza C, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2020) Control charts for monitoring ship operating conditions and CO2 emissions based on scalar-on-function regression. *Applied Stochastic Models in Business and Industry*, 36(3):477–500. doi:10.1002/asmb.2507

Capizzi, G., & Masarotto, G. (2018). Phase I distribution-free analysis with the R package dfphase1. In *Frontiers in Statistical Quality Control 12 (pp. 3-19)*. Springer International Publishing.

See Also

`regr_cc_fof`

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:220, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:100]
y_tuning <- y[101:200]
y2 <- y[201:220]
mfdobj_x1 <- mfdobj_x[1:100]
mfdobj_x_tuning <- mfdobj_x[101:200]
mfdobj_x2 <- mfdobj_x[201:220]
pca <- pca_mfd(mfdobj_x1)
cclist <- control_charts_pca(pca = pca,
                             tuning_data = mfdobj_x_tuning,
                             newdata = mfdobj_x2)
```

```
plot_control_charts(cclist)
```

`control_charts_pca_mfd_real_time` *Real-time T2 and SPE control charts for multivariate functional data*

Description

This function produces a list of data frames, each of them is produced by `control_charts_pca` and is needed to plot control charts for monitoring multivariate functional covariates each evolving up to an intermediate domain point.

Usage

```
control_charts_pca_mfd_real_time(
  pca_list,
  components_list = NULL,
  mfdobj_x_test,
  mfdobj_x_tuning = NULL,
  alpha = 0.05,
  limits = "standard",
  seed,
  nfold = NULL,
  tot_variance_explained = 0.9,
  single_min_variance_explained = 0,
  absolute_error = FALSE,
  ncores = 1
)
```

Arguments

`pca_list`	A list of lists produced by `pca_mfd_real_time`, containing a list of multivariate functional principal component analysis models estimated on functional data each evolving up to an intermediate domain point.
`components_list`	A list of components given as input to `pca_mfd` for each intermediate domain point.
`mfdobj_x_test`	A list created using `get_mfd_df_real_time` or `get_mfd_list_real_time`, denoting a list of functional data objects in the phase II monitoring data set, each evolving up to an intermediate domain point, with observations of the multivariate functional data. The length of this list and `pca_list` must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point.

mfdobj_x_tuning
: A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the tuning data set (used to estimate control chart limits), each evolving up to an intermediate domain point, with observations of the multivariate functional data The length of this list and pca_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point. If NULL, the training data, i.e. the functional data in pca_list, are also used as the tuning data set. Default is NULL.

alpha
: See control_charts_pca.

limits
: See control_charts_pca.

seed
: Deprecated: See control_charts_pca.

nfold
: See control_charts_pca.

tot_variance_explained
: See control_charts_pca.

single_min_variance_explained
: See control_charts_pca.

absolute_error
: See control_charts_pca.

ncores
: If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of data.frames each produced by control_charts_pca, corresponding to a given instant.

See Also

pca_mfd_real_time, control_charts_pca

Examples

```
library(funcharts)
data("air")
air1 <- lapply(air, function(x) x[1:8, , drop = FALSE])
air2 <- lapply(air, function(x) x[9:10, , drop = FALSE])
mfdobj_x1_list <- get_mfd_list_real_time(air1[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_x2_list <- get_mfd_list_real_time(air2[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
pca_list <- pca_mfd_real_time(mfdobj_x1_list)

cclist <- control_charts_pca_mfd_real_time(
  pca_list = pca_list,
  components_list = 1:3,
  mfdobj_x_test = mfdobj_x2_list)
```

```
plot_control_charts_real_time(cclist, 1)
```

`control_charts_sof_pc` *Control charts for monitoring a scalar quality characteristic adjusted for by the effect of multivariate functional covariates*

Description

This function builds a data frame needed to plot control charts for monitoring a monitoring a scalar quality characteristic adjusted for the effect of multivariate functional covariates based on scalar-on-function regression, as proposed in Capezza et al. (2020).

In particular, this function provides:

- the Hotelling's T2 control chart,
- the squared prediction error (SPE) control chart,
- the scalar regression control chart.

This function calls `control_charts_pca` for the control charts on the multivariate functional covariates and `regr_cc_sof` for the scalar regression control chart.

The training data have already been used to fit the model. An optional tuning data set can be provided that is used to estimate the control chart limits. A phase II data set contains the observations to be monitored with the control charts.

Usage

```
control_charts_sof_pc(
  mod,
  y_test,
  mfdobj_x_test,
  mfdobj_x_tuning = NULL,
  alpha = list(T2 = 0.0125, spe = 0.0125, y = 0.025),
  limits = "standard",
  seed,
  nfold = NULL,
  ncores = 1
)
```

Arguments

`mod`	A list obtained as output from `sof_pc`, i.e. a fitted scalar-on-function linear regression model.
`y_test`	A numeric vector containing the observations of the scalar response variable in the phase II data set.
`mfdobj_x_test`	An object of class `mfd` containing the phase II data set of the functional covariates observations.

`mfdobj_x_tuning`
: An object of class `mfd` containing the tuning set of the multivariate functional data, used to estimate the T2 and SPE control chart limits. If NULL, the training data, i.e. the data used to fit the MFPCA model, are also used as the tuning data set, i.e. `tuning_data=pca$data`. Default is NULL.

`alpha`
: A named list with three elements, named `T2`, `spe`, and `y`, respectively, each containing the desired Type I error probability of the corresponding control chart (`T2` corresponds to the T2 control chart, `spe` corresponds to the SPE control chart, `y` corresponds to the scalar regression control chart). Note that at the moment you have to take into account manually the family-wise error rate and adjust the two values accordingly. See Capezza et al. (2020) for additional details. Default value is `list(T2 = 0.0125, spe = 0.0125, y = 0.025)`.

`limits`
: A character value. If "standard", it estimates the control limits on the tuning data set. If "cv", the function calculates the control limits only on the training data using cross-validation using `calculate_cv_limits`. Default is "standard".

`seed`
: If `limits=="cv"`, since the split in the k groups is random, you can fix a seed to ensure reproducibility. Deprecated: use `set.seed()` before calling the function for reproducibility.

`nfold`
: If `limits=="cv"`, this gives the number of groups k used for k-fold cross-validation. If it is equal to the number of observations in the training data set, then we have leave-one-out cross-validation. Otherwise, this argument is ignored.

`ncores`
: If `limits=="cv"`, if you want perform the analysis in the k groups in parallel, give the number of cores/threads. Otherwise, this argument is ignored.

Value

A `data.frame` with as many rows as the number of multivariate functional observations in the phase II data set and the following columns:

- one `id` column identifying the multivariate functional observation in the phase II data set,
- one `T2` column containing the Hotelling T2 statistic calculated for all observations,
- one column per each functional variable, containing its contribution to the T2 statistic,
- one `spe` column containing the SPE statistic calculated for all observations,
- one column per each functional variable, containing its contribution to the SPE statistic,
- `T2_lim` gives the upper control limit of the Hotelling's T2 control chart,
- one `contribution_T2_*_lim` column per each functional variable giving the limits of the contribution of that variable to the Hotelling's T2 statistic,
- `spe_lim` gives the upper control limit of the SPE control chart
- one `contribution_spe*_lim` column per each functional variable giving the limits of the contribution of that variable to the SPE statistic.
- `y_hat`: the predictions of the response variable corresponding to `mfdobj_x_new`,
- `y`: the same as the argument `y_new` given as input to this function,
- `lwr`: lower limit of the `1-alpha` prediction interval on the response,

- pred_err: prediction error calculated as y-y_hat,
- pred_err_sup: upper limit of the 1-alpha prediction interval on the prediction error,
- pred_err_inf: lower limit of the 1-alpha prediction interval on the prediction error.

See Also

control_charts_pca, regr_cc_sof

Examples

```
#' library(funcharts)
data("air")
air <- lapply(air, function(x) x[201:300, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:60]
y2 <- y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod <- sof_pc(y1, mfdobj_x1)
cclist <- control_charts_sof_pc(mod = mod,
                                y_test = y2,
                                mfdobj_x_test = mfdobj_x2,
                                mfdobj_x_tuning = mfdobj_x_tuning)
plot_control_charts(cclist)
```

---

control_charts_sof_pc_real_time

Real-time scalar-on-function regression control charts

Description

This function is deprecated. Use regr_cc_sof_real_time. This function produces a list of data frames, each of them is produced by control_charts_sof_pc and is needed to plot control charts for monitoring in real time a scalar quality characteristic adjusted for by the effect of multivariate functional covariates.

Usage

```
control_charts_sof_pc_real_time(
  mod_list,
  y_test,
```

```
  mfdobj_x_test,
  mfdobj_x_tuning = NULL,
  alpha = list(T2 = 0.0125, spe = 0.0125, y = 0.025),
  limits = "standard",
  seed,
  nfold = NULL,
  ncores = 1
)
```

Arguments

mod_list	A list of lists produced by sof_pc_real_time, containing a list of scalar-on-function linear regression models estimated on functional data each evolving up to an intermediate domain point.
y_test	A numeric vector containing the observations of the scalar response variable in the phase II monitoring data set.
mfdobj_x_test	A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the phase II monitoring data set, each evolving up to an intermediate domain point, with observations of the multivariate functional covariates. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point.
mfdobj_x_tuning	A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the tuning data set (used to estimate control chart limits), each evolving up to an intermediate domain point, with observations of the multivariate functional covariates. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point. If NULL, the training data, i.e. the functional covariates in mod_list, are also used as the tuning data set. Default is NULL.
alpha	See control_charts_sof_pc.
limits	See control_charts_sof_pc.
seed	Deprecated: see control_charts_sof_pc.
nfold	See control_charts_sof_pc.
ncores	If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of data.frames each produced by control_charts_sof_pc, corresponding to a given instant.

See Also

sof_pc_real_time, control_charts_sof_pc

Examples

```
library(funcharts)
data("air")
air1 <- lapply(air, function(x) x[1:8, , drop = FALSE])
air2 <- lapply(air, function(x) x[9:10, , drop = FALSE])
mfdobj_x1_list <- get_mfd_list_real_time(air1[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_x2_list <- get_mfd_list_real_time(air2[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
y1 <- rowMeans(air1$NO2)
y2 <- rowMeans(air2$NO2)
mod_list <- sof_pc_real_time(y1, mfdobj_x1_list)
cclist <- control_charts_sof_pc_real_time(
  mod_list = mod_list,
  y_test = y2,
  mfdobj_x_test = mfdobj_x2_list)
plot_control_charts_real_time(cclist, 1)
```

---

cor_mfd *Correlation Function for Multivariate Functional Data*

Description

Computes the correlation function for two multivariate functional data objects of class `mfd`.

Usage

```
cor_mfd(mfdobj1, mfdobj2 = mfdobj1)
```

Arguments

`mfdobj1`	An object of class `mfd` representing the first multivariate functional data set. It contains N observations of a p-dimensional multivariate functional variable.
`mfdobj2`	An object of class `mfd` representing the second multivariate functional data set. Defaults to `mfdobj1`. If provided, it must also contain N observations of a p-dimensional multivariate functional variable.

Details

The function calculates the correlation between all pairs of dimensions from the two multivariate functional data objects. The data is first scaled using `scale_mfd`, and the correlation is then computed as the covariance of the scaled data using `cov_mfd`.

Value

A bifd object representing the correlation function of the two input objects. The output is a collection of p^2 functional surfaces, each corresponding to the correlation between two components of the multivariate functional data.

Examples

```
library(funcharts)
data("air")
x <- get_mfd_list(air[1:3])
cor_result <- cor_mfd(x)
plot_bifd(cor_result)
```

---

cov_mfd *Covariance Function for Multivariate Functional Data*

Description

Computes the covariance function for two multivariate functional data objects of class `mfd`.

Usage

```
cov_mfd(mfdobj1, mfdobj2 = mfdobj1)
```

Arguments

`mfdobj1`	An object of class `mfd` representing the first multivariate functional data set. It contains N observations of a p-dimensional multivariate functional variable.
`mfdobj2`	An object of class `mfd` representing the second multivariate functional data set. Defaults to `mfdobj1`. If provided, it must also contain N observations of a p-dimensional multivariate functional variable.

Details

The function calculates the covariance between all pairs of dimensions from the two multivariate functional data objects. Each covariance is represented as a functional surface in the resulting bifd object. The covariance function is useful for analyzing relationships between functional variables.

Value

A bifd object representing the covariance function of the two input objects. The output is a collection of p^2 functional surfaces, each corresponding to the covariance between two components of the multivariate functional data.

Examples

```
library(funcharts)
data("air")
x <- get_mfd_list(air[1:3])
cov_result <- cov_mfd(x)
plot_bifd(cov_result)
```

`data_sim_mfd` *Simulate multivariate functional data*

Description

Simulate random coefficients and create a multivariate functional data object of class `mfd`. It is mainly for internal use, to check that the package functions work.

Usage

```
data_sim_mfd(nobs = 5, nbasis = 5, nvar = 2, seed)
```

Arguments

`nobs`	Number of functional observations to be simulated.
`nbasis`	Number of basis functions.
`nvar`	Number of functional covariates.
`seed`	Deprecated: use `set.seed()` before calling the function for reproducibility.

Value

A simulated object of class `mfd`.

Examples

```
library(funcharts)
data_sim_mfd()
```

`estimate_mixture` *Performs the estimation of gaussian mixtures of regression models and gaussian mixture models. Used in FMRCC_PhaseI.*

Description

Performs the estimation of gaussian mixtures of regression models and gaussian mixture models. Used in FMRCC_PhaseI.

Usage

```
estimate_mixture(
  y = NULL,
  x = NULL,
  ninit = 10,
  groups = 1:5,
  mode = "regression",
  intercept = TRUE,
  init_met = "kmeans",
  sigma_par = c("VVV", "EEE", "VII", "EII")
)
```

Arguments

`y`	a matrix with the scores of the response variable
`x`	a matrix with the scores of the covariates
`ninit`	the number of random starts for the model estimation. It is ignored if init_met = 'kmeans'. Default is 10
`groups`	the number of groups to consider in the model estimation. Default is 1:3
`mode`	the type of model to estimate, it can be 'regression' or 'clustering'. Default is 'regression'
`intercept`	logical, if TRUE the model includes an intercept. Default is TRUE
`init_met`	the method to initialize the model, it can be 'kmeans' or 'random'. Default is 'kmeans'
`sigma_par`	the covariance parametrization to consider in the model estimation. Default is c('VVV','EEE','VII','EII')

Value

a list with the model estimated, the residuals variance matrix and the BIC values

FMRCC_PhaseI *Phase I of the FMRCC*

Description

Performs Phase I of the Functional Mixture Regression Control Chart methodology, which consists of model estimation and control limit calculation using training and tuning datasets.

Usage

```
FMRCC_PhaseI(
  Y_train,
  X_train,
  Y_tun,
  X_tun,
  FVEy,
  FVEx,
  studentized = T,
  alpha = 0.01,
  intercept = T,
  init_met = "kmeans",
  ninit = 10,
  groups = 1:5,
  sigma_par = c("VVV", "EEE", "VII", "EII"),
  scale = T,
  ncompx = NULL,
  ncompy = NULL,
  userBwCov = NULL
)
```

Arguments

Y_train	Training response variable. Object of class 'mfd' (dense functional data) or 'list' (sparse functional data). For dense data, pca_mfd is performed. For sparse data, PACE (Yao et al., 2005) via FPCA is used.
X_train	Training predictor variables. Object of class 'mfd' (dense functional), 'matrix' (scalar), or 'list' (sparse functional). For dense data, pca_mfd is performed. For sparse data, PACE (Yao et al., 2005) via FPCA is used.
Y_tun	Tuning response variable for control limit calculation. Must be same type as Y_train.
X_tun	Tuning predictor variables for control limit calculation. Must be same type as X_train.
FVEy	Fraction of variance explained threshold for response variable.
FVEx	Fraction of variance explained threshold for covariates. Ignored if covariates are scalar.

studentized	Logical. If TRUE, statistics are studentized. Default is TRUE.
alpha	Type I error rate for control limit calculation. Default is 0.01.
intercept	Logical. If TRUE, model includes an intercept. Default is TRUE.
init_met	Initialization method: 'kmeans' or 'random'. If 'random', ninit initializations are performed and the model with lowest BIC is retained. Default is 'kmeans'.
ninit	Number of random starts for model estimation. Ignored if init_met = 'kmeans'. Default is 10.
groups	Integer vector specifying number of mixture components to consider. Default is 1:5.
sigma_par	Character vector of covariance parametrizations to consider. Options are 'VVV' (variable volume, shape, orientation), 'EEE' (equal volume, shape, orientation), 'VII' (variable volume, spherical), 'EII' (equal volume, spherical). Default is c('VVV','EEE','VII','EII').
scale	Logical. Should dense functional objects be scaled? Default is TRUE.
ncompx	Integer. Number of principal components to retain for functional covariates. If NULL, chosen according to FVEx. Default is NULL.
ncompy	Integer. Number of principal components to retain for functional response. If NULL, chosen according to FVEy. Default is NULL.
userBwCov	Bandwidth for covariance smoothing in PACE. See FPCA for details. Default is NULL.

Value

A list containing:

model	The best fitted mixture regression model
phaseI	Phase I results including control limits
estimate	Estimation results including values to studentize residuals
fpca	FPCA results for response and (if applicable) covariates
BIC_plt	ggplot object showing BIC values across models
studentized	Logical indicating if studentization was used
intercept	Logical indicating if intercept was included
type_y	Character indicating response type ('dense' or 'sparse')
type_x	Character indicating covariate type ('dense', 'sparse', or 'scalar')

References

Capezza, C., Centofanti, F., Forcina, D., Lepore, A., & Palumbo, B. (2025). Functional Mixture Regression Control Chart. Accepted for publication in *Annals of Applied Statistics*. arXiv:2410.20138.

Yao, F., Müller, H. G., & Wang, J. L. (2005). Functional data analysis for sparse longitudinal data. *Journal of the American Statistical Association*, 100(470), 577-590.

See Also

FMRCC_PhaseII, FPCA

Examples

```
# Example with dense functional data
# Length of the functional grid
l <- 100
# Number of observations
n <- 300

# Generate training in-control data with three equally-sized clusters, maximum dissimilarity
data <- simulate_data_fmrcc(n_obs = n, delta_1 = 1, delta_2 = 0.5, len_grid = l, severity = 0)
X_train_mfd <- get_mfd_list(data_list = data['X'], n_basis = 20)
Y_train_mfd <- get_mfd_list(data_list = data['Y'], n_basis = 20)

# Generate tuning in-control data with three equally-sized clusters, maximum dissimilarity
data <- simulate_data_fmrcc(n_obs = n, delta_1 = 1, delta_2 = 0.5, len_grid = l, severity = 0)
X_tun_mfd <- get_mfd_list(data_list = data['X'], n_basis = 20)
Y_tun_mfd <- get_mfd_list(data_list = data['Y'], n_basis = 20)

# Example with dense functional data
phaseI_results <- FMRCC_PhaseI(
  Y_train = Y_train_mfd,
  X_train = X_train_mfd,
  Y_tun = Y_tun_mfd,
  X_tun = X_tun_mfd,
  FVEy = 0.95,
  FVEx = 0.90,
  alpha = 0.01,
  groups = 1:3,
  sigma_par = c('VVV', 'EEE')
)

# View BIC plot
phaseI_results$BIC_plt
```

FMRCC_PhaseII *Phase II of the FMRCC*

Description

Performs Phase II of the FMRCC methodology.

Usage

```
FMRCC_PhaseII(Y_test, X_test, phaseI)
```

Arguments

Y_test	Test response variable. Must be same type as training data (mfd for dense or list for sparse functional data).
X_test	Test predictor variables. Must be same type as training data (mfd, matrix, or list).
phaseI	Output from FMRCC_PhaseI containing the trained model and parameters.

Value

A list containing:

ARL	Average Run Length
phaseII	Detailed Phase II results, a list with: • df: Data frame with columns: – id: Observation identifier – loglikelihood: Log-likelihood statistic for each observation – status: 'IC' (in-control) or 'OC' (out-of-control) • ARL: Average Run Length value

See Also

FMRCC_PhaseI

Examples

```
# Length of the functional grid
l <- 100
# Number of observations
n <- 300

# Generate training in-control data with three equally-sized clusters, maximum dissimilarity
data <- simulate_data_fmrcc(n_obs = n, delta_1 = 1, delta_2 = 0.5, len_grid = l, severity = 0)
X_train_mfd <- get_mfd_list(data_list = data['X'], n_basis = 20)
Y_train_mfd <- get_mfd_list(data_list = data['Y'], n_basis = 20)

# Generate tuning in-control data with three equally-sized clusters, maximum dissimilarity
data <- simulate_data_fmrcc(n_obs = n, delta_1 = 1, delta_2 = 0.5, len_grid = l, severity = 0)
X_tun_mfd <- get_mfd_list(data_list = data['X'], n_basis = 20)
Y_tun_mfd <- get_mfd_list(data_list = data['Y'], n_basis = 20)

# Example with dense functional data
phaseI_results <- FMRCC_PhaseI(
  Y_train = Y_train_mfd,
  X_train = X_train_mfd,
  Y_tun = Y_tun_mfd,
  X_tun = X_tun_mfd,
  FVEy = 0.95,
  FVEx = 0.90,
  alpha = 0.01,
```

```
  groups = 1:3,
  sigma_par = c('VVV', 'EEE')
)

# View BIC plot
phaseI_results$BIC_plt

# Generate out-of-control data with three equally-sized clusters, maximum dissimilarity
data <- simulate_data_fmrcc(n_obs = n, delta_1 = 1, delta_2 = 0.5, len_grid = l, severity = 2)
X_test_mfd <- get_mfd_list(data_list = data['X'], n_basis = 20)
Y_test_mfd <- get_mfd_list(data_list = data['Y'], n_basis = 20)

# Perform the monitoring of the Phase II data
phaseII_results <- FMRCC_PhaseII(
  Y_test = Y_test_mfd,
  X_test = X_test_mfd,
  phaseI = phaseI_results
)

# Check Average Run Length
phaseII_results$ARL

# View monitoring results
head(phaseII_results$phaseII$df)

# Identify out-of-control observations
oc_observations <- phaseII_results$phaseII$df[phaseII_results$phaseII$df$status == 'OC',]
oc_observations
```

`fof_pc` *Function-on-function linear regression based on principal components*

Description

Function-on-function linear regression based on principal components. This function performs multivariate functional principal component analysis (MFPCA) to extract multivariate functional principal components from the multivariate functional covariates as well as from the functional response, then it builds a linear regression model of the response scores on the covariate scores. Both functional covariates and response are standardized before the regression. See Centofanti et al. (2021) for additional details.

Usage

```
fof_pc(
  mfdobj_y,
  mfdobj_x,
  tot_variance_explained_x = 0.95,
```

```
  tot_variance_explained_y = 0.95,
  tot_variance_explained_res = 0.95,
  components_x = NULL,
  components_y = NULL,
  type_residuals = "standard"
)
```

Arguments

mfdobj_y	A multivariate functional data object of class mfd denoting the functional response variable. Although it is a multivariate functional data object, it must have only one functional variable.
mfdobj_x	A multivariate functional data object of class mfd denoting the functional covariates.
tot_variance_explained_x	The minimum fraction of variance that has to be explained by the multivariate functional principal components retained into the MFPCA model fitted on the functional covariates. Default is 0.95.
tot_variance_explained_y	The minimum fraction of variance that has to be explained by the multivariate functional principal components retained into the MFPCA model fitted on the functional response. Default is 0.95.
tot_variance_explained_res	The minimum fraction of variance that has to be explained by the multivariate functional principal components retained into the MFPCA model fitted on the functional residuals of the functional regression model. Default is 0.95.
components_x	A vector of integers with the components over which to project the functional covariates. If NULL, the first components that explain a minimum fraction of variance equal to tot_variance_explained_x is selected. #' If this is not NULL, the criteria to select components are ignored. Default is NULL.
components_y	A vector of integers with the components over which to project the functional response. If NULL, the first components that explain a minimum fraction of variance equal to tot_variance_explained_y is selected. #' If this is not NULL, the criteria to select components are ignored. Default is NULL.
type_residuals	A character value that can be "standard" or "studentized". If "standard", the MFPCA on functional residuals is calculated on the standardized covariates and response. If "studentized", the MFPCA on studentized version of the functional residuals is calculated on the non-standardized covariates and response. See Centofanti et al. (2021) for additional details.

Value

A list containing the following arguments:

- mod: an object of class lm that is a linear regression model where the response variables are the MFPCA scores of the response variable and the covariates are the MFPCA scores of the functional covariates. mod$coefficients contains the matrix of coefficients of the functional regression basis functions,

- `beta_fd`: a `bifd` object containing the bivariate functional regression coefficients $\beta(s,t)$ estimated with the function-on-function linear regression model,
- `fitted.values`: a multivariate functional data object of class mfd with the fitted values of the functional response observations based on the function-on-function linear regression model,
- `residuals_original_scale`: a multivariate functional data object of class mfd with the functional residuals of the function-on-function linear regression model on the original scale, i.e. they are the difference between `mfdobj_y` and `fitted.values`,
- `residuals`: a multivariate functional data object of class mfd with the functional residuals of the function-on-function linear regression model, standardized or studentized depending on the argument `type_residuals`,
- `type_residuals`: the same as the provided argument,
- `pca_x`: an object of class `pca_mfd` obtained by doing MFPCA on the functional covariates,
- `pca_y`: an object of class `pca_mfd` obtained by doing MFPCA on the functional response,
- `pca_res`: an object of class `pca_mfd` obtained by doing MFPCA on the functional residuals,
- `components_x`: a vector of integers with the components selected in the `pca_x` model,
- `components_y`: a vector of integers with the components selected in the `pca_y` model,
- `components_res`: a vector of integers with the components selected in the `pca_res` model,
- `y_standardized`: the standardized functional response obtained doing `scale_mfd(mfdobj_y)`,
- `tot_variance_explained_x`: the same as the provided argument
- `tot_variance_explained_y`: the same as the provided argument
- `tot_variance_explained_res`: the same as the provided argument
- `get_studentized_residuals`: a function that allows to calculate studentized residuals on new data, given the estimated function-on-function linear regression model.

References

Centofanti F, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2021) Functional Regression Control Chart. *Technometrics*, 63(3), 281–294. doi:10.1080/00401706.2020.1753581

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj <- get_mfd_list(air, lambda = 1e-2)
mfdobj_y <- mfdobj[, "NO2"]
mfdobj_x <- mfdobj[, fun_covariates]
mod <- fof_pc(mfdobj_y, mfdobj_x)
```

`fof_pc_real_time` *Get a list of function-on-function linear regression models estimated on functional data each evolving up to an intermediate domain point.*

Description

This function produces a list of objects, each of them contains the result of applying `fof_pc` to a functional response variable and multivariate functional covariates evolved up to an intermediate domain point.

Usage

```
fof_pc_real_time(
  mfdobj_y_list,
  mfdobj_x_list,
  tot_variance_explained_x = 0.95,
  tot_variance_explained_y = 0.95,
  tot_variance_explained_res = 0.95,
  components_x = NULL,
  components_y = NULL,
  type_residuals = "standard",
  ncores = 1
)
```

Arguments

`mfdobj_y_list`	A list created using `get_mfd_df_real_time` or `get_mfd_list_real_time`, denoting a list of functional data objects, each evolving up to an intermediate domain point, with observations of the functional response variable.
`mfdobj_x_list`	A list created using `get_mfd_df_real_time` or `get_mfd_list_real_time`, denoting a list of functional data objects, each evolving up to an intermediate domain point, with observations of the multivariate functional covariates.
`tot_variance_explained_x`	See `fof_pc`.
`tot_variance_explained_y`	See `fof_pc`.
`tot_variance_explained_res`	See `fof_pc`.
`components_x`	See `fof_pc`.
`components_y`	See `fof_pc`.
`type_residuals`	See `fof_pc`.
`ncores`	If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of lists each produced by fof_pc, corresponding to a given instant.

See Also

fof_pc, get_mfd_df_real_time, get_mfd_list_real_time

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
mfdobj_y_list <- get_mfd_list_real_time(air["NO2"],
                                        n_basis = 15,
                                        lambda = 1e-2,
                                        k_seq = c(0.5, 0.75, 1))
mfdobj_x_list <- get_mfd_list_real_time(air[c("CO", "temperature")],
                                        n_basis = 15,
                                        lambda = 1e-2,
                                        k_seq = c(0.5, 0.75, 1))
mod_list <- fof_pc_real_time(mfdobj_y_list, mfdobj_x_list)
```

FRTM_PhaseI *Phase I of the FRTM method.*

Description

This function implements the design phase (Phase I) of FRTM method.

Usage

```
FRTM_PhaseI(
  data_tra,
  data_tun = NULL,
  alpha = 0.05,
  n_basis_xall = 30,
  control.FDTW = list(),
  control.mFPCA = list(),
  control.rtr = list(),
  ncores = 1,
  print = TRUE
)
```

Arguments

data_tra	A list containing the following arguments: x_err a list containing the discrete observations for each curve of the training set; grid_i a list of vector of time points where the curves of the training set are sampled.
data_tun	A list containing the following arguments: grid_i a list containing the discrete observations for each curve of the tuning set; grid_i a list of vector of time points where the curves of the tuning set are sampled. If NULL, the tuning set is not used.
alpha	Overall type I error probability to obtain the control chart limits.
n_basis_xall	Number of basis to obtain the functional observation via the spline smoothing approach based on cubic B-splines and a roughness penalty on the second derivative.
control.FDTW	A list of control parameters for the open-end/open-begin functional dynamic time warping to replace defaults returned by par.FDTW. Values not set assume default values.
control.mFPCA	A list of control parameters for the mixed functional principal component analysis to replace defaults returned by par.mFPCA. Values not set assume default values.
control.rtr	A list of control parameters for the real-time registration step to replace defaults returned by par.rtr. Values not set assume default values.
ncores	If ncores>1, then parallel computing is used, with ncores cores. Default is 1.
print	If TRUE, some information are printed. Default is TRUE.

Value

A list containing the following arguments:

T2_fd List of T^2 functions for each observation in the tuning set.

SPE_fd List of SPE functions for each observation in the tuning set.

CL_T2 Control limit of the Hotelling's T^2 control chart.

CL_SPE Control limit of the SPE control chart.

template_fd Template function used in the registration.

der_template_fd First derivative of the template function.

u_fd Upper extreme of the band constraint.

l_fd Lower extreme of the band constraint.

x_list_tun List, for each observation in the tuning set, of partial registered functions.

h_list_tun List, for each observation in the tuning set, of partial warping functions.

x_list List, for each observation in the training set, of partial registered functions.

h_list List, for each observation in the training set, of partial warping functions.

x_err A list containing the discrete observations for each curve of the training set.

grid_i A list of vector of time points where the curves of the training set are sampled.

x_list_smooth Smooth curves from the training set.

`lambda` Lambda identified through the average curve distance to obtain the OEB-FDTW solution.

`par_reg` Additional parameters to be used in the monitoring phase (Phase II).

References

Centofanti, F., A. Lepore, M. Kulahci, and M. P. Spooner (2024). Real-time monitoring of functional data. *Journal of Quality Technology*, 57(2):135–152, doi:https://doi.org/10.1080/00224065.2024.2430978.

See Also

FRTM_PhaseI

Examples

```
library(funcharts)
data <- simulate_data_FRTM(n_obs = 20)

data_oc <-
  simulate_data_FRTM(
    n_obs = 2,
    scenario = "1",
    shift = "OC_h",
    severity = 0.5
  )

lambda <- 10 ^ -5
max_x <- max(unlist(data$grid_i))
seq_t_tot <- seq(0, 1, length.out = 30)[-1]
seq_x <- seq(0.1, max_x, length.out = 10)



  mod_phaseI_FRTM <- FRTM_PhaseI(
    data_tra =  data,
    control.FDTW = list(
      M = 30,
      N = 30,
      lambda = lambda,
      seq_t = seq_t_tot,
      iter_tem = 1,
      iter = 1
    ),
    control.rtr = list(seq_x = seq_x)
  )
  mod_phaseII_FRTM <- FRTM_PhaseII(data_oc = data_oc , mod_phaseI = mod_phaseI_FRTM)

  plot(mod_phaseI_FRTM)
  plot(mod_phaseII_FRTM)
```

FRTM_PhaseII *Phase II of the FRTM method.*

Description

This function implements the monitoring phase (Phase II) of FRTM method.

Usage

```
FRTM_PhaseII(data_oc, mod_phaseI, ncores = 1)
```

Arguments

data_oc	A list containing the following arguments: x_err a list containing the discrete observations for each curve to be monitored; grid_i a list of vector of time points where the curves to be monitored are sampled.
mod_phaseI	An object of class mod_phaseI_FRTM obtained as output of the function FRTM_PhaseI.
ncores	If ncores>1, then parallel computing is used, with ncores cores. Default is 1.

Value

A list containing the following arguments:

T2_fd List of T^2 functions for each observation.

SPE_fd List of SPE functions for each observation.

CL_T2 Control limit of the Hotelling's T^2 control chart.

CL_SPE Control limit of the SPE control chart.

x_err A list containing the discrete observations for each curve.

grid_i A list of vector of time points where the curves are sampled.

x_list_smooth Smooth curves.

mod_phaseI An object of class mod_phaseI_FRTM obtained as output of the function FRTM_PhaseI.

References

Centofanti, F., A. Lepore, M. Kulahci, and M. P. Spooner (2024). Real-time monitoring of functional data. *Journal of Quality Technology*, 57(2):135–152, doi:https://doi.org/10.1080/00224065.2024.2430978.

Examples

```
library(funcharts)
data <- simulate_data_FRTM(n_obs = 20)

data_oc <-
  simulate_data_FRTM(
    n_obs = 2,
    scenario = "1",
```

```
    shift = "OC_h",
    severity = 0.5
  )

lambda <- 10 ^ -5
max_x <- max(unlist(data$grid_i))
seq_t_tot <- seq(0, 1, length.out = 30)[-1]
seq_x <- seq(0.1, max_x, length.out = 10)



  mod_phaseI_FRTM <- FRTM_PhaseI(
    data_tra =  data,
    control.FDTW = list(
      M = 30,
      N = 30,
      lambda = lambda,
      seq_t = seq_t_tot,
      iter_tem = 1,
      iter = 1
    ),
    control.rtr = list(seq_x = seq_x)
  )
  mod_phaseII_FRTM <- FRTM_PhaseII(data_oc = data_oc , mod_phaseI = mod_phaseI_FRTM)

  plot(mod_phaseI_FRTM)
  plot(mod_phaseII_FRTM)
```

`functional_filter` *Finds functional componentwise outliers*

Description

It finds functional componentwise outliers as described in Capezza et al. (2024).

Usage

```
functional_filter(
  mfdobj,
  method_pca = "ROBPCA",
  alpha = 0.95,
  fev = 0.999,
  delta = 0.1,
  alpha_binom = 0.99,
  bivariate = TRUE,
  max_proportion_componentwise = 0.5
)
```

Arguments

mfdobj	A multivariate functional data object of class mfd.
method_pca	The method used in rpca_mfd to perform robust multivariate functional principal component analysis (RoMFPCA). See rpca_mfd.
alpha	Probability value such that only values of functional distances greater than the alpha-quantile of the Chi-squared distribution, with a number of degrees of freedom equal to the number of principal components selected by fev, are considered to determine the proportion of flagged componentwise outliers. Default value is 0.95, as recommended by Agostinelli et al. (2015). See Capezza et al. (2024) for more details.
fev	Number between 0 and 1 denoting the fraction of variability that must be explained by the principal components to be selected to calculate functional distances after applying RoMFPCA on mfdobj. Default is 0.999.
delta	Number between 0 and 1 denoting the parameter of the Binomial distribution whose alpha_binom-quantile determines the threshold used in the bivariate filter. Given the i-th observation and the j-th functional variable, the number of pairs flagged as functional componentwise outliers in the i-th observation where the component (i, j) is involved is compared against this threshold to identify additional functional componentwise outliers to the ones found by the univariate filter. Default is 0.1, recommended as conservative choice by Leung et al. (2017). See Capezza et al. (2024) for more details.
alpha_binom	Probability value such that the alpha-quantile of the Binomial distribution is considered as threshold in the bivariate filter. See delta and Capezza et al. (2024) for more details. Default value is 0.99.
bivariate	If TRUE, both univariate and bivariate filters are applied. If FALSE, only the univariate filter is used. Default is TRUE.
max_proportion_componentwise	If the functional filter identifies a proportion of functional componentwise outliers larger than max_proportion_componentwise, for a given observation, then it is considered as a functional casewise outlier. Default value is 0.5.

Value

A list with two elements. The first element is an mfd object containing the original observation in the mfdobj input, but where the basis coefficients of the components identified as functional componentwise outliers are replaced by NA. The second element of the list is a list of numbers, with length equal to the number of functional variables in mfdobj. Each element of this list contains the observations of the flagged functional componentwise outliers for the corresponding functional variable.

References

Agostinelli, C., Leung, A., Yohai, V. J., and Zamar, R. H. (2015). Robust estimation of multivariate location and scatter in the presence of componentwise and casewise contamination. *Test*, 24(3):441–461.

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Leung, A., Yohai, V., and Zamar, R. (2017). Multivariate location and scatter matrix estimation under componentwise and casewise contamination. *Computational Statistics & Data Analysis*, 111:59–76.

Examples

```
library(funcharts)
mfdobj <- get_mfd_list(air, grid = 1:24, n_basis = 13, lambda = 1e-2)
plot_mfd(mfdobj)
out <- functional_filter(mfdobj, bivariate = FALSE)
```

`get_mfd_array` *Get Multivariate Functional Data from a three-dimensional array*

Description

Get Multivariate Functional Data from a three-dimensional array

Usage

```
get_mfd_array(
  data_array,
  grid = NULL,
  n_basis = 30,
  n_order = 4,
  basisobj = NULL,
  Lfdobj = 2,
  lambda = NULL,
  lambda_grid = 10^seq(-10, 1, length.out = 10),
  ncores = 1
)
```

Arguments

`data_array`	A three-dimensional array. The first dimension corresponds to argument values, the second to replications, and the third to variables within replications.
`grid`	See `get_mfd_list`.
`n_basis`	See `get_mfd_list`.
`n_order`	#' See `get_mfd_list`.
`basisobj`	#' See `get_mfd_list`.
`Lfdobj`	#' See `get_mfd_list`.
`lambda`	See `get_mfd_list`.
`lambda_grid`	See `get_mfd_list`.
`ncores`	Deprecated. See `get_mfd_list`.

Value

An object of class `mfd`. See also `?mfd` for additional details on the multivariate functional data class.

See Also

`get_mfd_list`, `get_mfd_df`

Examples

```
library(funcharts)
library(fda)
data("CanadianWeather")
mfdobj <- get_mfd_array(CanadianWeather$dailyAv[, 1:10, ],
                        lambda = 1e-5)
plot_mfd(mfdobj)
```

---

get_mfd_array_real_time

Get a list of functional data objects each evolving up to an intermediate domain point.

Description

This function produces a list functional data objects, each evolving up to an intermediate domain point, that can be used to estimate models that allow real-time predictions of incomplete functions, from the current functional domain up to the end of the observation, and to build control charts for real-time monitoring.

It calls the function `get_mfd_array` for each domain point.

Usage

```
get_mfd_array_real_time(
  data_array,
  grid = NULL,
  n_basis = 30,
  n_order = 4,
  basisobj = NULL,
  Lfdobj = 2,
  lambda = NULL,
  lambda_grid = 10^seq(-10, 1, length.out = 10),
  k_seq = seq(from = 0.25, to = 1, length.out = 10),
  ncores = 1
)
```

Arguments

data_array	See get_mfd_array.
grid	See get_mfd_array.
n_basis	See get_mfd_array.
n_order	See get_mfd_array.
basisobj	See get_mfd_array.
Lfdobj	See get_mfd_array.
lambda	See get_mfd_array.
lambda_grid	See get_mfd_array.
k_seq	A vector of values between 0 and 1, containing the domain points over which functional data are to be evaluated in real time. If the domain is the interval (a,b), for each instant k in the sequence, functions are evaluated in (a,k(b-a)).
ncores	If you want parallelization, give the number of cores/threads to be used when creating mfd objects separately for different instants.

Value

A list of mfd objects as produced by get_mfd_array.

See Also

get_mfd_array

Examples

```
library(funcharts)
library(fda)
data("CanadianWeather")
fdobj <- get_mfd_array_real_time(CanadianWeather$dailyAv[, 1:5, 1:2],
                                 lambda = 1e-2)
```

---

get_mfd_df *Get Multivariate Functional Data from a data frame*

Description

Get Multivariate Functional Data from a data frame

Usage

```
get_mfd_df(
  dt,
  domain,
  arg,
  id,
  variables,
  n_basis = 30,
  n_order = 4,
  basisobj = NULL,
  Lfdobj = 2,
  lambda = NULL,
  lambda_grid = 10^seq(-10, 1, length.out = 10),
  ncores = 1
)
```

Arguments

dt	A data.frame containing the discrete data. For each functional variable, a single column, whose name is provided in the argument variables, contains discrete values of that variable for all functional observation. The column indicated by the argument id denotes which is the functional observation in each row. The column indicated by the argument arg gives the argument value at which the discrete values of the functional variables are observed for each row.
domain	A numeric vector of length 2 defining the interval over which the functional data object can be evaluated.
arg	A character variable, which is the name of the column of the data frame dt giving the argument values at which the functional variables are evaluated for each row.
id	A character variable indicating which is the functional observation in each row.
variables	A vector of characters of the column names of the data frame dt indicating the functional variables.
n_basis	An integer variable specifying the number of basis functions; default value is 30. See details on basis functions.
n_order	An integer specifying the order of b-splines, which is one higher than their degree. The default of 4 gives cubic splines.
basisobj	An object of class basisfd defining the basis function expansion. Default is NULL, which means that a basisfd object is created by doing create.bspline.basis(rangeval = domain,nbasis = n_basis, norder = n_order)
Lfdobj	An object of class Lfd defining a linear differential operator of order m. It is used to specify a roughness penalty through fdPar. Alternatively, a nonnegative integer specifying the order m can be given and is passed as Lfdobj argument to the function fdPar, which indicates that the derivative of order m is penalized. Default value is 2, which means that the integrated squared second derivative is penalized.

lambda	A non-negative real number. If you want to use a single specified smoothing parameter for all functional data objects in the dataset, this argument is passed to the function fda::fdPar. Default value is NULL, in this case the smoothing parameter is chosen by minimizing the generalized cross-validation (GCV) criterion over the grid of values given by the argument. See details on how smoothing parameters work.
lambda_grid	A vector of non-negative real numbers. If lambda is provided as a single number, this argument is ignored. If lambda is NULL, then this provides the grid of values over which the optimal smoothing parameter is searched. Default value is 10^seq(-10,1,l=20).
ncores	If you want parallelization, give the number of cores/threads to be used when doing GCV separately on all observations.

Details

Basis functions are created with fda::create.bspline.basis(domain, n_basis), i.e. B-spline basis functions of order 4 with equally spaced knots are used to create mfd objects.

The smoothing penalty lambda is provided as fda::fdPar(bs, 2, lambda), where bs is the basis object and 2 indicates that the integrated squared second derivative is penalized.

Rather than having a data frame with long format, i.e. with all functional observations in a single column for each functional variable, if all functional observations are observed on a common equally spaced grid, discrete data may be available in matrix form for each functional variable. In this case, see get_mfd_list.

Value

An object of class mfd. See also ?mfd for additional details on the multivariate functional data class.

See Also

get_mfd_list

Examples

```
library(funcharts)

x <- seq(1, 10, length = 25)
y11 <- cos(x)
y21 <- cos(2 * x)
y12 <- sin(x)
y22 <- sin(2 * x)
df <- data.frame(id = factor(rep(1:2, each = length(x))),
                 x = rep(x, times = 2),
                 y1 = c(y11, y21),
                 y2 = c(y12, y22))

mfdobj <- get_mfd_df(dt = df,
                     domain = c(1, 10),
                     arg = "x",
```

```
                    id = "id",
                    variables = c("y1", "y2"),
                    lambda = 1e-5)
```

`get_mfd_df_real_time` *Get a list of functional data objects each evolving up to an intermediate domain point.*

Description

This function produces a list functional data objects, each evolving up to an intermediate domain point, that can be used to estimate models that allow real-time predictions of incomplete functions, from the current functional domain up to the end of the observation, and to build control charts for real-time monitoring.

It calls the function `get_mfd_df` for each domain point.

Usage

```
get_mfd_df_real_time(
  dt,
  domain,
  arg,
  id,
  variables,
  n_basis = 30,
  n_order = 4,
  basisobj = NULL,
  Lfdobj = 2,
  lambda = NULL,
  lambda_grid = 10^seq(-10, 1, length.out = 10),
  k_seq = seq(from = 0.25, to = 1, length.out = 10),
  ncores = 1
)
```

Arguments

`dt`	See `get_mfd_df`.
`domain`	See `get_mfd_df`.
`arg`	See `get_mfd_df`.
`id`	See `get_mfd_df`.
`variables`	See `get_mfd_df`.
`n_basis`	See `get_mfd_df`.
`n_order`	See `get_mfd_df`.
`basisobj`	See `get_mfd_df`.

Lfdobj	See get_mfd_df.
lambda	See get_mfd_df.
lambda_grid	See get_mfd_df.
k_seq	A vector of values between 0 and 1, containing the domain points over which functional data are to be evaluated in real time. If the domain is the interval (a,b), for each instant k in the sequence, functions are evaluated in (a,k(b-a)).
ncores	If you want parallelization, give the number of cores/threads to be used when creating mfd objects separately for different instants.

Value

A list of mfd objects as produced by get_mfd_df, corresponding to a given instant.

See Also

get_mfd_df

Examples

```
library(funcharts)

x <- seq(1, 10, length = 25)
y11 <- cos(x)
y21 <- cos(2 * x)
y12 <- sin(x)
y22 <- sin(2 * x)
df <- data.frame(id = factor(rep(1:2, each = length(x))),
                 x = rep(x, times = 2),
                 y1 = c(y11, y21),
                 y2 = c(y12, y22))

mfdobj_list <- get_mfd_df_real_time(dt = df,
                                    domain = c(1, 10),
                                    arg = "x",
                                    id = "id",
                                    variables = c("y1", "y2"),
                                    lambda = 1e-2)
```

---

get_mfd_fd *Convert a* fd *object into a Multivariate Functional Data object.*

Description

Convert a fd object into a Multivariate Functional Data object.

Usage

```
get_mfd_fd(fdobj)
```

Arguments

fdobj	An object of class fd.

Value

An object of class mfd. See also ?mfd for additional details on the multivariate functional data class.

See Also

mfd

Examples

```
library(funcharts)
library(fda)
bs <- create.bspline.basis(nbasis = 10)
fdobj <- fd(coef = 1:10, basisobj = bs)
mfdobj <- get_mfd_fd(fdobj)
```

---

get_mfd_list *Get Multivariate Functional Data from a list of matrices*

Description

Get Multivariate Functional Data from a list of matrices

Usage

```
get_mfd_list(
  data_list,
  grid = NULL,
  n_basis = 30,
  n_order = 4,
  basisobj = NULL,
  Lfdobj = 2,
  lambda = NULL,
  lambda_grid = 10^seq(-10, 1, length.out = 10),
  ncores = 1
)
```

Arguments

data_list	A named list of matrices. Names of the elements in the list denote the functional variable names. Each matrix in the list corresponds to a functional variable. All matrices must have the same dimension, where the number of rows corresponds to replications, while the number of columns corresponds to the argument values at which functions are evaluated.

grid	A numeric vector, containing the argument values at which functions are evaluated. Its length must be equal to the number of columns in each matrix in data_list. Default is NULL, in this case a vector equally spaced numbers between 0 and 1 is created, with as many numbers as the number of columns in each matrix in data_list.
n_basis	An integer variable specifying the number of basis functions; default value is 30. See details on basis functions.
n_order	An integer specifying the order of B-splines, which is one higher than their degree. The default of 4 gives cubic splines.
basisobj	An object of class basisfd defining the B-spline basis function expansion. Default is NULL, which means that a basisfd object is created by doing create.bspline.basis(rangeval = domain,nbasis = n_basis, norder = n_order)
Lfdobj	An object of class Lfd defining a linear differential operator of order m. It is used to specify a roughness penalty through fdPar. Alternatively, a nonnegative integer specifying the order m can be given and is passed as Lfdobj argument to the function fdPar, which indicates that the derivative of order m is penalized. Default value is 2, which means that the integrated squared second derivative is penalized.
lambda	A non-negative real number. If you want to use a single specified smoothing parameter for all functional data objects in the dataset, this argument is passed to the function fda::fdPar. Default value is NULL, in this case the smoothing parameter is chosen by minimizing the generalized cross-validation (GCV) criterion over the grid of values given by the argument. See details on how smoothing parameters work.
lambda_grid	A vector of non-negative real numbers. If lambda is provided as a single number, this argument is ignored. If lambda is NULL, then this provides the grid of values over which the optimal smoothing parameter is searched. Default value is 10^seq(-10,1,l=20).
ncores	Deprecated.

Details

Basis functions are created with fda::create.bspline.basis(domain, n_basis), i.e. B-spline basis functions of order 4 with equally spaced knots are used to create mfd objects.

The smoothing penalty lambda is provided as fda::fdPar(bs, 2, lambda), where bs is the basis object and 2 indicates that the integrated squared second derivative is penalized.

Rather than having a list of matrices, you may have a data frame with long format, i.e. with all functional observations in a single column for each functional variable. In this case, see get_mfd_df.

Value

An object of class mfd. See also mfd for additional details on the multivariate functional data class.

See Also

mfd, get_mfd_list, get_mfd_array

Examples

```
library(funcharts)
data("air")
# Only take first 5 multivariate functional observations
# and only two variables from air
air_small <- lapply(air[c("NO2", "CO")], function(x) x[1:5, ])
mfdobj <- get_mfd_list(data_list = air_small)
```

`get_mfd_list_real_time` *Get a list of functional data objects each evolving up to an intermediate domain point.*

Description

This function produces a list functional data objects, each evolving up to an intermediate domain point, that can be used to estimate models that allow real-time predictions of incomplete functions, from the current functional domain up to the end of the observation, and to build control charts for real-time monitoring.

It calls the function `get_mfd_list` for each domain point.

Usage

```
get_mfd_list_real_time(
  data_list,
  grid = NULL,
  n_basis = 30,
  n_order = 4,
  basisobj = NULL,
  Lfdobj = 2,
  lambda = NULL,
  lambda_grid = 10^seq(-10, 1, length.out = 10),
  k_seq = seq(from = 0.2, to = 1, by = 0.1),
  ncores = 1
)
```

Arguments

`data_list`	See `get_mfd_list`.
`grid`	See `get_mfd_list`.
`n_basis`	See `get_mfd_list`.
`n_order`	See `get_mfd_list`.
`basisobj`	See `get_mfd_list`.
`Lfdobj`	See `get_mfd_list`.

lambda	See get_mfd_list.
lambda_grid	See get_mfd_df.
k_seq	A vector of values between 0 and 1, containing the domain points over which functional data are to be evaluated in real time. If the domain is the interval (a,b), for each instant k in the sequence, functions are evaluated in (a,a+k(b-a)).
ncores	If you want parallelization, give the number of cores/threads to be used when creating mfd objects separately for different instants.

Value

A list of mfd objects as produced by get_mfd_list.

See Also

get_mfd_list

Examples

```
library(funcharts)
data("air")
# Only take first 5 multivariate functional observations from air
air_small <- lapply(air, function(x) x[1:5, ])
# Consider only 3 domain points: 0.5, 0.75, 1
mfdobj <- get_mfd_list_real_time(data_list = air_small,
                                 lambda = 1e-2,
                                 k_seq = c(0.5, 0.75, 1))
```

---

get_ooc *Get out of control observations from control charts*

Description

Get out of control observations from control charts

Usage

```
get_ooc(cclist)
```

Arguments

cclist	A data.frame produced by control_charts_pca, control_charts_sof_pc, regr_cc_fof, or regr_cc_sof.

Value

A data.frame with the same number of rows as cclist, and the same number of columns apart from the columns indicating control chart limits. Each value is TRUE if the corresponding observation is in control and FALSE otherwise.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[201:300, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:60]
y_tuning <- y[61:90]
y2 <- y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod <- sof_pc(y1, mfdobj_x1)
cclist <- regr_cc_sof(object = mod,
                      y_new = y2,
                      mfdobj_x_new = mfdobj_x2,
                      y_tuning = y_tuning,
                      mfdobj_x_tuning = mfdobj_x_tuning,
                      include_covariates = TRUE)
get_ooc(cclist)
```

---

get_outliers_mfd *Get outliers from multivariate functional data*

Description

Get outliers from multivariate functional data using the functional boxplot with the modified band depth of Sun et al. (2011, 2012). This function relies on the `fbplot` function of the `roahd` package.

Usage

```
get_outliers_mfd(mfdobj)
```

Arguments

`mfdobj`	A multivariate functional data object of class mfd

Value

A numeric vector with the indexes of the functional observations signaled as outliers.

References

- Sun, Y., & Genton, M. G. (2011). Functional boxplots. *Journal of Computational and Graphical Statistics*, 20(2), 316-334.
- Sun, Y., & Genton, M. G. (2012). Adjusted functional boxplots for spatio-temporal data visualization and outlier detection. *Environmetrics*, 23(1), 54-64.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:20, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates], lambda = 1e-2)
get_outliers_mfd(mfdobj_x)
```

---

get_sof_pc_outliers *Get possible outliers of a training data set of a scalar-on-function regression model.*

Description

Get possible outliers of a training data set of a scalar-on-function regression model. It sets the training data set also as tuning data set for the calculation of control chart limits, and as phase II data set to compare monitoring statistics against the limits and identify possible outliers. This is only an empirical approach. It is advised to use methods appropriately designed for phase I monitoring to identify outliers.

Usage

```
get_sof_pc_outliers(y, mfdobj)
```

Arguments

y	A numeric vector containing the observations of the scalar response variable.
mfdobj	A multivariate functional data object of class mfd denoting the functional covariates.

Value

A character vector with the ids of functional observations signaled as possibly anomalous.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates], lambda = 1e-2)
y <- rowMeans(air$NO2)
get_sof_pc_outliers(y, mfdobj_x)
```

---

inprod_mfd *Inner products of functional data contained in* mfd *objects.*

Description

Inner products of functional data contained in mfd objects.

Usage

```
inprod_mfd(mfdobj1, mfdobj2 = NULL)
```

Arguments

mfdobj1	A multivariate functional data object of class mfd.
mfdobj2	A multivariate functional data object of class mfd. It must have the same functional variables as mfdobj1. If NULL, it is equal to mfdobj1.

Details

Note that *L^2* inner products are not calculated for couples of functional data from different functional variables. This function is needed to calculate the inner product in the product Hilbert space in the case of multivariate functional data, which for each observation is the sum of the *L^2* inner products obtained for each functional variable.

Value

a three-dimensional array of *L^2* inner products. The first dimension is the number of functions in argument mfdobj1, the second dimension is the same thing for argument mfdobj2, the third dimension is the number of functional variables. If you sum values over the third dimension, you get a matrix of inner products in the product Hilbert space of multivariate functional data.

Examples

```
library(funcharts)
set.seed(123)
mfdobj1 <- data_sim_mfd()
mfdobj2 <- data_sim_mfd()
inprod_mfd(mfdobj1)
inprod_mfd(mfdobj1, mfdobj2)
```

`inprod_mfd_diag` *Inner product of two multivariate functional data objects, for each observation*

Description

Inner product of two multivariate functional data objects, for each observation

Usage

```
inprod_mfd_diag(mfdobj1, mfdobj2 = NULL)
```

Arguments

`mfdobj1`	A multivariate functional data object of class `mfd`.
`mfdobj2`	A multivariate functional data object of class `mfd`, with the same number of functional variables and observations as `mfdobj1`. If NULL, then `mfdobj2=mfdobj1`. Default is NULL.

Value

It calculates the inner product of two multivariate functional data objects. The main function `inprod` of the package `fda` calculates inner products among all possible couples of observations. This means that, if `mfdobj1` has `n1` observations and `mfdobj2` has `n2` observations, then for each variable `n1 X n2` inner products are calculated. However, often one is interested only in calculating the `n` inner products between the `n` observations of `mfdobj1` and the corresponding `n` observations of `mfdobj2`. This function provides this "diagonal" inner products only, saving a lot of computation with respect to using `fda::inprod` and then extracting the diagonal elements. Note that the code of this function calls a modified version of `fda::inprod()`.

Examples

```
mfdobj <- data_sim_mfd()
inprod_mfd_diag(mfdobj)
```

is.mfd *Confirm Object has Class* mfd

Description

Check that an argument is a multivariate functional data object of class mfd.

Usage

```
is.mfd(mfdobj)
```

Arguments

mfdobj	An object to be checked.

Value

a logical value: TRUE if the class is correct, FALSE otherwise.

lines.mfd *Add curves to an existing multivariate functional data plot*

Description

Adds new curves from an mfd object to an existing plot created by plot.mfd. Each variable is added to its corresponding panel, using the same scales and layout as the original plot.

Usage

```
## S3 method for class 'mfd'
lines(x, ...)
```

Arguments

x	An mfd object.
...	Graphical arguments passed to plot.mfd with add=TRUE.

Value

Invisibly returns NULL.

See Also

plot.mfd, abline_mfd

lines_mfd *Add the plot of a new multivariate functional data object to an existing plot.*

Description

Add the plot of a new multivariate functional data object to an existing plot.

Usage

```
lines_mfd(
  plot_mfd_obj,
  mfdobj_new,
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = TRUE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  type_mfd = "mfd",
  y_lim_equal = FALSE,
  ...
)
```

Arguments

plot_mfd_obj	A plot produced by link{plot_mfd}
mfdobj_new	A new multivariate functional data object of class mfd to be plotted.
mapping	See plot_mfd.
data	See plot_mfd.
stat	See plot_mfd.
position	See plot_mfd.
na.rm	See plot_mfd.
orientation	See plot_mfd.
show.legend	See plot_mfd.
inherit.aes	See plot_mfd.
type_mfd	See plot_mfd.
y_lim_equal	See plot_mfd.
...	See plot_mfd.

Value

A plot of the multivariate functional data object added to the existing one.

Examples

```
library(funcharts)
library(ggplot2)
mfdobj1 <- data_sim_mfd()
mfdobj2 <- data_sim_mfd()
p <- plot_mfd(mfdobj1)
lines_mfd(p, mfdobj_new = mfdobj2)
```

---

mean.mfd *Mean Function for Multivariate Functional Data*

Description

Computes the mean function for an object of class `mfd`.

Usage

```
## S3 method for class 'mfd'
mean(x, ...)
```

Arguments

`x`	An object of class `mfd`, containing N observations of a p-dimensional multivariate functional variable.
`...`	Further arguments are ignored (required for S3 consistency).

Details

The method averages the coefficient array across the observation dimension, resulting in a new `mfd` object with one observation (the sample mean).

Value

An object of class `mfd` representing the mean function. The output contains a single observation, corresponding to the mean function for each variable.

See Also

`mfd`, `plot.mfd`

Examples

```
library(funcharts)
data(air)
mfdobj <- get_mfd_list(air)
mean_result <- mean(mfdobj)
plot(mean_result)
```

`mfd` *Define a Multivariate Functional Data Object*

Description

This is the constructor function for objects of the mfd class. It is a wrapper to `fda::fd`, but it forces the coef argument to be a three-dimensional array of coefficients even if the functional data is univariate. Moreover, it allows to include the original raw data from which you get the smooth functional data. Finally, it also includes the matrix of precomputed inner products of the basis functions, which can be useful to speed up computations when calculating inner products between functional observations

Usage

```
mfd(coef, basisobj, fdnames = NULL, raw = NULL, id_var = NULL, B = NULL)
```

Arguments

`coef`	A three-dimensional array of coefficients: • the first dimension corresponds to basis functions. • the second dimension corresponds to the number of multivariate functional observations. • the third dimension corresponds to variables.
`basisobj`	A functional basis object defining the basis, as provided to `fda::fd`, but there is no default.
`fdnames`	A list of length 3, each member being a string vector containing labels for the levels of the corresponding dimension of the discrete data. The first dimension is for a single character indicating the argument values, i.e. the variable on the functional domain. The second is for replications, i.e. it denotes the functional observations. The third is for functional variables' names.
`raw`	A data frame containing the original discrete data. Default is NULL, however, if provided, it must contain: a column (indicated by the `id_var` argument) denoting the functional observations, which must correspond to values in `fdnames[[2]]`,

	a column named as `fdnames[[1]]`, returning the argument values of each function
	as many columns as the functional variables, named as in `fdnames[[3]]`, containing the discrete functional values for each variable.
`id_var`	A single character value indicating the column in the `raw` argument containing the functional observations (as in `fdnames[[2]]`), default is NULL.
`B`	A matrix with the inner products of the basis functions. If NULL, it is calculated from the basis object provided. Default is NULL.

Details

To check that an object is of this class, use function is.mfd.

Value

A multivariate functional data object (i.e., having class `mfd`), which is a list with components named `coefs`, `basis`, and `fdnames`, as for class `fd`, with possibly in addition the components `raw` and `id_var`.

References

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis*, 2nd ed., Springer, New York.

Ramsay, James O., and Silverman, Bernard W. (2002), *Applied Functional Data Analysis*, Springer, New York.

Examples

```
library(funcharts)
library(fda)
set.seed(0)
nobs <- 5
nbasis <- 10
nvar <- 2
coef <- array(rnorm(nobs * nbasis * nvar), dim = c(nbasis, nobs, nvar))
bs <- create.bspline.basis(rangeval = c(0, 1), nbasis = nbasis)
mfdobj <- mfd(coef = coef, basisobj = bs)
plot_mfd(mfdobj)
```

---

mFPCA *Mixed Functional Principal Component Analysis (mFPCA)*

Description

This function implements the mFPCA.

Usage

```
mFPCA(x_fd, h_fd = NULL, k_weights = "equal", ncom = "ptv", par_ncom = 0.9)
```

Arguments

x_fd	An object of class fd corresponding to the registered functions.
h_fd	An object of class fd corresponding to the warping functions.
k_weights	The vector of the four constants in the inner product computation. If "equal", the choice of Centofanti et al. (2024) is used.
ncom	It is the way to select the number of principal components. If "ptv", it is selected considering the percentage of the total variability explained. If "kaiserrule", it is selected considering the Kaiser rule. The number of principal components may be indicated directly as an integer as well.
par_ncom	If ncom="ptv", the threshold for the percentage of the total variability explained. If ncom="kaiserrule", the threshold for the Kaiser rule. Otherwise, this parameter is not considered.

Value

A list containing the following arguments:

eigfun_fd A List of functions corresponding to the functional part of the principal components.

eigvect_sc A matrix corresponding to the scalar part of the principal components.

scores Scores corresponding to x_fd and h_fd.

values Eigenvalues corresponding to the selected principal components.

varprop Variance proportion explained by each principal component.

k_weights The vector of the four constants in the inner product computation.

x_fd_list A List of two elements: the list of the registered functions and the list of the centered log-ratio transformation of the first derivatives of the normalized warping functions.

sc_mat Two column matrix corresponding to the scalar part of the observations used.

mean_fd_list Mean functions of the functional part.

mean_sc_mat Means of the scalar part.

sd_fd_list The standard deviation of the functional part.

sd_sc_mat Standard deviations of the scalar part.

h_fd An object of class fd corresponding to the warping functions.

x_fd An object of class fd corresponding to the registered functions. ind_var Additional parameter used in FRTM_PhaseI.

References

Centofanti, F., A. Lepore, M. Kulahci, and M. P. Spooner (2024). Real-time monitoring of functional data. *Journal of Quality Technology*, 57(2):135–152, doi:https://doi.org/10.1080/00224065.2024.2430978.

Examples

```
library(funcharts)

data <- simulate_data_FRTM(n_obs = 100)
X <- sapply(1:100, function(ii)
  data$x_true[[ii]])
x_fd <-
  fda::smooth.basis(y = X,
                    argvals =  data$grid,
                    fda::create.bspline.basis(c(0, 1), 30))$fd
H <- sapply(1:100, function(ii)
  data$h[[ii]])
h_fd <-
  fda::smooth.basis(y = H,
                    argvals =  data$grid,
                    fda::create.bspline.basis(c(0, 1), 30))$fd
mod_mFPCA <- mFPCA(x_fd, h_fd, ncom = "ptv", par_ncom = 0.95)
plot(mod_mFPCA)
```

`minus_mfd` *Subtract multivariate functional data (and unary negation)*

Description

Subtracts two objects of class `mfd` (elementwise on their coefficient arrays). The same basis, variable-count, and observation-replication rules as in `plus_mfd` apply. If `mfdobj2` is missing, returns the unary negation of `mfdobj1`.

Usage

```
minus_mfd(mfdobj1, mfdobj2)

## S3 method for class 'mfd'
mfdobj1 - mfdobj2
```

Arguments

`mfdobj1`, `mfdobj2`
: Objects of class `mfd`. If `mfdobj2` is missing, unary minus is applied to `mfdobj1`.

Value

An object of class `mfd` with coefficients equal to the (possibly replicated) difference `mfdobj1 - mfdobj2`, or the negation of `mfdobj1` for unary minus.

See Also

`plus_mfd`, `nobs`, `nbasis`, `nvar`, `mfd`

Examples

```
# mfdobj_a - mfdobj_b
# minus_mfd(mfdobj_a, mfdobj_b)
# Unary minus:
# -mfdobj_a
```

`mixregfit_multivariate`

Performs the estimation of gaussian mixtures of regression models and gaussian mixture models. Used in FMRCC_PhaseI.

Description

Performs the estimation of gaussian mixtures of regression models and gaussian mixture models. Used in FMRCC_PhaseI.

Usage

```
mixregfit_multivariate(
  y,
  x,
  k,
  init_met = "random",
  intercept = FALSE,
  eps = 1e-06,
  max_iter = 500,
  model_Sigma
)
```

Arguments

`y`	a matrix with the scores of the response variable
`x`	a matrix with the scores of the covariates
`k`	the number of groups to consider in the model estimation
`init_met`	the method to initialize the model, it can be 'kmeans' or 'random'. Default is 'kmeans'
`intercept`	logical, if TRUE the model includes an intercept. Default is TRUE
`eps`	the convergence criterion. Default is 1e-6
`max_iter`	the maximum number of iterations. Default is 500
`model_Sigma`	the parametrization of the covariance. It can be 'VVV', 'EEE', 'VII' or 'EII', with no default

Value

a list with the estimated parameters of the model

nbasis *Number of basis functions*

Description

Generic function to extract the number of basis functions from an object.

Usage

```
nbasis(object, ...)
```

Arguments

object	An object from which to extract the number of basis functions.
...	Further arguments passed to methods (not used).

nobs.mfd *Number of observations in a multivariate functional data object*

Description

Number of observations in a multivariate functional data object

Usage

```
## S3 method for class 'mfd'
nobs(object, ...)
```

Arguments

object	An object of class mfd.
...	Further arguments passed to methods (not used).

Value

An integer: the number of observations.

Examples

```
# nobs(mfdobj)
```

norm.mfd *Norm of Multivariate Functional Data*

Description

Norm of multivariate functional data contained in a `mfd` object.

Usage

```
norm.mfd(mfdobj)
```

Arguments

`mfdobj`	A multivariate functional data object of class `mfd`.

Value

A vector of length equal to the number of replications in `mfdobj`, containing the norm of each multivariate functional observation in the product Hilbert space, i.e. the sum of *L^2* norms for each functional variable.

Examples

```
library(funcharts)
mfdobj <- data_sim_mfd()
norm.mfd(mfdobj)
```

nvar *Number of variables*

Description

Generic function to extract the number of variables from an object.

Usage

```
nvar(object, ...)
```

Arguments

`object`	An object from which to extract the number of variables.
`...`	Further arguments passed to methods (not used).

OEBFDTW *Open-end/open-begin Functional Dynamic Time Warping (OEB-FDTW)*

Description

This function implements the OEB-FDTW.

Usage

```
OEBFDTW(
  x_fd,
  template_fd,
  der_x_fd,
  der_template_fd,
  alpha_vec = c(0, 0.5, 1),
  fit_c = FALSE,
  N = 100,
  M = 50,
  smin = 0.01,
  smax = 1000,
  lambda = 10^-5,
  eta = 0.5,
  iter = 3,
  delta_xs = 0,
  delta_xe = 0,
  delta_ys = 0,
  delta_ye = 0,
  der_0 = NULL,
  seq_t = NULL,
  get_fd = "no",
  n_basis_x = NULL
)
```

Arguments

x_fd	An object of class fd corresponding to the misaligned function.
template_fd	An object of class fd corresponding to the template function.
der_x_fd	An object of class fd corresponding to the first derivative of x_fd.
der_template_fd	An object of class fd corresponding to the first derivative of template_fd.
alpha_vec	Grid of values to find the optimal value of α_i.
fit_c	If TRUE, the value of the objective function without the penalization evaluated at the solution is returned.
N	The number N_t of evenly spaced values along the template domain $\mathcal{D}_Y$.

M	The number M_x of evenly spaced values along the functional observation domain $\mathcal{D}_{X_i}$.
smin	The minimum values allowed for the first derivative of the warping function h_i. If NULL, in FRTM_PhaseI, it is set as 0.001 multiplied by the ratio between the size of the monitoring and template domains.
smax	The maximum values allowed for the first derivative of the warping function h_i. If NULL, in FRTM_PhaseI, it is set as 100 multiplied by the ratio between the size of the monitoring and template domains.
lambda	The smoothing parameter λ_i.
eta	Fraction η for updating the constraint bounds to reduce the error associated to the discretization (Deriso and Boyd, 2022).
iter	Number of iteration in the iterative refinement to reduce the error associated to the discretization (Deriso and Boyd, 2022).
delta_xs	Maximum allowed misalignment at the beginning of the process along the misaligned function domain.
delta_xe	Maximum allowed misalignment at the end of the process along the misaligned function domain.
delta_ys	Maximum allowed misalignment at the beginning of the process along the template domain.
delta_ye	Maximum allowed misalignment at the end of the process along the template domain.
der_0	The target values towards which shrinking the warping function slope. If NULL, it is equal to the ratio between the size of the domain of x_fd and the size of the domain of template_fd.
seq_t	Discretized sequence in the template domain $\mathcal{D}_Y$. If NULL, an equally spaced grid of length N in the template domain is used.
get_fd	If "x_reg", the registered function as a class fd object is returned. If "no", the registered function as a class fd object is not returned.
n_basis_x	Number of basis to obtain the registered function. If NULL, it is set as 0.5 the length of the optimal path.

Value

A list containing the following arguments:

mod that is a list composed by

- h_fd: The estimated warping function.
- x_reg: The registered function.
- h_list: A list containing the discretized warping function for each iteration of the iterative refinement.
- min_cost: Optimal value of the objective function.
- lambda: The smoothing parameter λ.
- alpha: Optimal value of the parameter α_i.

`obj` Values of the objective function for each value in `alpha_vec`.

`fit` Values of the objective function without the penalization for each value in `alpha_vec`.

`obj_opt` Optimal value of the objective function.

`fit_opt` Optimal value of the objective function without the penalization.

References

Centofanti, F., A. Lepore, M. Kulahci, and M. P. Spooner (2024). Real-time monitoring of functional data. *Journal of Quality Technology*, 57(2):135–152, doi:https://doi.org/10.1080/00224065.2024.2430978.

Deriso, D. and S. Boyd (2022). A general optimization framework for dynamic time warping. *Optimization and Engineering, 1-22*.

Examples

```
set.seed(1)
data <- simulate_data_FRTM(n_obs = 100)

dom <- c(0, 1)
basis_x <- fda::create.bspline.basis(c(0, 1), nbasis = 30)
x_fd <-
 fda::smooth.basis(data$grid_i[[1]] / max(data$grid_i[[1]]), data$x_err[[1]], basis_x)$fd
template_fd <-
  fda::smooth.basis(data$grid_template, data$template, basis_x)$fd
der_x_fd <- fda::deriv.fd(x_fd, 1)
der_template_fd <- fda::deriv.fd(template_fd, 1)

mod <-
  OEBFDTW(x_fd, template_fd, der_x_fd , der_template_fd, get_fd = "x_reg")
```

`par.FDTW` *Setting open-end/open-begin functional dynamic time warping (OEB-FDTW) defaults*

Description

This is an internal function of package `FRTM` which allows controlling the parameters to implement the OEB-FDTW in the FRTM method.

Usage

```
par.FDTW(
  N = 100,
  M = 50,
  smin = NULL,
  smax = NULL,
  alpha_vec = c(0, 0.5, 1),
  frac_oeob = 0.1,
```

```
  eta = 0.5,
  iter = 3,
  template = "Procrustes",
  grid_tem = NULL,
  index_tem = NULL,
  iter_tem = 2,
  lambda = c(0, 10^seq(-8, -2, by = 0.25), 10^5),
  threshold = 0.01,
  seq_t = seq(0.01, 1, length.out = 100)
)
```

Arguments

N	The number N_t of evenly spaced values along the template domain $\mathcal{D}_Y$.
M	The number M_x of evenly spaced values along the functional observation domain $\mathcal{D}_{X_i}$.
smin	The minimum values allowed for the first derivative of the warping function h_i. If NULL, in FRTM_PhaseI, it is set as 0.001 multiplied by the ratio between the size of the monitoring and template domains.
smax	The maximum values allowed for the first derivative of the warping function h_i. If NULL, in FRTM_PhaseI, it is set as 100 multiplied by the ratio between the size of the monitoring and template domains.
alpha_vec	Grid of values to find the optimal value of α_i.
frac_oeob	Fraction of $\mathcal{D}_Y$ and $\mathcal{D}_{X_i}$ to obtain $\delta_{t,s}$, $\delta_{t,e}$,$\delta_{x,s}$ and $\delta_{x,e}$.
eta	Fraction η for updating the constraint bounds to reduce the error associated to the discretization (Deriso and Boyd, 2022).
iter	Number of iteration in the iterative refinement to reduce the error associated to the discretization (Deriso and Boyd, 2022).
template	If "Procrustes", the Procrustes fitting process is used to select the template function. If numeric, the discrete observations of the template function.
grid_tem	If template is numeric, a vector of time points where the discrete observations of the template function are sampled.
index_tem	If NULL and template="Procrustes", the function in the training set, whose domain length is nearest the median domain length, is chosen as initial estimate of the template function. If an integer and template="Procrustes", the index_tem function in the training set is chosen as initial estimate of the template function. If template is numeric, this parameter is not used.
iter_tem	Number of iterations in the Procrustes fitting process.
lambda	Grid of smoothing parameters to evaluate the average curve distance (ACD).
threshold	The fraction δ of the difference between the maximum and the minimum distance in the selection of the smoothing parameter via the ACD.
seq_t	Discretized sequence in the template domain $\mathcal{D}_Y$.

References

Deriso, D. and S. Boyd (2022). A general optimization framework for dynamic time warping. *Optimization and Engineering, 1-22*.

See Also

FRTM_PhaseI

Examples

```
library(funcharts)
par.FDTW()
```

`par.mFPCA` *Setting mixed functional principal component analysis (mFPCA) defaults*

Description

This is an internal function of package `FRTM` which allows controlling the parameters used in the mFPCA in the FRTM method.

Usage

```
par.mFPCA(perc_pca = 0.9, perc_basis_x_pca = 1, perc_basis_h = 0.2)
```

Arguments

`perc_pca`	Percentage of the total variability used to select the number L of principal components.
`perc_basis_x_pca`	Multiplied by the maximum number of basis of the registered functions for each time point in $\mathcal{D}_Y$, it is the number of basis functions of the registered functions in the mFPCA.
`perc_basis_h`	Multiplied by the mean number of basis of the warping functions for each time point in $\mathcal{D}_Y$, it is the number of basis functions of the warping functions in the mFPCA.

See Also

FRTM_PhaseI

Examples

```
library(funcharts)
par.mFPCA()
```

`par.rtr` *Setting real-time registration step defaults*

Description

This is an internal function of package `FRTM` which allows controlling the parameters to implement real-time registration step in the FRTM method.

Usage

```
par.rtr(
  seq_x = NULL,
  delta_d = 0.05,
  delta_v = 0.03,
  delta_c = 0.04,
  Delta = 0.1,
  length_grid_window = 10,
  length_grid_owindow = 20,
  eval_seq_der = seq(0.02, 0.1, length.out = 10),
  perc_basis_x_reg = 0.3
)
```

Arguments

`seq_x`	Discretized sequence in the monitoring domain $\mathcal{D}_m$.
`delta_d`	The parameter δ_d in the adaptive band constraint calculation.
`delta_v`	The parameter δ_v in the adaptive band constraint calculation.
`delta_c`	The parameter δ_c in the adaptive band constraint calculation.
`Delta`	The parameter Δ in the adaptive band constraint calculation.
`length_grid_window`	Number of points to be explored in the interval of the band constraint for each point in $\mathcal{D}_m$ when the adaptive band constraint is considered.
`length_grid_owindow`	Number of points to be explored in the interval of the band constraint for each point in $\mathcal{D}_m$ when the original band constraint is considered.
`eval_seq_der`	If multiplied by the template domain range, the distances from the domain right boundaries over which are calculated the first derivative to mitigate the effects of possible estimation errors in the adaptive band constraint calculation.
`perc_basis_x_reg`	Multiplied by the number of observed discrete points, it is the number of basis functions used in the real-time registration step for each time point. This latter number cannot be grater than `n_basis_xall`.

See Also

`FRTM_PhaseI`

Examples

```
library(funcharts)
par.rtr()
```

---

pca_mfd *Multivariate functional principal components analysis*

Description

Multivariate functional principal components analysis (MFPCA) performed on an object of class mfd. It is a wrapper to fda::pca.fd, providing some additional arguments.

Usage

```
pca_mfd(mfdobj, scale = TRUE, nharm = 20)
```

Arguments

mfdobj	A multivariate functional data object of class mfd.
scale	If TRUE, it scales data before doing MFPCA using scale_mfd. Default is TRUE.
nharm	Number of multivariate functional principal components to be calculated. Default is 20.

Value

Modified pca.fd object, with multivariate functional principal component scores summed over variables (fda::pca.fd returns an array of scores when providing a multivariate functional data object). Moreover, the multivariate functional principal components given in harmonics are converted to the mfd class.

See Also

scale_mfd

Examples

```
library(funcharts)
mfdobj <- data_sim_mfd()
pca_obj <- pca_mfd(mfdobj)
plot_pca_mfd(pca_obj)
```

`pca_mfd_real_time` *Get a list of multivariate functional principal component analysis models estimated on functional data each evolving up to an intermediate domain point.*

Description

This function produces a list of objects, each of them contains the result of applying `pca_mfd` to a multivariate functional data object evolved up to an intermediate domain point.

Usage

```
pca_mfd_real_time(mfdobj_list, scale = TRUE, nharm = 20, ncores = 1)
```

Arguments

`mfdobj_list`	A list created using `get_mfd_df_real_time` or `get_mfd_list_real_time`, denoting a list of functional data objects, each evolving up to an intermediate domain point, with observations of the multivariate functional data.
`scale`	See `pca_mfd`.
`nharm`	See `pca_mfd`.
`ncores`	If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of lists each produced by `pca_mfd`, corresponding to a given instant.

See Also

`pca_mfd`

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
mfdobj_list <- get_mfd_list_real_time(air[c("CO", "temperature")],
                                      n_basis = 15,
                                      lambda = 1e-2,
                                      k_seq = seq(0.25, 1, length = 5))
mod_list <- pca_mfd_real_time(mfdobj_list)
```

plot.AMFCC_PhaseI *Plot the results of the Phase I and the Phase II of the AMFCC*

Description

This function provides plots of either the monitoring statistics or the contribution plot for a given observation.

Usage

```
## S3 method for class 'AMFCC_PhaseI'
plot(x, ...)

## S3 method for class 'AMFCC_PhaseII'
plot(x, ...)
```

Arguments

x	The output of either AMFCC_PhaseI or AMFCC_PhaseII.
...	Select the type of plot to produce either the contribution plot 'cont' or the monitoring plot 'mon'. Default is 'mon'. Select the combining_function to use for the monitoring plot either 'Fisher' or 'Tippett'. Default is 'Fisher'. Set the observation index ind_obs for which producing the contribution plot.

Value

No return value, called for side effects.

Examples

```
library(funcharts)
N <- 10
l_grid <- 10
p <- 2
grid <- seq(0, 1, l = l_grid)


Xall_tra <- funcharts::simulate_mfd(
  nobs = N,
  p = p,
  ngrid = l_grid,
  correlation_type_x = c("Bessel", "Gaussian")
)
X_tra <-
  data.frame(
    x = c(Xall_tra$X_list[[1]], Xall_tra$X_list[[2]]),
    timeindex = rep(rep(1:l_grid, each = (N)), p),
    curve = rep(1:(N), l_grid * p),
    var = rep(1:p, each = l_grid * N)
```

```
  )

Xall_II <- funcharts::simulate_mfd(
  nobs = N,
  p = p,
  ngrid = l_grid,
  shift_type_x = list("A", "B"),
  d_x = c(10, 10),
  correlation_type_x = c("Bessel", "Gaussian")
)

X_II <-
  data.frame(
    x = c(Xall_II$X_list[[1]], Xall_II$X_list[[2]]),
    timeindex = rep(rep(1:l_grid, each = (N)), p),
    curve = rep(1:(N), l_grid * p),
    var = rep(1:p, each = l_grid * N)
  )

# AMFCC ------------------------------------------------------------------
print("AMFCC")

mod_phaseI_AMFCC <- AMFCC_PhaseI(
  data_tra = X_tra,
  data_tun =
    NULL,
  grid = grid,
  ncores = 1
)

mod_phaseII_AMFCC <- AMFCC_PhaseII(data = X_II,
mod_Phase_I = mod_phaseI_AMFCC,
ncores = 1)

plot(mod_phaseII_AMFCC)
plot(mod_phaseII_AMFCC,type='cont',ind_obs=1)
```

`plot.FRTM_PhaseI` *Plot the results of the Phase I and the Phase II of the FRTM*

Description

This function provides plots of the Hotelling's T^2 and SPE control charts.

Usage

```
## S3 method for class 'FRTM_PhaseI'
plot(x, ...)
```

```
## S3 method for class 'FRTM_PhaseII'
plot(x, ...)
```

Arguments

x	The output of either FRTM_PhaseI or FRTM_PhaseII.
...	A variable ind_sel could be provided to select some observations from either the tuning or monitoring set.

Value

No return value, called for side effects.

Examples

```
library(funcharts)
data <- simulate_data_FRTM(n_obs = 20)

data_oc <-
  simulate_data_FRTM(
    n_obs = 2,
    scenario = "1",
    shift = "OC_h",
    severity = 0.5
  )

lambda <- 10 ^ -5
max_x <- max(unlist(data$grid_i))
seq_t_tot <- seq(0, 1, length.out = 30)[-1]
seq_x <- seq(0.1, max_x, length.out = 10)



  mod_phaseI_FRTM <- FRTM_PhaseI(
    data_tra =  data,
    control.FDTW = list(
      M = 30,
      N = 30,
      lambda = lambda,
      seq_t = seq_t_tot,
      iter_tem = 1,
      iter = 1
    ),
    control.rtr = list(seq_x = seq_x)
  )
  mod_phaseII_FRTM <- FRTM_PhaseII(data_oc = data_oc , mod_phaseI = mod_phaseI_FRTM)

  plot(mod_phaseI_FRTM)
  plot(mod_phaseII_FRTM)
```

plot.mfd *Plot multivariate functional data*

Description

Plot multivariate functional data

Usage

```
## S3 method for class 'mfd'
plot(x, y, add = FALSE, common_ylim = TRUE, ...)
```

Arguments

x	An mfd object.
y	Ignored.
add	Logical; if TRUE, add curves to an existing mfd plot (using matlines) instead of creating a new one.
common_ylim	Logical; if TRUE, all panels share the same y-limits, otherwise each panel adapts its own scale.
...	Graphical arguments passed to matplot (if add=FALSE) or to matlines (if add=TRUE).

plot.mFPCA *Plot the results of the Mixed Functional Principal Component Analysis (mFPCA)*

Description

This function provides plots of the principal components of the mFPCA.

Usage

```
## S3 method for class 'mFPCA'
plot(x, ...)
```

Arguments

x	The output of mFPCA.
...	A variable type could be provided that can assume two values. If type="single", the principal components are plotted separately. If type="all", the principal components are plotted together.

Value

No return value, called for side effects.

Examples

```
library(funcharts)

data <- simulate_data_FRTM(n_obs = 100)
X <- sapply(1:100, function(ii)
  data$x_true[[ii]])
x_fd <-
  fda::smooth.basis(y = X,
                    argvals =  data$grid,
                    fda::create.bspline.basis(c(0, 1), 30))$fd
H <- sapply(1:100, function(ii)
  data$h[[ii]])
h_fd <-
  fda::smooth.basis(y = H,
                    argvals =  data$grid,
                    fda::create.bspline.basis(c(0, 1), 30))$fd
mod_mFPCA <- mFPCA(x_fd, h_fd, ncom = "ptv", par_ncom = 0.95)
plot(mod_mFPCA)
```

`plot_bifd` *Plot a Bivariate Functional Data Object.*

Description

Plot an object of class `bifd` using `ggplot2` and `geom_tile`. The object must contain only one single functional replication.

Usage

```
plot_bifd(bifd_obj, type_plot = "raster", phi = 40, theta = 40)
```

Arguments

`bifd_obj`	A bivariate functional data object of class bifd, containing one single replication.
`type_plot`	a character value If "raster", it plots the bivariate functional data object as a raster image. If "contour", it produces a contour plot. If "perspective", it produces a perspective plot. Default value is "raster".
`phi`	If `type_plot=="perspective"`, it is the `phi` argument of the function `plot3D::persp3D`.
`theta`	If `type_plot=="perspective"`, it is the `theta` argument of the function `plot3D::persp3D`.

Value

A ggplot with a geom_tile layer providing a plot of the bivariate functional data object as a heat map.

Examples

```
library(funcharts)
mfdobj <- data_sim_mfd(nobs = 1)
tp <- tensor_product_mfd(mfdobj)
plot_bifd(tp)
```

`plot_bootstrap_sof_pc` *Plot bootstrapped estimates of the scalar-on-function regression coefficient*

Description

Plot bootstrapped estimates of the scalar-on-function regression coefficient for empirical uncertainty quantification. For each iteration, a data set is sampled with replacement from the training data use to fit the model, and the regression coefficient is estimated.

Usage

```
plot_bootstrap_sof_pc(mod, nboot = 25, ncores = 1)
```

Arguments

`mod`	A list obtained as output from `sof_pc`, i.e. a fitted scalar-on-function linear regression model.
`nboot`	Number of bootstrap replicates
`ncores`	If you want estimate the bootstrap replicates in parallel, give the number of cores/threads.

Value

A ggplot showing several bootstrap replicates of the multivariate functional coefficients estimated fitting the scalar-on-function linear model. Gray lines indicate the different bootstrap estimates, the black line indicate the estimate on the entire dataset.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates], lambda = 1e-2)
y <- rowMeans(air$NO2)
mod <- sof_pc(y, mfdobj_x)
plot_bootstrap_sof_pc(mod, nboot = 5)
```

plot_control_charts *Plot control charts*

Description

This function takes as input a data frame produced with functions such as control_charts_pca and control_charts_sof_pc and produces a ggplot with the desired control charts, i.e. it plots a point for each observation in the phase II data set against the corresponding control limits.

Usage

```
plot_control_charts(cclist, nobsI = 0)
```

Arguments

cclist	A data.frame produced by control_charts_pca, control_charts_sof_pc regr_cc_fof, or regr_cc_sof.
nobsI	An integer indicating the first observations that are plotted in gray. It is useful when one wants to plot the phase I data set together with the phase II data. In that case, one needs to indicate the number of phase I observations included in cclist. Default is zero.

Details

Out-of-control points are signaled by colouring them in red.

Value

A ggplot with the functional control charts.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:100, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
mfdobj_y <- get_mfd_list(air["NO2"],
                         n_basis = 15,
                         lambda = 1e-2)
mfdobj_y1 <- mfdobj_y[1:60]
mfdobj_y_tuning <- mfdobj_y[61:90]
mfdobj_y2 <- mfdobj_y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod_fof <- fof_pc(mfdobj_y1, mfdobj_x1)
```

```
cclist <- regr_cc_fof(mod_fof,
                      mfdobj_y_new = mfdobj_y2,
                      mfdobj_x_new = mfdobj_x2,
                      mfdobj_y_tuning = NULL,
                      mfdobj_x_tuning = NULL)
plot_control_charts(cclist)
```

`plot_control_charts_real_time` *Plot real-time control charts*

Description

This function produces a ggplot with the desired real-time control charts. It takes as input a list of data frames, produced with functions such as `regr_cc_fof_real_time` and `control_charts_sof_pc_real_time`, and the id of the observations for which real-time control charts are desired to be plotted. For each control chart, the solid line corresponds to the profile of the monitoring statistic and it is compared against control limits plotted as dashed lines. If a line is outside its limits it is coloured in red.

Usage

```
plot_control_charts_real_time(cclist, id_num)
```

Arguments

`cclist`	A list of data frames, produced with functions such as `regr_cc_fof_real_time` and `control_charts_sof_pc_real_time`,
`id_num`	An index number giving the observation in the phase II data set to be plotted, i.e. 1 for the first observation, 2 for the second, and so on.

Details

If the line, representing the profile of the monitoring statistic over the functional domain, is out-of-control, then it is coloured in red.

Value

A ggplot with the real-time functional control charts.

See Also

`regr_cc_fof_real_time`, `control_charts_sof_pc_real_time`

Examples

```
library(funcharts)
data("air")
air1 <- lapply(air, function(x) x[1:8, , drop = FALSE])
air2 <- lapply(air, function(x) x[9:10, , drop = FALSE])
mfdobj_x1_list <- get_mfd_list_real_time(air1[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_x2_list <- get_mfd_list_real_time(air2[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
y1 <- rowMeans(air1$NO2)
y2 <- rowMeans(air2$NO2)
mod_list <- sof_pc_real_time(y1, mfdobj_x1_list)
cclist <- regr_cc_sof_real_time(
  mod_list = mod_list,
  y_new = y2,
  mfdobj_x_new = mfdobj_x2_list,
  include_covariates = TRUE)
plot_control_charts_real_time(cclist, 1)
```

---

plot_mfd *Plot a Multivariate Functional Data Object.*

Description

Plot an object of class `mfd` using `ggplot2` and `patchwork`.

Usage

```
plot_mfd(
  mfdobj,
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = TRUE,
  orientation = NA,
  show.legend = NA,
  inherit.aes = TRUE,
  type_mfd = "mfd",
  y_lim_equal = FALSE,
  ...
)
```

Arguments

mfdobj	A multivariate functional data object of class mfd.
mapping	Set of aesthetic mappings additional to x and y as passed to the function ggplot2::geom:line.
data	A data.frame providing columns to create additional aesthetic mappings. It must contain a factor column "id" with the replication values as in mfdobj$fdnames[[2]]. If it contains a column "var", this must contain the functional variables as in mfdobj$fdnames[[3]].
stat	See ggplot2::geom_line.
position	See ggplot2::geom_line.
na.rm	See ggplot2::geom_line.
orientation	See ggplot2::geom_line.
show.legend	See ggplot2::geom_line.
inherit.aes	See ggplot2::geom_line.
type_mfd	A character value equal to "mfd" or "raw". If "mfd", the smoothed functional data are plotted, if "raw", the original discrete data are plotted.
y_lim_equal	A logical value. If TRUE, the limits of the y-axis are the same for all functional variables. If FALSE, limits are different for each variable. Default value is FALSE.
...	See ggplot2::geom_line.

Value

A plot of the multivariate functional data object.

Examples

```
library(funcharts)
library(ggplot2)
mfdobj <- data_sim_mfd()
ids <- mfdobj$fdnames[[2]]
df <- data.frame(id = ids, first_two_obs = ids %in% c("rep1", "rep2"))
plot_mfd(mapping = aes(colour = first_two_obs),
         data = df,
         mfdobj = mfdobj)
```

---

plot_mon *Plot multivariate functional object over the training data set*

Description

This function plots selected functions in a phase_II monitoring data set against the corresponding training data set to be compared.

Usage

```
plot_mon(cclist, fd_train, fd_test, plot_title = FALSE, print_id = FALSE)
```

Arguments

cclist	A data.frame produced by control_charts_pca, control_charts_sof_pc regr_cc_fof, or regr_cc_sof.
fd_train	An object of class mfd containing the training data set of the functional variables. They are plotted in gray in the background.
fd_test	An object of class mfd containing the phase II data set of the functional variables to be monitored. They are coloured in black or red on the foreground.
plot_title	A logical value. If TRUE, it prints the title with the observation name. Default is FALSE.
print_id	A logical value. If TRUE, and also plot_title is TRUE, it prints also the id of the observation in the title of the ggplot. Default is FALSE

Value

A ggplot of the multivariate functional data. In particular, the multivariate functional data given in fd_train are plotted on the background in gray, while the multivariate functional data given in fd_test are plotted on the foreground, the colour of each curve is black or red depending on if that curve was signal as anomalous by at least a contribution plot.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[201:300, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:60]
y_tuning <- y[61:90]
y2 <- y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod <- sof_pc(y1, mfdobj_x1)
cclist <- regr_cc_sof(object = mod,
                      y_new = y2,
                      mfdobj_x_new = mfdobj_x2,
                      y_tuning = y_tuning,
                      mfdobj_x_tuning = mfdobj_x_tuning,
                      include_covariates = TRUE)
get_ooc(cclist)
cont_plot(cclist, 3)
plot_mon(cclist, fd_train = mfdobj_x1, fd_test = mfdobj_x2[3])
```

plot_pca_mfd *Plot the harmonics of a* pca_mfd *object*

Description

Plot the harmonics of a pca_mfd object

Usage

```
plot_pca_mfd(pca, harm = 0, scaled = FALSE)
```

Arguments

pca	A fitted multivariate functional principal component analysis (MFPCA) object of class pca_mfd.
harm	A vector of integers with the harmonics to plot. If 0, all harmonics are plotted. Default is 0.
scaled	If TRUE, eigenfunctions are multiplied by the square root of the corresponding eigenvalues, if FALSE the are not scaled and the all have unit norm. Default is FALSE

Value

A ggplot of the harmonics/multivariate functional principal components contained in the object pca.

Examples

```
library(funcharts)
mfdobj <- data_sim_mfd()
pca_obj <- pca_mfd(mfdobj)
plot_pca_mfd(pca_obj)
```

plus_mfd *Add multivariate functional data*

Description

Adds two objects of class mfd (elementwise on their coefficient arrays). The two objects must share the same basis system and number of variables. If one object contains a single replication (i.e., one observation) and the other contains multiple, the single replication is replicated across observations before addition.

Usage

```
plus_mfd(mfdobj1, mfdobj2)

## S3 method for class 'mfd'
mfdobj1 + mfdobj2
```

Arguments

mfdobj1, mfdobj2	Objects of class mfd. If mfdobj2 is missing, unary plus is applied.

Details

If mfdobj2 is missing, the function returns mfdobj1 unchanged (i.e., unary plus).

Let the coefficient arrays have dimensions $(nbasis, nobs, nvar)$. The following checks/rules are enforced:

- Both inputs must be mfd objects; otherwise an error is thrown.
- The basis systems must be identical (checked via identical()).
- The number of variables must match.
- For the number of observations: if both $nobs_1$ and $nobs_2$ are greater than one, they must be equal; otherwise, the object with $nobs = 1$ is replicated to match the other.

Value

An object of class mfd with coefficients equal to the (possibly replicated) sum of the inputs. The fdnames are taken from the input providing the observation indexing after replication (if any), otherwise from mfdobj1.

See Also

nobs, nbasis, nvar, mfd

Examples

```
# Assuming mfdobj_a and mfdobj_b are 'mfd' objects on the same basis:
# mfdobj_a + mfdobj_b
# plus_mfd(mfdobj_a, mfdobj_b)
```

`predict.pca_mfd` *Predict from a multivariate functional PCA*

Description

Computes either the scores of new observations on selected principal components, or their reconstruction from the selected components, given a PCA fitted by `pca_mfd`.

Usage

```
## S3 method for class 'pca_mfd'
predict(
  object,
  newdata = NULL,
  components = seq_len(ncol(object$pcscores)),
  type = c("scores", "reconstruction"),
  ...
)
```

Arguments

`object`	An object of class `"pca_mfd"`, typically the output of `pca_mfd`.
`newdata`	An object of class `"mfd"` containing the new multivariate functional data to be projected. If `NULL`, the training data used to fit `object` are used.
`components`	Integer vector specifying the indices of the principal components to use. Defaults to all available components.
`type`	Character string: either `"scores"` (default) to return the scores of `newdata`, or `"reconstruction"` to return the data reconstructed from the selected components.
`...`	Further arguments passed to or from other methods (not used).

Details

This function is an S3 method for objects of class `"pca_mfd"`. It is usually called via the generic `predict` function.

The new data are first centered and (optionally) scaled using the functional center and scale stored in the PCA object.

- If `type = "scores"`, inner products with the selected eigenfunctions are computed and summed across basis functions.
- If `type = "reconstruction"`, the predicted functional data are reconstructed from the scores and harmonics.

Value

- If `type = "scores"`, a numeric matrix of dimension $nobs \times length(components)$.
- If `type = "reconstruction"`, an object of class `"mfd"`.

See Also

pca_mfd, scale_mfd

---

predict_fof_pc *Use a function-on-function linear regression model for prediction*

Description

Predict new observations of the functional response variable and calculate the corresponding prediction error (and their standardized or studentized version) given new observations of functional covariates and a fitted function-on-function linear regression model.

Usage

```
predict_fof_pc(object, mfdobj_y_new, mfdobj_x_new)
```

Arguments

object	A list obtained as output from fof_pc, i.e. a fitted function-on-function linear regression model.
mfdobj_y_new	An object of class mfd containing new observations of the functional response.
mfdobj_x_new	An object of class mfd containing new observations of the functional covariates.

Value

A list of mfd objects. It contains:

- pred_error: the prediction error of the standardized functional response variable,
- pred_error_original_scale: the prediction error of the functional response variable on the original scale,
- y_hat_new: the prediction of the functional response observations on the original scale,
- y_z_new: the standardized version of the functional response observations provided in mfdobj_y_new,
- y_hat_z_new: the prediction of the functional response observations on the standardized/studentized scale.

References

Centofanti F, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2021) Functional Regression Control Chart. *Technometrics*, 63(3):281–294. doi:10.1080/00401706.2020.1753581

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates], lambda = 1e-2)
mfdobj_y <- get_mfd_list(air["NO2"], lambda = 1e-2)
mod <- fof_pc(mfdobj_y, mfdobj_x)
predict_fof_pc(mod,
               mfdobj_y_new = mfdobj_y,
               mfdobj_x_new = mfdobj_x)
```

---

predict_sof_pc *Use a scalar-on-function linear regression model for prediction*

Description

Predict new observations of the scalar response variable and calculate the corresponding prediction error, with prediction interval limits, given new observations of functional covariates and a fitted scalar-on-function linear regression model

Usage

```
predict_sof_pc(
  object,
  y_new = NULL,
  mfdobj_x_new = NULL,
  alpha = 0.05,
  newdata
)
```

Arguments

object	A list obtained as output from sof_pc, i.e. a fitted scalar-on-function linear regression model.
y_new	A numeric vector containing the new observations of the scalar response variable to be predicted.
mfdobj_x_new	An object of class mfd containing new observations of the functional covariates. If NULL, it is set as the functional covariates data used for model fitting.
alpha	A numeric value indicating the Type I error for the regression control chart and such that this function returns the 1-alpha prediction interval on the response. Default is 0.05.
newdata	Deprecated, use mfdobj_x_new argument.

Value

A data.frame with as many rows as the number of functional replications in newdata, with the following columns:

- fit: the predictions of the response variable corresponding to new_data,
- lwr: lower limit of the 1-alpha prediction interval on the response, based on the assumption that it is normally distributed.
- upr: upper limit of the 1-alpha prediction interval on the response, based on the assumption that it is normally distributed.
- res: the residuals obtained as the values of y_new minus their fitted values. If the scalar-on-function model has been fitted with type_residual == "studentized", then the studentized residuals are calculated.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates], lambda = 1e-2)
y <- rowMeans(air$NO2)
mod <- sof_pc(y, mfdobj_x)
predict_sof_pc(mod)
```

---

rbind_mfd *Bind replications of two Multivariate Functional Data Objects*

Description

Bind replications of two Multivariate Functional Data Objects

Usage

```
rbind_mfd(mfdobj1, mfdobj2)
```

Arguments

mfdobj1	An object of class mfd, with the same variables of mfdobj2 and different replication names with respect to mfdobj2.
mfdobj2	An object of class mfd, with the same variables of mfdobj1, and different replication names with respect to mfdobj1.

Value

An object of class mfd, whose variables are the same of mfdobj1 and mfdobj2 and whose replications are the union of the replications in mfdobj1 and mfdobj2.

Examples

```
library(funcharts)
mfdobj1 <- data_sim_mfd(nvar = 3, nobs = 4)
mfdobj2 <- data_sim_mfd(nvar = 3, nobs = 5)
dimnames(mfdobj2$coefs)[[2]] <-
  mfdobj2$fdnames[[2]] <-
  c("rep11", "rep12", "rep13", "rep14", "rep15")
mfdobj_rbind <- rbind_mfd(mfdobj1, mfdobj2)
plot_mfd(mfdobj_rbind)
```

`regr_cc_fof` *Functional Regression Control Chart*

Description

It builds a data frame needed to plot the Functional Regression Control Chart introduced in Centofanti et al. (2021), for monitoring a functional quality characteristic adjusted for by the effect of multivariate functional covariates, based on a fitted function-on-function linear regression model. The training data have already been used to fit the model. An optional tuning data set can be provided that is used to estimate the control chart limits. A phase II data set contains the observations to be monitored with the control charts. It also allows to jointly monitor the multivariate functional covariates.

Usage

```
regr_cc_fof(
  object,
  mfdobj_y_new,
  mfdobj_x_new,
  mfdobj_y_tuning = NULL,
  mfdobj_x_tuning = NULL,
  alpha = 0.05,
  include_covariates = FALSE,
  absolute_error = FALSE
)
```

Arguments

`object`	A list obtained as output from `fof_pc`, i.e. a fitted function-on-function linear regression model.
`mfdobj_y_new`	An object of class `mfd` containing the phase II data set of the functional response observations to be monitored.
`mfdobj_x_new`	An object of class `mfd` containing the phase II data set of the functional covariates observations to be monitored.

`mfdobj_y_tuning`
: An object of class `mfd` containing the tuning data set of the functional response observations, used to estimate the control chart limits. If NULL, the training data, i.e. the data used to fit the function-on-function linear regression model, are also used as the tuning data set, i.e. `mfdobj_y_tuning=object$pca_y$data`. Default is NULL.

`mfdobj_x_tuning`
: An object of class `mfd` containing the tuning data set of the functional covariates observations, used to estimate the control chart limits. If NULL, the training data, i.e. the data used to fit the function-on-function linear regression model, are also used as the tuning data set, i.e. `mfdobj_x_tuning=object$pca_x$data`. Default is NULL.

`alpha`
: If it is a number between 0 and 1, it defines the overall type-I error probability. By default, it is equal to 0.05 and the Bonferroni correction is applied by setting the type-I error probabilities equal to `alpha/2` in the Hotelling's T2 and SPE control charts. If `include_covariates` is `TRUE`, i.e., the Hotelling's T2 and SPE control charts are built also on the multivariate functional covariates, then the Bonferroni correction is applied by setting the type-I error probability in the four control charts equal to `alpha/4`. If you want to set manually the Type-I error probabilities, then the argument `alpha` must be a named list with elements named as `T2`, `spe`, `T2_x` and, `spe_x`, respectively, containing the desired Type I error probability of the T2 and SPE control charts for the functional response and the multivariate functional covariates, respectively.

`include_covariates`
: If TRUE, also functional covariates are monitored through `control_charts_pca`,. If FALSE, only the functional response, conditionally on the covariates, is monitored.

`absolute_error`
: A logical value that, if `include_covariates` is TRUE, is passed to `control_charts_pca`.

Value

A `data.frame` containing the output of the function `control_charts_pca` applied to the prediction errors.

References

Centofanti F, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2021) Functional Regression Control Chart. *Technometrics*, 63(3):281–294. doi:10.1080/00401706.2020.1753581

See Also

`control_charts_pca`

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:100, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
```

```
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
mfdobj_y <- get_mfd_list(air["NO2"],
                         n_basis = 15,
                         lambda = 1e-2)
mfdobj_y1 <- mfdobj_y[1:60]
mfdobj_y_tuning <- mfdobj_y[61:90]
mfdobj_y2 <- mfdobj_y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod_fof <- fof_pc(mfdobj_y1, mfdobj_x1)
cclist <- regr_cc_fof(mod_fof,
                      mfdobj_y_new = mfdobj_y2,
                      mfdobj_x_new = mfdobj_x2,
                      mfdobj_y_tuning = NULL,
                      mfdobj_x_tuning = NULL)
plot_control_charts(cclist)
```

`regr_cc_fof_real_time` *Real-time functional regression control chart*

Description

This function produces a list of data frames, each of them is produced by `regr_cc_fof` and is needed to plot control charts for monitoring in real time a functional quality characteristic adjusted for by the effect of multivariate functional covariates.

Usage

```
regr_cc_fof_real_time(
  mod_list,
  mfdobj_y_new_list,
  mfdobj_x_new_list,
  mfdobj_y_tuning_list = NULL,
  mfdobj_x_tuning_list = NULL,
  alpha = 0.05,
  include_covariates = FALSE,
  absolute_error = FALSE,
  ncores = 1
)
```

Arguments

`mod_list`	A list of lists produced by `fof_pc_real_time`, containing a list of function-on-function linear regression models estimated on functional data each evolving up to an intermediate domain point.

mfdobj_y_new_list
: A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the phase II monitoring data set, each evolving up to an intermediate domain point, with observations of the functional response variable The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point.

mfdobj_x_new_list
: A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the phase II monitoring data set, each evolving up to an intermediate domain point, with observations of the multivariate functional covariates. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point.

mfdobj_y_tuning_list
: A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the tuning data set (used to estimate control chart limits), each evolving up to an intermediate domain point, with observations of the functional response variable. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point. If NULL, the training data, i.e. the functional response in mod_list, is also used as the tuning data set. Default is NULL.

mfdobj_x_tuning_list
: A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the tuning data set (used to estimate control chart limits), each evolving up to an intermediate domain point, with observations of the multivariate functional covariates. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point. If NULL, the training data, i.e. the functional covariates in mod_list, are also used as the tuning data set. Default is NULL.

alpha
: See regr_cc_fof.

include_covariates
: See regr_cc_fof.

absolute_error
: See regr_cc_fof.

ncores
: If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of data.frames each produced by regr_cc_fof, corresponding to a given instant.

See Also

fof_pc_real_time, regr_cc_fof

Examples

```
library(funcharts)
data("air")
air1 <- lapply(air, function(x) x[1:8, , drop = FALSE])
air2 <- lapply(air, function(x) x[9:10, , drop = FALSE])
mfdobj_x1_list <- get_mfd_list_real_time(air1[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_x2_list <- get_mfd_list_real_time(air2[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_y1_list <- get_mfd_list_real_time(air1["NO2"],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_y2_list <- get_mfd_list_real_time(air2["NO2"],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mod_list <- fof_pc_real_time(mfdobj_y1_list, mfdobj_x1_list)
cclist <- regr_cc_fof_real_time(
  mod_list = mod_list,
  mfdobj_y_new_list = mfdobj_y2_list,
  mfdobj_x_new_list = mfdobj_x2_list)
plot_control_charts_real_time(cclist, 1)
```

---

regr_cc_sof *Scalar-on-Function Regression Control Chart*

Description

This function is deprecated. Use regr_cc_sof. This function builds a data frame needed to plot the scalar-on-function regression control chart, based on a fitted function-on-function linear regression model and proposed in Capezza et al. (2020). If include_covariates is TRUE, it also plots the Hotelling's T2 and squared prediction error control charts built on the multivariate functional covariates.

Usage

```
regr_cc_sof(
  object,
  y_new,
  mfdobj_x_new,
  y_tuning = NULL,
  mfdobj_x_tuning = NULL,
  alpha = 0.05,
```

```
  parametric_limits = FALSE,
  include_covariates = FALSE,
  absolute_error = FALSE
)
```

Arguments

object	A list obtained as output from sof_pc, i.e. a fitted scalar-on-function linear regression model.
y_new	A numeric vector containing the observations of the scalar response variable in the phase II data set.
mfdobj_x_new	An object of class mfd containing the phase II data set of the functional covariates observations.
y_tuning	A numeric vector containing the observations of the scalar response variable in the tuning data set. If NULL, the training data, i.e. the data used to fit the scalar-on-function regression model, are also used as the tuning data set. Default is NULL.
mfdobj_x_tuning	An object of class mfd containing the tuning set of the multivariate functional data, used to estimate the control chart limits. If NULL, the training data, i.e. the data used to fit the scalar-on-function regression model, are also used as the tuning data set. Default is NULL.
alpha	If it is a number between 0 and 1, it defines the overall type-I error probability. If include_covariates is TRUE, i.e., also the Hotelling's T2 and SPE control charts are built on the functional covariates, then the Bonferroni correction is applied by setting the type-I error probability in the three control charts equal to alpha/3. In this last case, if you want to set manually the Type-I error probabilities, then the argument alpha must be a named list with three elements, named T2, spe and y, respectively, each containing the desired Type I error probability of the corresponding control chart, where y refers to the regression control chart. Default value is 0.05.
parametric_limits	If TRUE, the limits are calculated based on the normal distribution assumption on the response variable, as in Capezza et al. (2020). If FALSE, the limits are calculated nonparametrically as empirical quantiles of the distribution of the residuals calculated on the tuning data set. The default value is FALSE.
include_covariates	If TRUE, also functional covariates are monitored through control_charts_pca,. If FALSE, only the scalar response, conditionally on the covariates, is monitored.
absolute_error	A logical value that, if include_covariates is TRUE, is passed to control_charts_pca.

Details

The training data have already been used to fit the model. An additional tuning data set can be provided that is used to estimate the control chart limits. A phase II data set contains the observations to be monitored with the built control charts.

Value

A `data.frame` with as many rows as the number of functional replications in `mfdobj_x_new`, with the following columns:

- `y_hat`: the predictions of the response variable corresponding to `mfdobj_x_new`,
- `y`: the same as the argument `y_new` given as input to this function,
- `lwr`: lower limit of the `1-alpha` prediction interval on the response,
- `pred_err`: prediction error calculated as `y-y_hat`,
- `pred_err_sup`: upper limit of the `1-alpha` prediction interval on the prediction error,
- `pred_err_inf`: lower limit of the `1-alpha` prediction interval on the prediction error.

References

Capezza C, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2020) Control charts for monitoring ship operating conditions and CO2 emissions based on scalar-on-function regression. *Applied Stochastic Models in Business and Industry*, 36(3):477–500. doi:10.1002/asmb.2507

Examples

```
library(funcharts)
air <- lapply(air, function(x) x[1:100, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:80]
y2 <- y[81:100]
mfdobj_x1 <- mfdobj_x[1:80]
mfdobj_x2 <- mfdobj_x[81:100]
mod <- sof_pc(y1, mfdobj_x1)
cclist <- regr_cc_sof(object = mod,
                      y_new = y2,
                      mfdobj_x_new = mfdobj_x2)
plot_control_charts(cclist)
```

---

regr_cc_sof_real_time *Real-time Scalar-on-Function Regression Control Chart*

Description

This function builds a list of data frames, each of them is produced by `regr_cc_sof` and is needed to plot control charts for monitoring in real time a scalar quality characteristic adjusted for by the effect of multivariate functional covariates. The training data have already been used to fit the model. An additional tuning data set can be provided that is used to estimate the control chart limits. A phase II data set contains the observations to be monitored with the built control charts.

Usage

```
regr_cc_sof_real_time(
  mod_list,
  y_new,
  mfdobj_x_new_list,
  y_tuning = NULL,
  mfdobj_x_tuning_list = NULL,
  alpha = 0.05,
  parametric_limits = TRUE,
  include_covariates = FALSE,
  absolute_error = FALSE,
  ncores = 1
)
```

Arguments

mod_list	A list of lists produced by sof_pc_real_time, containing a list of scalar-on-function linear regression models estimated on functional data each evolving up to an intermediate domain point.
y_new	A numeric vector containing the observations of the scalar response variable in the phase II monitoring data set.
mfdobj_x_new_list	A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the phase II monitoring data set, each evolving up to an intermediate domain point, with observations of the multivariate functional covariates. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point.
y_tuning	An optional numeric vector containing the observations of the scalar response variable in the tuning data set. If NULL, the training data, i.e. the scalar response in mod_list, is also used as the tuning data set. Default is NULL.
mfdobj_x_tuning_list	A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects in the tuning data set (used to estimate control chart limits), each evolving up to an intermediate domain point, with observations of the multivariate functional covariates. The length of this list and mod_list must be equal, and their elements in the same position in the list must correspond to the same intermediate domain point. If NULL, the training data, i.e. the functional covariates in mod_list, are also used as the tuning data set. Default is NULL.
alpha	See regr_cc_sof.
parametric_limits	See regr_cc_sof.
include_covariates	See regr_cc_sof.
absolute_error	See regr_cc_sof.

ncores — If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of data.frames each produced by regr_cc_sof, corresponding to a given instant.

See Also

sof_pc_real_time, regr_cc_sof

Examples

```
library(funcharts)
data("air")
air1 <- lapply(air, function(x) x[1:8, , drop = FALSE])
air2 <- lapply(air, function(x) x[9:10, , drop = FALSE])
mfdobj_x1_list <- get_mfd_list_real_time(air1[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_x2_list <- get_mfd_list_real_time(air2[c("CO", "temperature")],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_y1_list <- get_mfd_list_real_time(air1["NO2"],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mfdobj_y2_list <- get_mfd_list_real_time(air2["NO2"],
                                         n_basis = 15,
                                         lambda = 1e-2,
                                         k_seq = c(0.5, 1))
mod_list <- fof_pc_real_time(mfdobj_y1_list, mfdobj_x1_list)
cclist <- regr_cc_fof_real_time(
  mod_list = mod_list,
  mfdobj_y_new_list = mfdobj_y2_list,
  mfdobj_x_new_list = mfdobj_x2_list)
plot_control_charts_real_time(cclist, 1)
```

RoAMFEWMA_PhaseI — *Robust Adaptive Multivariate Functional EWMA Control Chart - Phase I*

Description

It performs Phase I of the Robust Adaptive Multivariate Functional EWMA control chart (RoAMFEWMA). The procedure combines:

1. Functional cellwise outlier detection via functional_filter,
2. Robust Multivariate Functional Data Imputation (RoMFDI) via RoMFDI,
3. Casewise outliers detection via RoMFCC (RoMFCC_PhaseI_casewise and RoMFCC_PhaseII_casewise), on the imputed Phase I data,
4. AMFEWMA Phase I calibration (AMFEWMA_PhaseI) on cellwise and casewise clean data.

The resulting object can be directly used as mod_1 argument in AMFEWMA_PhaseII

Usage

```
RoAMFEWMA_PhaseI(
  mfdobj,
  mfdobj_tuning,
  functional_filter_par = list(filter = TRUE),
  imputation_par = list(method_imputation = "RoMFDI", n_dataset = 1),
  verbose = FALSE
)
```

Arguments

mfdobj	A multivariate functional data object of class mfd. This dataset is used as the Phase I **training set**. A functional filter is first applied to detect functional cellwise outliers; the flagged components are then imputed through a robust multivariate functional imputation procedure. The imputed data are subsequently processed by the Robust Multivariate Functional Control Chart to detect and remove functional casewise outliers. The resulting clean dataset (free of both cellwise and casewise outliers) is then used as the training set in the Phase I of the Adaptive Multivariate Functional EWMA control chart.
mfdobj_tuning	A multivariate functional data object of class mfd representing the Phase I **tuning set**. This dataset undergoes the same functional filtering and robust imputation steps applied to mfdobj. The filtered and imputed data are used within the Robust Multivariate Functional Control Chart to estimate tuning quantities and control limits. After casewise cleaning, the resulting clean tuning set is employed in the Phase I calibration of the Adaptive Multivariate Functional EWMA control chart.
functional_filter_par	A list with an argument filter that can be TRUE or FALSE depending on if the functional filter step must be performed or not. All the other arguments of this list are passed as arguments to the function functional_filter in the filtering step. All the arguments that are not passed take their default values. See functional_filter for all the arguments and their default values. Default is list(filter = TRUE).
imputation_par	A list with an argument method_imputation that can be "RoMFDI" or "mean" depending on if the imputation step must be done by means of RoMFDI or by just using the mean of each functional variable. If method_imputation = "RoMFDI", all the other arguments of this list are passed as arguments to the function RoMFDI in the imputation step. All the arguments that are not passed take their default

values. See RoMFDI for all the arguments and their default values. Default value is `list(method_imputation = "RoMFDI")`.

`verbose` If TRUE, it prints messages about the steps of the algorithm. Default is FALSE.

Details

Among the multiple imputed datasets, the first one is used to build the cleaned training and tuning sets for AMFEWMA.

Value

A list of the following elements:

- `mod_1` object returned by `AMFEWMA_PhaseI`, see the value of AMFEWMA_PhaseI for a full description of its component;
- `mfd_clean_training` training data after complete cleaning, containing no outliers at either cellwise or casewise;
- `mfd_clean_tuning` tuning data after complete cleaning, containing no outliers at either cellwise or casewise;
- `mfd_all_clean` full Phase I clean data (training + tuning);
- `idx_casewise_outliers` indices of observations indetified as casewise outliers by RoMFCC Phase II;
- `ff_training` training set after the functional filter;
- `ff_tuning` tuning set after the functional filter;
- `X_imp_training_1` first imputation of the training set after RoMFDI
- `X_imp_tuning_1` first imputation of the tuning set after RoMFDI
- `X_all_imputed` training + tuning data after robust multivariate functional imputation;
- `mod_RoMFCC_phaseI_casewise` object returned by RoMFCC_PhaseI, see the value of RoMFCC_PhaseI_casewise for a full description of its component;
- `mod_RoMFCC_phaseII_casewise` object returned by RoMFCC_PhaseII, see the value of RoMFCC_PhaseII_casewise for a full description of its component;

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Capezza, C., Capizzi, G., Centofanti, F., Lepore, A., Palumbo, B. (2025) An Adaptive Multivariate Functional EWMA Control Chart. *Journal of Quality Technology*, 57(1):1–15, doi:https://doi.org/10.1080/00224065.2024.23

Examples

```
## Not run:
set.seed(0)
dat_phaseI <- simulate_data_RoMFCC(p_cellwise = 0.05,
                                   p_casewise = 0.05,
                                   outlier = "outlier_E",
```

```
                            M_outlier_cell = 0.03,
                            M_outlier_case = 0.01,
                            max_n_cellwise = 10)
dat_phaseII <- simulate_data_RoMFCC(OC = "OC_E",
                                    M_OC = 0.01,
                                    which_OC = 5)
mfdobj_phaseI <- get_mfd_list(dat_phaseI$X_mat_list, n_basis = 5)
mfdobj_phaseII <- get_mfd_list(dat_phaseII$X_mat_list, n_basis = 5)
mfdobj_training_phaseI <- mfdobj_phaseI[1:333, ]
mfdobj_tuning_phaseI <- mfdobj_phaseI[334:1000, ]
out_phaseI <- RoAMFEWMA_PhaseI(mfdobj = mfdobj_training_phaseI,
                               mfdobj_tuning = mfdobj_tuning_phaseI)
out_phaseII <- RoAMFEWMA_PhaseII(mfdobj_2 = mfdobj_phaseII,
                                 mod_1 = out_phaseI)
plot_control_charts(out_phaseII$cc)

## End(Not run)
```

`RoAMFEWMA_PhaseII` *Robust Adaptive Multivariate Functional EWMA Control Chart - Phase II*

Description

This function performs Phase II of the Robust Adaptive Multivariate Functional EWMA (RoAMFEWMA) control chart.

Usage

```
RoAMFEWMA_PhaseII(mfdobj_2, mod_1, n_seq_2 = 1, l_seq_2 = 2000)
```

Arguments

`mfdobj_2`	An object of class `mfd` containing the Phase II multivariate functional data set, to be monitored with the RoAMFEWMA control chart.
`mod_1`	The output of the Phase I achieved through the `RoAMFEWMA_PhaseI` function.
`n_seq_2`	If it is 1, the Phase II monitoring statistic is calculated on the data sequence. If it is an integer number larger than 1, a number `n_seq_2` of bootstrap sequences are sampled with replacement from `mfdobj_2` to allow uncertainty quantification on the estimation of the run length. Default value is 1.
`l_seq_2`	If `n_seq_2` is larger than 1, this parameter sets the length of each bootstrap sequence to be generated. Default value is 2000.

Details

This function is conceptually similar to `AMFEWMA_PhaseII`, proposed by Capezza et al. (2024), but adapted to the RoAMFEWMA framework. In Phase II, monitoring relies on the RoAMFEWMA model calibrated in Phase I on data cleaned from both cellwise and casewise outliers. The monitoring statistic, control limit, and bootstrap-based ARL estimation remain unchanged, but the input model must be the robust one obtained through `RoAMFEWMA_PhaseI`.

Value

A list with the following elements.

- ARL_2: the average run length estimated over the bootstrap sequences. If n_seq_2 is 1, it is simply the run length observed over the Phase II sequence, i.e., the number of observations up to the first alarm,
- RL: the run length observed over the Phase II sequence, i.e., the number of observations up to the first alarm,
- V2: a list with length n_seq_2, containing the AMFEWMA monitoring statistic in Equation (8) of Capezza et al. (2024), calculated in each bootstrap sequence, until the first alarm.
- cc: a data frame with the information needed to plot the AMFEWMA control chart in Phase II, with the following columns. id contains the id of each multivariate functional observation, amfewma_monitoring_statistic contains the AMFEWMA monitoring statistic values calculated on the Phase II sequence, amfewma_monitoring_statistic_lim is the upper control limit.

References

Capezza, C., Capizzi, G., Centofanti, F., Lepore, A., Palumbo, B. (2025) An Adaptive Multivariate Functional EWMA Control Chart. *Journal of Quality Technology*, 57(1):1–15, doi:https://doi.org/10.1080/00224065.2024.23

Examples

```
## Not run:
set.seed(0)
dat_phaseI <- simulate_data_RoMFCC(p_cellwise = 0.05,
                           p_casewise = 0.05,
                           outlier = "outlier_E",
                           M_outlier_cell = 0.03,
                           M_outlier_case = 0.01,
                           max_n_cellwise = 10)
dat_phaseII <- simulate_data_RoMFCC(OC = "OC_E",
                                    M_OC = 0.01,
                                    which_OC = 5)
mfdobj_phaseI <- get_mfd_list(dat_phaseI$X_mat_list, n_basis = 5)
mfdobj_phaseII <- get_mfd_list(dat_phaseII$X_mat_list, n_basis = 5)
mfdobj_training_phaseI <- mfdobj_phaseI[1:333, ]
mfdobj_tuning_phaseI <- mfdobj_phaseI[334:1000, ]
out_phaseI <- RoAMFEWMA_PhaseI(mfdobj = mfdobj_training_phaseI,
                               mfdobj_tuning = mfdobj_tuning_phaseI)
out_phaseII <- RoAMFEWMA_PhaseII(mfdobj_2 = mfdobj_phaseII,
                                 mod_1 = out_phaseI)
plot_control_charts(out_phaseII$cc)

## End(Not run)
```

RoMFCC_PhaseI *Robust Multivariate Functional Control Charts - Phase I*

Description

It performs Phase I of the Robust Multivariate Functional Control Chart (RoMFCC) as proposed by Capezza et al. (2024).

Usage

```
RoMFCC_PhaseI(
  mfdobj,
  mfdobj_tuning = NULL,
  functional_filter_par = list(filter = TRUE),
  imputation_par = list(method_imputation = "RoMFDI"),
  pca_par = list(fev = 0.7),
  alpha = 0.05,
  verbose = FALSE
)
```

Arguments

`mfdobj`	A multivariate functional data object of class mfd. A functional filter is applied to this data set, then flagged functional componentwise outliers are imputed in the robust imputation step. Finally robust multivariate functional principal component analysis is applied to the imputed data set for dimension reduction.
`mfdobj_tuning`	An additional functional data object of class mfd. After applying the filter and imputation steps on this data set, it is used to robustly estimate the distribution of the Hotelling's T2 and SPE statistics in order to calculate control limits to prevent overfitting issues that could reduce the monitoring performance of the RoMFCC. Default is NULL, but it is strongly recommended to use a tuning data set.
`functional_filter_par`	A list with an argument `filter` that can be TRUE or FALSE depending on if the functional filter step must be performed or not. All the other arguments of this list are passed as arguments to the function `functional_filter` in the filtering step. All the arguments that are not passed take their default values. See `functional_filter` for all the arguments and their default values. Default is `list(filter = TRUE)`.
`imputation_par`	A list with an argument `method_imputation` that can be `"RoMFDI"` or `"mean"` depending on if the imputation step must be done by means of `RoMFDI` or by just using the mean of each functional variable. If `method_imputation = "RoMFDI"`, all the other arguments of this list are passed as arguments to the function `RoMFDI` in the imputation step. All the arguments that are not passed take their default values. See `RoMFDI` for all the arguments and their default values. Default value is `list(method_imputation = "RoMFDI")`.

pca_par	A list with an argument fev, indicating a number between 0 and 1 denoting the fraction of variability that must be explained by the principal components to be selected in the RoMFPCA step. All the other arguments of this list are passed as arguments to the function rpca_mfd in the RoMFPCA step. All the arguments that are not passed take their default values. See rpca_mfd for all the arguments and their default values. Default value is list(fev = 0.7).
alpha	The overall nominal type-I error probability used to set control chart limits. Default value is 0.05.
verbose	If TRUE, it prints messages about the steps of the algorithm. Default is FALSE.

Value

A list of the following elements that are needed in Phase II:

- T2 the Hotelling's T2 statistic values for the Phase I data set,
- SPE the SPE statistic values for the Phase I data set,
- T2_tun the Hotelling's T2 statistic values for the tuning data set,
- SPE_tun the SPE statistic values for the tuning data set,
- T2_lim the Phase II control limit of the Hotelling's T2 control chart,
- spe_lim the Phase II control limit of the SPE control chart,
- tuning TRUE if the tuning data set is provided, FALSE otherwise,
- mod_pca the final RoMFPCA model fitted on the Phase I data set,
- K = K the number of selected principal components,
- T_T2_inv if a tuning data set is provided, it returns the inverse of the covariance matrix of the first K scores, needed to calculate the Hotelling's T2 statistic for the Phase II observations.
- mean_scores_tuning_rob_mean if a tuning data set is provided, it returns the robust location estimate of the scores, needed to calculate the Hotelling's T2 and SPE statistics for the Phase II observations.

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Examples

```
library(funcharts)
mfdobj <- get_mfd_list(air, n_basis = 5)
nobs <- dim(mfdobj$coefs)[2]
set.seed(0)
ids <- sample(1:nobs)
mfdobj1 <- mfdobj[ids[1:100]]
mfdobj_tuning <- mfdobj[ids[101:300]]
mfdobj2 <- mfdobj[ids[-(1:300)]]
mod_phase1 <- RoMFCC_PhaseI(mfdobj = mfdobj1,
                            mfdobj_tuning = mfdobj_tuning,
                            functional_filter_par = list(filter = FALSE))
```

```
phase2 <- RoMFCC_PhaseII(mfdobj_new = mfdobj2,
                         mod_phase1 = mod_phase1)
plot_control_charts(phase2)
```

`RoMFCC_PhaseI_casewise` *Robust Multivariate Functional Control Charts - Phase I (casewise version)*

Description

It performs Phase I of the Robust Multivariate Functional Control Chart (RoMFCC), proposed by Capezza et al. (2024), applied to casewise outlier detection.

Usage

```
RoMFCC_PhaseI_casewise(
  mfdobj_imp,
  mfdobj_imp_tuning,
  pca_par = list(fev = 0.7),
  alpha_casewise = 0.0027,
  verbose = FALSE
)
```

Arguments

`mfdobj_imp`	A multivariate functional data object of class `mfd`, already imputed and filtered, so with no cellwise outliers. A robust multivariate principal component functional analysis is applied to the imputed dataset for dimension reduction.
`mfdobj_imp_tuning`	An additional functional data object of class `mfd`, already imputed and filtered, so with no cellwise outliers. It is used to robustly estimate the distribution of the Hotelling's T2 and SPE statistics in order to calculate control limits to prevent overfitting issues that could reduce the monitoring performance of the RoMFCC.
`pca_par`	A list with an argument `fev`, indicating a number between 0 and 1 denoting the fraction of variability that must be explained by the principal components to be selected in the RoMFPCA step. All the other arguments of this list are passed as arguments to the function `rpca_mfd` in the RoMFPCA step. All the arguments that are not passed take their default values. See `rpca_mfd` for all the arguments and their default values. Default value is `list(fev = 0.7)`.
`alpha_casewise`	The overall nominal type-I error probability used to set control chart limits and to identify functional casewise outliers Default value is 0.0027.
`verbose`	If TRUE, it prints messages about the steps of the algorithm. Default is FALSE.

Details

Unlike the original RoMFCC implementation, this version assumes that:

- functional filter
- robust multivariate functional imputation have already been applied to the training and tuning datasets. Therefore, the input data are expected to be multivariate functional data free of cellwise outliers (casewise outliers may still be present).

Value

A list of the following elements that are needed in Phase II:

- `T2` the Hotelling's T2 statistic values for the Phase I data set,
- `SPE` the SPE statistic values for the Phase I data set,
- `T2_tun` the Hotelling's T2 statistic values for the tuning data set,
- `SPE_tun` the SPE statistic values for the tuning data set,
- `T2_lim` the Phase II control limit of the Hotelling's T2 control chart,
- `spe_lim` the Phase II control limit of the SPE control chart,
- `mod_pca` the final RoMFPCA model fitted on the Phase I data set,
- `K` = K the number of selected principal components,
- `T_T2_inv` if a tuning data set is provided, it returns the inverse of the covariance matrix of the first `K` scores, needed to calculate the Hotelling's T2 statistic for the Phase II observations.
- `mean_scores_tuning_rob_mean` if a tuning data set is provided, it returns the robust location estimate of the scores, needed to calculate the Hotelling's T2 and SPE statistics for the Phase II observations.

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Examples

```
## Not run:
library(funcharts)
set.seed(0)
dat <- simulate_data_RoMFCC(p_cellwise = 0.05,
                            p_casewise = 0.05,
                            outlier = "outlier_E",
                            M_outlier_cell = 0.03,
                            M_outlier_case = 0.01,
                            max_n_cellwise = 10)
mfdobj <- get_mfd_list(dat$X_mat_list, n_basis = 5)
mfdobj_training <- mfdobj[1:333, ]
mfdobj_tuning <- mfdobj[334:1000, ]
ff_training <- functional_filter(mfdobj = mfdobj_training)
ff_tuning <- functional_filter(mfdobj = mfdobj_tuning)
```

```
x_imp_training <- RoMFDI(mfdobj = ff_training$mfdobj)
x_imp_tuning <- RoMFDI(mfdobj = ff_tuning$mfdobj)
X_imp_training <- x_imp_training[[1]]
X_imp_tuning <- x_imp_tuning[[1]]
out_phase1_casewise <- RoMFCC_PhaseI_casewise(
  mfdobj_imp = X_imp_training,
  mfdobj_imp_tuning = X_imp_tuning
)
mfd_all_imputed <- rbind_mfd(X_imp_training, X_imp_tuning)
out_phase2_casewise <- RoMFCC_PhaseII_casewise(
  mfdobj_all_imp = mfdobj_all_imputed,
  mod_phaseI_casewise = out_phase1_casewise
)
plot_control_charts(out_phase2_casewise)

## End(Not run)
```

`RoMFCC_PhaseII` *Robust Multivariate Functional Control Charts - Phase II*

Description

It calculates the Hotelling's and SPE monitoring statistics needed to plot the Robust Multivariate Functional Control Chart in Phase II.

Usage

```
RoMFCC_PhaseII(mfdobj_new, mod_phase1)
```

Arguments

`mfdobj_new`	A multivariate functional data object of class mfd, containing the Phase II observations to be monitored.
`mod_phase1`	Output obtained by applying the function `RoMFCC_PhaseI` to perform Phase I. See `RoMFCC_PhaseI`.

Value

A `data.frame` with as many rows as the number of multivariate functional observations in the phase II data set and the following columns:

- one `id` column identifying the multivariate functional observation in the phase II data set,
- one `T2` column containing the Hotelling T2 statistic calculated for all observations,
- one column per each functional variable, containing its contribution to the T2 statistic,
- one `spe` column containing the SPE statistic calculated for all observations,
- `T2_lim` gives the upper control limit of the Hotelling's T2 control chart,
- `spe_lim` gives the upper control limit of the SPE control chart

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Examples

```
library(funcharts)
mfdobj <- get_mfd_list(air, n_basis = 5)
nobs <- dim(mfdobj$coefs)[2]
set.seed(0)
ids <- sample(1:nobs)
mfdobj1 <- mfdobj[ids[1:100]]
mfdobj_tuning <- mfdobj[ids[101:300]]
mfdobj2 <- mfdobj[ids[-(1:300)]]
mod_phase1 <- RoMFCC_PhaseI(mfdobj = mfdobj1,
                            mfdobj_tuning = mfdobj_tuning,
                            functional_filter_par = list(filter = FALSE))
phase2 <- RoMFCC_PhaseII(mfdobj_new = mfdobj2,
                         mod_phase1 = mod_phase1)
plot_control_charts(phase2)
```

RoMFCC_PhaseII_casewise

Robust Multivariate Functional Control Charts - Phase II (casewise version)

Description

It performs Phase II of the Robust Multivariate Functional Control Chart (RoMFCC) for casewise outlier detection, computing Hotelling's T2 and SPE monitoring statistics according to the methodology proposed by Capezza et al. (2024).

Usage

```
RoMFCC_PhaseII_casewise(mfdobj_all_imp, mod_phaseI_casewise)
```

Arguments

`mfdobj_all_imp`	A multivariate functional data object of class `mfd`, containing the concatenation of the fully imputed training and tuning sets to be monitored for casewise outliers.
`mod_phaseI_casewise`	Output obtained by applying the function `RoMFCC_PhaseI_casewise` to perform Phase I. See `RoMFCC_PhaseI_casewise`

Value

A `data.frame` with as many rows as the number of multivariate functional observations in the phase II data set and the following columns:

- one `id` column identifying the multivariate functional observation in the phase II data set,
- one `T2` column containing the Hotelling T2 statistic calculated for all observations,
- one column per each functional variable, containing its contribution to the T2 statistic,
- one `SPE` column containing the SPE statistic calculated for all observations,
- `T2_lim` gives the upper control limit of the Hotelling's T2 control chart,
- `SPE_lim` gives the upper control limit of the SPE control chart

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Examples

```
## Not run:
library(funcharts)
set.seed(0)
dat <- simulate_data_RoMFCC(p_cellwise = 0.05,
                            p_casewise = 0.05,
                            outlier = "outlier_E",
                            M_outlier_cell = 0.03,
                            M_outlier_case = 0.01,
                            max_n_cellwise = 10)
mfdobj <- get_mfd_list(dat$X_mat_list, n_basis = 5)
mfdobj_training <- mfdobj[1:333, ]
mfdobj_tuning <- mfdobj[334:1000, ]
ff_training <- functional_filter(mfdobj = mfdobj_training)
ff_tuning <- functional_filter(mfdobj = mfdobj_tuning)
x_imp_training <- RoMFDI(mfdobj = ff_training$mfdobj)
x_imp_tuning <- RoMFDI(mfdobj = ff_tuning$mfdobj)
X_imp_training <- x_imp_training[[1]]
X_imp_tuning <- x_imp_tuning[[1]]
out_phase1_casewise <- RoMFCC_PhaseI_casewise(
  mfdobj_imp = X_imp_training,
  mfdobj_imp_tuning,X_imp_tuning
)
mfd_all_imputed <- rbind_mfd(X_imp_training, X_imp_tuning)
out_phase2_casewise <- RoMFCC_PhaseII_casewise(
  mfdobj_all_imp = mfdobj_all_imputed,
  mod_phaseI_casewise = out_phase1_casewise
)
plot_control_charts(out_phase2_casewise)

## End(Not run)
```

RoMFDI *Robust Multivariate Functional Data Imputation (RoMFDI)*

Description

It performs Robust Multivariate Functional Data Imputation (RoMFDI) as in Capezza et al. (2024).

Usage

```
RoMFDI(
  mfdobj,
  method_pca = "ROBPCA",
  fev = 0.999,
  n_dataset = 3,
  update = TRUE,
  niter_update = 10,
  alpha = 0.8
)
```

Arguments

mfdobj	A multivariate functional data object of class mfd.
method_pca	The method used in rpca_mfd to perform robust multivariate functional principal component analysis (RoMFPCA). See rpca_mfd. Default is "ROBPCA".
fev	Number between 0 and 1 denoting the proportion of variability that must be explained by the principal components to be selected for dimension reduction after applying RoMFPCA on the observed components to impute the missing ones. Default is 0.999.
n_dataset	To take into account the increased noise due to single imputation, the proposed RoMFDI allows multiple imputation. Due to the presence of the stochastic component in the imputation, it is worth explicitly noting that the imputed data set is not deterministically assigned. Therefore, by performing several times the RoMFDI in the imputation step of the RoMFCC implementation, the corresponding multiple estimated RoMFPCA models could be combined by averaging the robustly estimated covariance functions, thus performing a multiple imputation strategy as suggested by Van Ginkel et al. (2007). Default is 3.
update	The RoMFDI performs sequential imputation of missing functional components. If TRUE, Robust Multivariate Functional Principal Component Analysis (RoMFPCA) niter_update is updated times during the algorithm. If FALSE, the RoMFPCA used for imputation is always the same, i.e., the one performed on the original data sets containing only the observations with no missing functional components. Default is TRUE.
niter_update	The number of times the RoMFPCA is updated during the algorithm. It applies only if update is TRUE. Default value is 10.
alpha	This parameter measures the fraction of outliers the RoMFPCA algorithm should resist and is used only if method_pca is "ROBPCA". Default is 0.8.

Value

A list with `n_dataset` elements. Each element is an `mfd` object containing `mfdobj` with stochastic imputation of the missing components.

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Van Ginkel, J. R., Van der Ark, L. A., Sijtsma, K., and Vermunt, J. K. (2007). Two-way imputation: a Bayesian method for estimating missing scores in tests and questionnaires, and an accurate approximation. *Computational Statistics & Data Analysis*, 51(8):4013—4027.

Examples

```
library(funcharts)
mfdobj <- get_mfd_list(air[1:3], grid = 1:24, n_basis = 13, lambda = 1e-2)
out <- functional_filter(mfdobj, bivariate = FALSE)
mfdobj_imp <- RoMFDI(out$mfdobj, n_dataset = 1, update = FALSE)
```

`rpca_mfd` *Robust multivariate functional principal components analysis*

Description

It performs robust MFPCA as described in Capezza et al. (2024).

Usage

```
rpca_mfd(
  mfdobj,
  center = "fusem",
  scale = "funmad",
  nharm = 20,
  method = "ROBPCA",
  alpha = 0.8
)
```

Arguments

`mfdobj`	A multivariate functional data object of class mfd.
`center`	If TRUE, it centers the data before doing MFPCA with respect to the functional mean of the input data. If `"fusem"`, it uses the functional M-estimator of location proposed by Centofanti et al. (2023) to center the data. Default is `"fusem"`.

scale	If "funmad", it scales the data before doing MFPCA using the functional normalized median absolute deviation estimator proposed by Centofanti et al. (2023). If TRUE, it scales data using scale_mfd. Default is "funmad".
nharm	Number of multivariate functional principal components to be calculated. Default is 20.
method	If "ROBPCA", MFPCA uses ROBPCA of Hubert et al. (2005), as described in Capezza et al. (2024). If "Locantore", MFPCA uses the Spherical Principal Components procedure proposed by Locantore et al. (1999). If "Proj", MFPCA uses the Robust Principal Components based on Projection Pursuit algorithm of Croux and Ruiz-Gazen (2005). method If "normal", it uses pca_mfd on mfdobj. Default is "ROBPCA".
alpha	This parameter measures the fraction of outliers the algorithm should resist and is used only if method is "ROBPCA". Default is 0.8.

Value

An object of pca_mfd class, as returned by the pca_mfd function when performing non robust multivariate functional principal component analysis.

References

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Centofanti, F., Colosimo, B.M., Grasso, M.L., Menafoglio, A., Palumbo, B., Vantini, S. (2023) Robust functional ANOVA with application to additive manufacturing. *Journal of the Royal Statistical Society Series C: Applied Statistics* 72(5), 1210–1234 doi:10.1093/jrsssc/qlad074

Croux, C., Ruiz-Gazen, A. (2005). High breakdown estimators for principal components: The projection-pursuit approach revisited. *Journal of Multivariate Analysis*, 95, 206–226, doi:10.1016/j.jmva.2004.08.002.

Hubert, M., Rousseeuw, P.J., Branden, K.V. (2005) ROBPCA: A New Approach to Robust Principal Component Analysis, *Technometrics* 47(1), 64–79, doi:10.1198/004017004000000563

Locantore, N., Marron, J., Simpson, D., Tripoli, N., Zhang, J., Cohen K., K. (1999), Robust principal components for functional data. *Test*, 8, 1-28. doi:10.1007/BF02595862

Examples

```
library(funcharts)
dat <- simulate_mfd(nobs = 20, p = 1, correlation_type_x = "Bessel")
mfdobj <- get_mfd_list(dat$X_list, n_basis = 5)

# contaminate first observation
mfdobj$coefs[, 1, ] <- mfdobj$coefs[, 1, ] + 0.05

# plot_mfd(mfdobj) # plot functions to see the outlier
# pca <- pca_mfd(mfdobj) # non robust MFPCA
rpca <- rpca_mfd(mfdobj) # robust MFPCA
# plot_pca_mfd(pca, harm = 1) # plot first eigenfunction, affected by outlier
# plot_pca_mfd(rpca, harm = 1) # plot first eigenfunction in robust case
```

scale_mfd *Standardize Multivariate Functional Data.*

Description

Scale multivariate functional data contained in an object of class mfd by subtracting the mean function and dividing by the standard deviation function.

Usage

```
scale_mfd(mfdobj, center = TRUE, scale = TRUE)
```

Arguments

mfdobj	A multivariate functional data object of class mfd.
center	A logical value, or a fd object. When providing a logical value, if TRUE, mfdobj is centered, i.e. the functional mean function is calculated and subtracted from all observations in mfdobj, if FALSE, mfdobj is not centered. If center is a fd object, then this function is used as functional mean for centering.
scale	A logical value, or a fd object. When providing a logical value, if TRUE, mfdobj is scaled after possible centering, i.e. the functional standard deviation is calculated from all functional observations in mfdobj and then the observations are divided by this calculated standard deviation, if FALSE, mfdobj is not scaled. If scale is a fd object, then this function is used as standard deviation function for scaling.

Details

This function has been written to work similarly as the function scale for matrices. When calculated, attributes center and scale are of class fd and have the same structure you get when you use fda::mean.fd and fda::sd.fd.

Value

A standardized object of class mfd, with two attributes, if calculated, center and scale, storing the mean and standard deviation functions used for standardization.

Examples

```
library(funcharts)
mfdobj <- data_sim_mfd()
mfdobj_scaled <- scale_mfd(mfdobj)
```

`sim_funcharts` *Simulate example data for funcharts*

Description

Function used to simulate three data sets to illustrate the use of `funcharts`. It uses the function `simulate_mfd`, which creates a data set with three functional covariates, a functional response generated as a function of the three functional covariates, and a scalar response generated as a function of the three functional covariates. This function generates three data sets, one for phase I, one for tuning, i.e., to estimate the control chart limits, and one for phase II monitoring. see also `simulate_mfd`.

Usage

```
sim_funcharts(nobs1 = 1000, nobs_tun = 1000, nobs2 = 60)
```

Arguments

`nobs1`	The number of observation to simulate in phase I. Default is 1000.
`nobs_tun`	The number of observation to simulate the tuning data set. Default is 1000.
`nobs2`	The number of observation to simulate in phase II. Default is 60.

Value

A list with three objects, `datI` contains the phase I data, `datI_tun` contains the tuning data, `datII` contains the phase II data. In the phase II data, the first group of observations are in control, the second group of observations contains a moderate mean shift, while the third group of observations contains a severe mean shift. The shift types are described in the paper from Capezza et al. (2023).

References

Centofanti F, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2021) Functional Regression Control Chart. *Technometrics*, 63(3):281–294. doi:10.1080/00401706.2020.1753581

Capezza, C., Centofanti, F., Lepore, A., Menafoglio, A., Palumbo, B., & Vantini, S. (2023). funcharts: Control charts for multivariate functional data in R. *Journal of Quality Technology*, 55(5), 566–583. doi:10.1080/00224065.2023.2219012

simulate_data_fmrcc *Simulate Data for Functional Mixture Regression Control Chart (FMRCC)*

Description

#' @description Generates synthetic in-control and out-of-control functional data for testing the Functional Mixture Regression Control Chart (FMRCC) framework. The function simulates a functional response Y influenced by a functional covariate X through a mixture of functional linear models (FLMs) with three distinct regression structures, as described in Section 3.1 of Capezza et al. (2025).

Usage

```
simulate_data_fmrcc(
  n_obs = 3000,
  mixing_prop = c(1/3, 1/3, 1/3),
  len_grid = 500,
  SNR = 4,
  shift_coef = c(0, 0, 0, 0),
  severity = 0,
  ncompx = 20,
  delta_1,
  delta_2,
  measurement_noise_sigma = 0,
  fun_noise = "normal",
  df = 3,
  alphasn = 4
)
```

Arguments

n_obs	Integer. Total number of observations to generate. Default is 3000.
mixing_prop	Numeric vector of length 3. Mixing proportions for the three clusters (must sum to 1). Default is c(1/3, 1/3, 1/3).
len_grid	Integer. Number of grid points for evaluating functional data on domain [0,1]. Default is 500.
SNR	Numeric. Signal-to-noise ratio controlling the variance of the error term. Default is 4.
shift_coef	Numeric vector of length 4 or character string. Controls the type and shape of the mean shift: • Numeric vector: Coefficients c(a3, a2, a1, a0) for polynomial shift: $Shift(t) = severity \times (a_3 t^3 + a_2 t^2 + a_1 t + a_0)$ • 'low': Applies a "low" shift pattern based on RSW dynamic resistance curves

- 'high': Applies a "high" shift pattern based on RSW dynamic resistance curves

Default is c(0,0,0,0) (no shift).

`severity` Numeric. Multiplier controlling the magnitude of the shift. Higher values produce larger shifts. This corresponds to the "Severity Level (SL)" in the simulation study. Default is 0 (no shift).

`ncompx` Integer. Number of functional principal components used to generate the functional covariate X. Default is 20.

`delta_1` Numeric in [0,1]. Controls dissimilarity between clusters in regression coefficient functions and functional intercepts (analogous to delta_1 in simulate_data_fmrcc). Required parameter with no default.

`delta_2` Numeric in [0,1]. Controls the relative contribution of functional intercept vs. regression coefficient function (analogous to delta_2 in simulate_data_fmrcc). Required parameter with no default.

`measurement_noise_sigma`
Numeric. Standard deviation of Gaussian measurement error added to both X and Y. Default is 0 (no measurement error).

`fun_noise` Character. Distribution for functional error term. Options:

- `'normal'`: Gaussian errors (default)
- `'t'`: Student's t-distribution errors with df degrees of freedom
- `'skewnormal'`: Skew-normal distribution with skewness parameter alphasn

`df` Numeric. Degrees of freedom for Student's t-distribution when `fun_noise = 't'`. Default is 3.

`alphasn` Numeric. Skewness parameter for skew-normal distribution when `fun_noise = 'skewnormal'`. Default is 4.

Details

The data generation follows Equation (18) in the paper:

$$Y(t) = (1 - \Delta_2)\beta_k^0(t) + \int_S \Delta_2(\beta_k^X(s,t))^T X(s)ds + \varepsilon(t)$$

The three clusters are characterized by:

- Different functional intercepts $\beta_k^0(t)$ (inspired by dynamic resistance curves in RSW processes)
- Different bivariate regression coefficient functions $\beta_k^X(s,t)$
- Functional errors with variance adjusted to achieve the specified SNR

Moreover, when when `severity != 0`, it applies a controlled shift to the functional response Y to simulate out-of-control conditions. The shift types include:

Polynomial shifts: When `shift_coef` is numeric, a polynomial of degree 3 is applied: $Shift(t) = severity \times (a_3t^3 + a_2t^2 + a_1t + a_0)$

Linear shift example: `shift_coef = c(0, 0, 1, 0)` produces a linear shift

Quadratic shift example: `shift_coef = c(0, 1, 0, 0)` produces a quadratic shift

RSW-specific shifts: When `shift_coef = 'low'` or `'high'`, the function applies shifts based on modifications to the dynamic resistance curve (DRC) parameters, simulating realistic fault patterns in resistance spot welding processes. The functional covariate X is generated using functional principal component analysis with standardized magnitudes (scaled by 1/5).

Value

A list containing:

`X` Matrix (`len_grid` $\times$ `n_obs`) of functional covariate observations.

`Y` Matrix (`len_grid` $\times$ `n_obs`) of shifted functional response observations.

`Eps_1`, `Eps_2`, `Eps_3`
Matrices of functional error terms for each cluster.

`beta_matrix_1`, `beta_matrix_2`, `beta_matrix_3`
Matrices (`len_grid` $\times$ `len_grid`) containing the bivariate regression coefficient functions $\beta_k^X(s,t)$ for k=1,2,3.

References

Capezza, C., Centofanti, F., Forcina, D., Lepore, A., and Palumbo, B. (2025). Functional Mixture Regression Control Chart. Annals of Applied Statistics.

Examples

```
# Generate in-control data with three equally-sized clusters, maximum dissimilarity
data <- simulate_data_fmrcc(n_obs = 300, delta_1 = 1, delta_2 = 0.5, severity = 0)

# In-control single cluster case (delta_1 = 0)
data_single <- simulate_data_fmrcc(n_obs = 300, delta_1 = 0, delta_2 = 0.5, severity = 0)

# In-control clusters differing only in regression coefficients
data_beta_only <- simulate_data_fmrcc(n_obs = 300, delta_1 = 1, delta_2 = 1, severity = 0)

# Add measurement noise and use t-distributed errors
data_t_noise <- simulate_data_fmrcc(n_obs = 300, delta_1 = 1, delta_2 = 0.5, severity = 0,
                                    measurement_noise_sigma = 0.01,
                                    fun_noise = 't', df = 5)

# Generate out-of-control data with linear shift
data_oc <- simulate_data_fmrcc(n_obs = 300,
                               shift_coef = c(0, 0, 1, 0),
                               severity = 2,
                               delta_1 = 1,
                               delta_2 = 0.5)

# Generate OC data with quadratic shift
data_quad <- simulate_data_fmrcc(n_obs = 300,
                                 shift_coef = c(0, 1, 0, 0),
                                 severity = 3,
                                 delta_1 = 1,
```

```
                                  delta_2 = 0.5)

# Generate OC data with RSW-specific "low" shift pattern
data_rsw_low <- simulate_data_fmrcc(n_obs = 300,
                                    shift_coef = 'low',
                                    severity = 1.5,
                                    delta_1 = 1,
                                    delta_2 = 0.5)

# Generate OC data with RSW-specific "high" shift pattern
data_rsw_high <- simulate_data_fmrcc(n_obs = 300,
                                     shift_coef = 'high',
                                     severity = 2,
                                     delta_1 = 0.66,
                                     delta_2 = 0.5)
```

`simulate_data_FRTM` *Simulate data for real-time monitoring of univariate functional data*

Description

Generate synthetic data as in the simulation study of Centofanti et al. (2024).

Usage

```
simulate_data_FRTM(
  n_obs = 100,
  scenario = "1",
  shift = "IC",
  alignemnt_level = "M1",
  t_out_type = "0.3",
  severity = 0.5,
  grid = seq(0, 1, length.out = 100)
)
```

Arguments

`n_obs`	Number of curves generated.
`scenario`	A character string indicating the scenario considered. It could be "1", and "2".
`shift`	A character string indicating the shift considered. It could be "IC", in-control data, "OC_h", Shift A (Phase),"OC_x", Shift B (Amplitude) and "OC_xh", Shift C (Amplitude and Phase).
`alignemnt_level`	A character string indicating the alignment level considered. It could be "M1", "M2", and "M3".

t_out_type	If "0.3", change point at the 30% of the process. If "0.6", change point at the 60% of the process.
severity	Severity level.
grid	Grid of evaluation points.

Value

A list containing the following arguments:

x_err: A list containing the discrete observations for each curve.

grid_i: A list of vector of time points where the curves are sampled.

h: A list containing the discrete observations of the warping function for each curve.

template: The discrete observations of the true template function.

grid_template: Time points where the template is sampled.

x_true: A list containing the discrete observations of the amplitude function for each curve.

grid: Grid of evaluation points.

out_control_t: Time of the change point.

References

Centofanti, F., A. Lepore, M. Kulahci, and M. P. Spooner (2024). Real-time monitoring of functional data. *Journal of Quality Technology*, 57(2):135–152, doi:https://doi.org/10.1080/00224065.2024.2430978.

Examples

```
library(funcharts)
data<-simulate_data_FRTM(n_obs=20)
```

---

simulate_data_RoMFCC *Simulate multivariate functional data with casewise and componentwise contamination*

Description

Generate multivariate functional data under different contamination models (casewise, componentwise, or both) for use in simulation studies of the Robust Multivariate Functional Control Chart (RoMFCC) as described in Capezza, Centofanti, Lepore, and Palumbo (2024).

Usage

```
simulate_data_RoMFCC(
  nobs = 1000,
  p = 3,
  p_cellwise = 0,
  p_casewise = 0,
  sd_e = 0.005,
```

```
  sd = 0.002,
  T_exp = 0.6,
  outlier = "no",
  M_outlier_case = 0,
  M_outlier_cell = 0,
  OC = "no",
  M_OC = 0,
  P = 100,
  max_n_cellwise = Inf,
  correlation = "decreasing",
  k = 1,
  which_OC = 5
)
```

Arguments

nobs	Integer. Number of observations to simulate (default 1000).
p	Integer. Number of functional components (variables) (default 3).
p_cellwise	Numeric in $[0, 1]$. Probability of cellwise contamination (componentwise outliers) for each component (default 0).
p_casewise	Numeric in $[0, 1]$. Probability of casewise contamination (entire observation contaminated) (default 0).
sd_e	Numeric. Standard deviation of the additive measurement noise (default 0.005).
sd	Numeric. Standard deviation scaling of the functional part (default 0.002).
T_exp	Numeric in $(0, 1)$. Expansion parameter for phase outliers (default 0.6).
outlier	Character. Type of contamination in Phase I sample: `"no"` (no contamination), `"outlier_M"` (mean shift), `"outlier_E"` (exponential shift), `"outlier_P"` (phase shift). Default `"no"`.
M_outlier_case	Numeric. Magnitude of casewise outlier contamination (default 0).
M_outlier_cell	Numeric. Magnitude of cellwise outlier contamination (default 0).
OC	Character. Out-of-control model in Phase II data: `"no"` (in control), `"OC_M"` (mean shift), `"OC_E"` (exponential shift), `"OC_P"` (phase shift). Default `"no"`.
M_OC	Numeric. Magnitude of out-of-control shift (default 0).
P	Integer. Number of grid points in each functional profile (default 100).
max_n_cellwise	Integer. Maximum number of components per observation allowed to be cellwise contaminated (default `Inf`).
correlation	Character. Correlation structure among components: typically `"decreasing"` (default).
k	Integer. Correlation parameter (default 1).
which_OC	Integer vector. Indices of components subject to out-of-control shifts in Phase II (default 5).

Details

The generated data mimic dynamic resistance curves (DRCs) in resistance spot welding processes and allow for controlled introduction of casewise and/or componentwise outliers, as in the Monte Carlo study presented in the RoMFCC paper.

The function generates `nobs` realizations of a `p`-variate functional quality characteristic observed on an equally spaced grid of size `P`. The underlying process is simulated through a Karhunen–Loève expansion with eigenfunctions and eigenvalues derived from a specified correlation structure. Outliers can be introduced at cellwise level (single components), casewise level (entire observation), or both, using probability parameters `p_cellwise` and `p_casewise` and magnitudes `M_outlier_cell`, `M_outlier_case`. Out-of-control shifts in Phase II can be introduced via the `OC` argument.

This setup mirrors the simulation design in Section 4 of Capezza et al. (2024) where RoMFCC was benchmarked against competing control charts under various contamination scenarios.

Value

A list with two elements:

X_mat_list A list of length `p`, each element a matrix of dimension `nobs` × `P` with the simulated functional observations.

ind_out A list of length `p`, each element containing the indices of observations contaminated in that component.

Capezza, C., Centofanti, F., Lepore, A., Palumbo, B. (2024) Robust Multivariate Functional Control Chart. *Technometrics*, 66(4):531–547, doi:10.1080/00401706.2024.2327346.

Examples

```
# Simulate uncontaminated data (Phase I)
sim <- simulate_data_RoMFCC(nobs = 200, p = 3, outlier = "no", OC = "no")
str(sim$X_mat_list)

# Simulate with componentwise outliers in Phase I
sim2 <- simulate_data_RoMFCC(nobs = 200, p = 3,
                             p_cellwise = 0.05, M_outlier_cell = 0.03,
                             outlier = "outlier_E")

# Simulate Phase II with a mean shift in one component
sim3 <- simulate_data_RoMFCC(nobs = 200, p = 3,
                             OC = "OC_M", M_OC = 0.04, which_OC = 2)
```

`simulate_mfd` *Simulate a data set for funcharts*

Description

Function used to simulate a data set to illustrate the use of `funcharts`. By default, it creates a data set with three functional covariates, a functional response generated as a function of the three functional covariates through a function-on-function linear model, and a scalar response generated as a function of the three functional covariates through a scalar-on-function linear model. This function covers the simulation study in Centofanti et al. (2021) for the function-on-function case and also simulates data in a similar way for the scalar response case. It is possible to select the number of functional covariates, the correlation function type for each functional covariate and the functional response, moreover it is possible to provide manually the mean and variance functions for both functional covariates and the response. In the default case, the function generates in-control data. Additional arguments can be used to generate additional data that are out of control, with mean shifts according to the scenarios proposed by Centofanti et al. (2021). Each simulated observation of a functional variable consists of a vector of discrete points equally spaced between 0 and 1 (by default 150 points), generated with noise.

Usage

```
simulate_mfd(
  nobs = 1000,
  p = 3,
  R2 = 0.97,
  shift_type_y = "0",
  shift_type_x = c("0", "0", "0"),
  correlation_type_y = "Bessel",
  correlation_type_x = c("Bessel", "Gaussian", "Exponential"),
  d_y = 0,
  d_y_scalar = 0,
  d_x = c(0, 0, 0),
  n_comp_y = 10,
  n_comp_x = 50,
  P = 500,
  ngrid = 150,
  save_beta = FALSE,
  mean_y = NULL,
  mean_x = NULL,
  variance_y = NULL,
  variance_x = NULL,
  sd_y = 0.3,
  sd_x = c(0.3, 0.05, 0.3),
  seed
)
```

Arguments

`nobs`	The number of observation to simulate
`p`	The number of functional covariates to simulate. Default value is 3.
`R2`	The desired coefficient of determination in the regression in both the scalar and functional response cases, Default is 0.97.

`shift_type_y` The shift type for the functional response. There are five possibilities: "0" if there is no shift, "A", "B", "C" or "D" for the corresponding shift types shown in Centofanti et al. (2021). Default is "0".

`shift_type_x` A list of length `p`, indicating, for each functional covariate, the shift type. For each element of the list, there are five possibilities: "0" if there is no shift, "A", "B", "C" or "D" for the corresponding shift types shown in Centofanti et al. (2021). By default, shift is not applied to any functional covariate.

`correlation_type_y`
A character vector indicating the type of correlation function for the functional response. See Centofanti et al. (2021) for more details. Three possible values are available, namely `"Bessel"`, `"Gaussian"` and `"Exponential"`. Default value is `"Bessel"`.

`correlation_type_x`
A list of `p` character vectors indicating the type of correlation function for each functional covariate. See Centofanti et al. (2021) for more details. For each element of the list, three possible values are available, namely `"Bessel"`, `"Gaussian"` and `"Exponential"`. Default value is `c("Bessel", "Gaussian", "Exponential")`.

`d_y` A number indicating the severity of the shift type for the functional response. Default is 0.

`d_y_scalar` A number indicating the severity of the shift type for the scalar response. Default is 0.

`d_x` A list of `p` numbers, each indicating the severity of the shift type for the corresponding functional covariate. By default, the severity is set to zero for all functional covariates.

`n_comp_y` A positive integer number indicating how many principal components obtained after the eigendecomposiiton of the covariance function of the functional response variable to retain. Default value is 10.

`n_comp_x` A positive integer number indicating how many principal components obtained after the eigendecomposiiton of the covariance function of the multivariate functional covariates variable to retain. Default value is 50.

`P` A positive integer number indicating the number of equally spaced grid points over which the covariance functions are discretized. Default value is 500.

`ngrid` A positive integer number indicating the number of equally spaced grid points between zero and one over which all functional observations are discretized before adding noise. Default value is 150.

`save_beta` If TRUE, the true regression coefficients of both the function-on-function and the scalar-on-function models are saved. Default is FALSE.

`mean_y` The mean function of the functional response can be set manually through this argument. If not NULL, it must be a vector of length equal to `ngrid`, providing the values of the mean function of the functional response discretized on `seq(0,1,l=ngrid)`. If NULL, the mean function is generated as done in the simulation study of Centofanti et al. (2021). Default is NULL.

`mean_x` The mean function of the functional covariates can be set manually through this argument. If not NULL, it must be a list of vectors, each with length equal to `ngrid`, providing the values of the mean function of each functional covariate

discretized on `seq(0,1,l=ngrid)`. If NULL, the mean function is generated as done in the simulation study of Centofanti et al. (2021). Default is NULL.

`variance_y` The variance function of the functional response can be set manually through this argument. If not NULL, it must be a vector of length equal to `ngrid`, providing the values of the variance function of the functional response discretized on `seq(0,1,l=ngrid)`. If NULL, the variance function is generated as done in the simulation study of Centofanti et al. (2021). Default is NULL.

`variance_x` The variance function of the functional covariates can be set manually through this argument. If not NULL, it must be a list of vectors, each with length equal to `ngrid`, providing the values of the variance function of each functional covariate discretized on `seq(0,1,l=ngrid)`. If NULL, the variance function is generated as done in the simulation study of Centofanti et al. (2021). Default is NULL.

`sd_y` A positive number indicating the standard deviation of the generated noise with which the functional response discretized values are observed. Default value is 0.3

`sd_x` A vector of `p` positive numbers indicating the standard deviation of the generated noise with which the functional covariates discretized values are observed. Default value is `c(0.3, 0.05, 0.3)`.

`seed` Deprecated: use `set.seed()` before calling the function for reproducibility.

Value

A list with the following elements:

- `X_list` is a list of `p` matrices, each with dimension `nobsxngrid`, containing the simulated observations of the multivariate functional covariate
- `Y` is a `nobsxngrid` matrix with the simulated observations of the functional response
- `y_scalar` is a vector of length `nobs` with the simulated observations of the scalar response
- `beta_fof`, if `save_beta = TRUE`, is a list of `p` matrices, each with dimension `PxP` with the discretized functional coefficients of the function-on-function regression
- `beta_sof`, if `save_beta = TRUE`, is a list of `p` vectors, each with length `P`, with the discretized functional coefficients of the scalar-on-function regression

References

Centofanti F, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2021) Functional Regression Control Chart. *Technometrics*, 63(3):281–294. doi:10.1080/00401706.2020.1753581

`sof_pc` *Scalar-on-function linear regression based on principal components*

Description

Scalar-on-function linear regression based on principal components. This function performs multivariate functional principal component analysis (MFPCA) to extract multivariate functional principal components from the multivariate functional covariates, then it builds a linear regression model of a scalar response variable on the covariate scores. Functional covariates are standardized before the regression. See Capezza et al. (2020) for additional details.

Usage

```
sof_pc(
  y,
  mfdobj_x,
  tot_variance_explained = 0.9,
  selection = "variance",
  single_min_variance_explained = 0,
  components = NULL
)
```

Arguments

`y`	A numeric vector containing the observations of the scalar response variable.
`mfdobj_x`	A multivariate functional data object of class mfd denoting the functional covariates.
`tot_variance_explained`	The minimum fraction of variance that has to be explained by the set of multivariate functional principal components retained into the MFPCA model fitted on the functional covariates. Default is 0.9.
`selection`	A character value with one of three possible values: if "variance", the first M multivariate functional principal components are retained into the MFPCA model such that together they explain a fraction of variance greater than `tot_variance_explained`, if "PRESS", each j-th functional principal component is retained into the MFPCA model if, by adding it to the set of the first j-1 functional principal components, then the predicted residual error sum of squares (PRESS) statistic decreases, and at the same time the fraction of variance explained by that single component is greater than `single_min_variance_explained`. This criterion is used in Capezza et al. (2020). if "gcv", the criterion is equal as in the previous "PRESS" case, but the "PRESS" statistic is substituted by the generalized cross-validation (GCV) score. Default value is "variance".
`single_min_variance_explained`	The minimum fraction of variance that has to be explained by each multivariate functional principal component into the MFPCA model fitted on the functional covariates such that it is retained into the MFPCA model. Default is 0.
`components`	A vector of integers with the components over which to project the functional covariates. If this is not NULL, the criteria to select components are ignored. If NULL, components are selected according to the criterion defined by `selection`. Default is NULL.

Value

a list containing the following arguments:

- `mod`: an object of class `lm` that is a linear regression model where the scalar response variable is `y` and the covariates are the MFPCA scores of the functional covariates,
- `mod$coefficients` contains the matrix of coefficients of the functional regression basis functions,
- `pca`: an object of class `pca_mfd` obtained by doing MFPCA on the functional covariates,
- `beta_fd`: an object of class `mfd` object containing the functional regression coefficient $\beta(t)$ estimated with the scalar-on-function linear regression model,
- `components`: a vector of integers with the components selected in the `pca` model,
- `selection`: the same as the provided argument
- `single_min_variance_explained`: the same as the provided argument
- `tot_variance_explained`: the same as the provided argument
- `gcv`: a vector whose j-th element is the GCV score obtained when retaining the first j components in the MFPCA model.
- `PRESS`: a vector whose j-th element is the PRESS statistic obtained when retaining the first j components in the MFPCA model.

References

Capezza C, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2020) Control charts for monitoring ship operating conditions and CO2 emissions based on scalar-on-function regression. *Applied Stochastic Models in Business and Industry*, 36(3):477–500. doi:10.1002/asmb.2507

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates], lambda = 1e-2)
y <- rowMeans(air$NO2)
mod <- sof_pc(y, mfdobj_x)
```

---

sof_pc_real_time

Get a list of scalar-on-function linear regression models estimated on functional data each evolving up to an intermediate domain point.

Description

This function produces a list of objects, each of them contains the result of applying `sof_pc` to a scalar response variable and multivariate functional covariates evolved up to an intermediate domain point. See Capezza et al. (2020) for additional details on real-time monitoring.

Usage

```
sof_pc_real_time(
  y,
  mfd_real_time_list,
  single_min_variance_explained = 0,
  tot_variance_explained = 0.9,
  selection = "PRESS",
  components = NULL,
  ncores = 1
)
```

Arguments

y	A numeric vector containing the observations of the scalar response variable.
mfd_real_time_list	A list created using get_mfd_df_real_time or get_mfd_list_real_time, denoting a list of functional data objects, each evolving up to an intermediate domain point, with observations of the multivariate functional covariates.
single_min_variance_explained	See sof_pc.
tot_variance_explained	See sof_pc.
selection	See sof_pc.
components	See sof_pc.
ncores	If you want parallelization, give the number of cores/threads to be used when creating objects separately for different instants.

Value

A list of lists each produced by sof_pc, corresponding to a given instant.

References

Capezza C, Lepore A, Menafoglio A, Palumbo B, Vantini S. (2020) Control charts for monitoring ship operating conditions and CO2 emissions based on scalar-on-function regression. *Applied Stochastic Models in Business and Industry*, 36(3):477–500. doi:10.1002/asmb.2507

See Also

sof_pc, get_mfd_df_real_time, get_mfd_list_real_time

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[1:10, , drop = FALSE])
mfdobj_list <- get_mfd_list_real_time(air[c("CO", "temperature")],
                                      n_basis = 15,
```

```
                                    lambda = 1e-2,
                                    k_seq = c(0.5, 0.75, 1))
y <- rowMeans(air$NO2)
mod_list <- sof_pc_real_time(y, mfdobj_list)
```

`tensor_product_mfd` *Tensor product of two Multivariate Functional Data objects*

Description

This function returns the tensor product of two Multivariate Functional Data objects. Each object must contain only one replication.

Usage

```
tensor_product_mfd(mfdobj1, mfdobj2 = NULL)
```

Arguments

`mfdobj1`	A multivariate functional data object, of class `mfd`, having only one functional observation.
`mfdobj2`	A multivariate functional data object, of class `mfd`, having only one functional observation. If NULL, it is set equal to `mfdobj1`. Default is NULL.

Value

An object of class `bifd`. If we denote with x(s)=(x_1(s),. . . ,x_p(s)) the vector of p functions represented by `mfdobj1` and with y(t)=(y_1(t),. . . ,y_q(t)) the vector of q functions represented by `mfdobj2`, the output is the vector of pq bivariate functions

f(s,t)=(x_1(s)y_1(t),. . . ,x_1(s)y_q(t), . . . ,x_p(s)y_1(t),. . . ,x_p(s)y_q(t)).

Examples

```
library(funcharts)
mfdobj1 <- data_sim_mfd(nobs = 1, nvar = 3)
mfdobj2 <- data_sim_mfd(nobs = 1, nvar = 2)
tensor_product_mfd(mfdobj1)
tensor_product_mfd(mfdobj1, mfdobj2)
```

`times_mfd` *Pointwise product of multivariate functional data (and scalar multiplication)*

Description

Computes the elementwise (pointwise) product of two objects of class `mfd`, returning an `mfd` on the same basis. If one object contains a single replication (one observation) and the other contains multiple, the single replication is recycled across observations before multiplication.

Usage

```
times_mfd(mfdobj1, mfdobj2)

## S3 method for class 'mfd'
mfdobj1 * mfdobj2
```

Arguments

`mfdobj1`, `mfdobj2`
: Objects of class `mfd` defined on the same basis.

Details

Alternatively, it also compute the product of an `mfd` object with a numeric scalar.

Let coefficient arrays have dimensions $(nbasis, nobs, nvar)$. The function:

- requires both inputs to be `mfd` objects;
- requires identical basis systems (checked with `identical()`);
- requires the same number of variables;
- for observations: if both $nobs_1$ and $nobs_2$ are greater than one, they must be equal; otherwise, the object with $nobs = 1$ is replicated to match the other.

Internally, coefficient arrays are converted to `fd` objects and multiplied via `times.fd`, with `basisobj` set to the common basis so that the result is re-expanded on the same basis.

Value

An object of class `mfd` whose coefficients are the pointwise product of the inputs (with recycling if needed). The basis is the common input basis. The `fdnames` are inherited from the input that supplies the observation indexing after any replication.

See Also

`plus_mfd`, `minus_mfd`, `nobs`, `nvar`, `nbasis`, `times.fd`, `mfd`

Examples

```
# Assuming mfdobj_a and mfdobj_b are 'mfd' objects on the same basis:
# mfdobj_a * mfdobj_b   # elementwise product
# 2 * mfdobj_a          # scalar multiplication
# mfdobj_a * 0.5        # scalar multiplication
```

`which_ooc` *Get the index of the out of control observations from control charts*

Description

This function returns a list for each control chart and returns the id of all observations that are out of control in that control chart.

Usage

```
which_ooc(cclist)
```

Arguments

`cclist`	A `data.frame` produced by `control_charts_sof_pc`.

Value

A list of as many `data.frame` objects as the control charts in `cclist`. Each data frame has two columns, the `n` contains an index number giving the observation in the phase II data set, i.e. 1 for the first observation, 2 for the second, and so on, while the `id` column contains the id of the observation, which can be general and depends on the specific data set.

Examples

```
library(funcharts)
data("air")
air <- lapply(air, function(x) x[201:300, , drop = FALSE])
fun_covariates <- c("CO", "temperature")
mfdobj_x <- get_mfd_list(air[fun_covariates],
                         n_basis = 15,
                         lambda = 1e-2)
y <- rowMeans(air$NO2)
y1 <- y[1:60]
y_tuning <- y[61:90]
y2 <- y[91:100]
mfdobj_x1 <- mfdobj_x[1:60]
mfdobj_x_tuning <- mfdobj_x[61:90]
mfdobj_x2 <- mfdobj_x[91:100]
mod <- sof_pc(y1, mfdobj_x1)
cclist <- regr_cc_sof(object = mod,
                      y_new = y2,
```

```
                        mfdobj_x_new = mfdobj_x2,
                        y_tuning = y_tuning,
                        mfdobj_x_tuning = mfdobj_x_tuning,
                        include_covariates = TRUE)
    which_ooc(cclist)
```

Index

∗ **datasets**
air, 6
∗.mfd *(*times_mfd*)*, 135
+.mfd *(*plus_mfd*)*, 90
-.mfd *(*minus_mfd*)*, 67
[.mfd, 4

abline, *5, 6*
abline_mfd, 5, *61*
air, 6
AMFCC_PhaseI, 7
AMFCC_PhaseII, 9
AMFEWMA_PhaseI, 12, *15*, *105, 106*
AMFEWMA_PhaseII, 15, *105*

cbind_mfd, 17
cont_plot, 18
control_charts_pca, *18*, 19, *22, 23*, *26*, *56*, *85*, *89*, *97*, *101*
control_charts_pca_mfd_real_time, 22
control_charts_sof_pc, *18*, 24, *26, 27*, *56*, *85*, *89*, *136*
control_charts_sof_pc_real_time, 26, *86*
cor_mfd, 28
cov_mfd, *28*, 29

data_sim_mfd, 30

estimate_mixture, 31

fd, *64*, *135*
FMRCC_PhaseI, 32, *35*
FMRCC_PhaseII, 34, *34*
fof_pc, 36, *39, 40*
fof_pc_real_time, 39, *98, 99*
FPCA, *32–34*
FRTM_PhaseI, 40, *42*, *75, 76*
FRTM_PhaseII, 43
functional_filter, 44, *105*, *109*

geom_line, *88*
get_mfd_array, 46, *47, 48*, *54*
get_mfd_array_real_time, 47
get_mfd_df, *47*, 48, *51, 52*, *56*
get_mfd_df_real_time, *22, 23*, *27*, *39, 40*, 51, *78*, *99*, *103*, *133*
get_mfd_fd, 52
get_mfd_list, *46, 47*, *50*, 53, *54–56*
get_mfd_list_real_time, *40*, 55, *133*
get_ooc, 56
get_outliers_mfd, 57
get_sof_pc_outliers, 58

inprod_mfd, 59
inprod_mfd_diag, 60
is.mfd, 61

lines.mfd, 61
lines_mfd, 62

mean.fd, *119*
mean.mfd, 63
mfd, *54*, *63*, 64, *67*, *91*, *135*
mFPCA, 65
minus_mfd, 67, *135*
mixregfit_multivariate, 68

nbasis, *67*, 69, *91*, *135*
nobs, *67*, *91*, *135*
nobs.mfd, 69
norm.mfd, 70
nvar, *67*, 70, *91*, *135*

OEBFDTW, 71

par.FDTW, 73
par.mFPCA, 75
par.rtr, 76
pca.fd, *77*
pca_mfd, *22*, *32*, 77, *78*, *92, 93*
pca_mfd_real_time, *22, 23*, 78
plot.AMFCC_PhaseI, 79

plot.AMFCC_PhaseII *(*plot.AMFCC_PhaseI*)*, 79
plot.FRTM_PhaseI, 80
plot.FRTM_PhaseII *(*plot.FRTM_PhaseI*)*, 80
plot.mfd, *5, 6*, *61*, *63*, 82
plot.mFPCA, 82
plot_bifd, 83
plot_bootstrap_sof_pc, 84
plot_control_charts, 85
plot_control_charts_real_time, 86
plot_mfd, *62*, 87
plot_mon, 88
plot_pca_mfd, 90
plus_mfd, *67*, 90, *135*
predict, *92*
predict.pca_mfd, 92
predict_fof_pc, 93
predict_sof_pc, 94

rbind_mfd, 95
regr_cc_fof, *18*, *21*, *56*, *85*, *89*, 96, *98, 99*
regr_cc_fof_real_time, *86*, 98
regr_cc_sof, *18*, *24*, *26*, *56*, *85*, *89*, 100, *100*, *102–104*
regr_cc_sof_real_time, *26*, 102
RoAMFEWMA_PhaseI, 104, *107*
RoAMFEWMA_PhaseII, 107
RoMFCC_PhaseI, 109, *113*
RoMFCC_PhaseI_casewise, *105, 106*, 111, *114*
RoMFCC_PhaseII, 113
RoMFCC_PhaseII_casewise, *105, 106*, 114
RoMFDI, *105, 106*, *109*, 116
rpca_mfd, *45*, *110, 111*, *116*, 117

scale, *119*
scale_mfd, *28*, *77*, *93*, 119
sd.fd, *119*
sim_funcharts, 120
simulate_data_fmrcc, 121
simulate_data_FRTM, 124
simulate_data_RoMFCC, 125
simulate_mfd, *120*, 127
sof_pc, *84*, 130, *132, 133*
sof_pc_real_time, *27*, *103, 104*, 132

tensor_product_mfd, 134
times.fd, *135*
times_mfd, 135

which_ooc, 136