Package 'ReMFPCA'

May 7, 2026

Type Package

Title Regularized Multivariate Functional Principal Component Analysis

Version 2.0.0

Maintainer Hossein Haghbin <haghbin@pgu.ac.ir>

Description Methods and tools for implementing regularized multivariate functional principal component analysis ('ReMFPCA') for multivariate functional data whose variables might be observed over different dimensional domains. 'ReMFPCA' is an object-oriented interface leveraging the extensibility and scalability of R6. It employs a parameter vector to control the smoothness of each functional variable. By incorporating smoothness constraints as penalty terms within a regularized optimization framework, 'ReMFPCA' generates smooth multivariate functional principal components, offering a concise and interpretable representation of the data. For detailed information on the methods and techniques used in 'ReMFPCA', please refer to Haghbin et al. (2023) <doi:10.48550/arXiv.2306.13980>.

URL https://github.com/haghbinh/ReMFPCA

License GPL (>= 2)

Encoding UTF-8

LazyData TRUE

Imports fda, expm, Matrix

RoxygenNote 7.3.2

Depends R (>= 4.0), R6

NeedsCompilation no

Author Hossein Haghbin [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8416-2354>),
Yue Zhao [aut] (ORCID: <https://orcid.org/0009-0000-4561-9163>),
Mehdi Maadooliat [aut] (ORCID: <https://orcid.org/0000-0002-5408-2676>)

Repository CRAN

Date/Publication 2025-04-12 08:50:02 UTC

Contents

`*.mfd` *Scalar multiplication of an ‘mfd‘ object*

Description

Scalar multiplication of an ‘mfd‘ object. One object must be an ‘mfd‘, and the other one a scalar

Usage

```
## S3 method for class 'mfd'
obj1 * obj2
```

Arguments

obj1	An ‘mfd‘ object or an scalar
obj2	An ‘mfd‘ object or an scalar

Value

An ‘mfd‘ object

See Also

basismfd, mfd

---

*.mvmfd *Multiplication of an ‘mvmfd‘ object with a scalar*

Description

Multiplication of an ‘mvmfd‘ object with a scalar

Usage

```
## S3 method for class 'mvmfd'
obj1 * obj2
```

Arguments

obj1	An ‘mvmfd‘ object or a scalar
obj2	An ‘mvmfd‘ object or a scalar

Value

An ‘mvmfd‘ object

See Also

mvmfd,mvbasismfd

+.mfd *Add two 'mfd' objects*

Description

Add two 'mfd' objects

Usage

```
## S3 method for class 'mfd'
obj1 + obj2 = NULL
```

Arguments

obj1	An 'mfd' object
obj2	An 'mfd' object or a scalar

Value

The sum of the two 'mfd' objects

See Also

basismfd, mfd

+.mvmfd *Addition of two 'mvmfd' objects*

Description

Addition of two 'mvmfd' objects

Usage

```
## S3 method for class 'mvmfd'
obj1 + obj2 = NULL
```

Arguments

obj1	An 'mvmfd' object
obj2	An optional 'mvmfd' object

Value

An 'mvmfd' object

See Also

mvmfd,mvbasismfd

`-.mfd` *Subtract two 'mfd' objects*

Description

Subtract two 'mfd' objects

Usage

```
## S3 method for class 'mfd'
obj1 - obj2 = NULL
```

Arguments

`obj1`	An 'mfd' object
`obj2`	An 'mfd' object or a scalar

Value

The difference between the two 'mfd' objects

See Also

basismfd, mfd

`-.mvmfd` *Subtraction of two 'mvmfd' objects*

Description

Subtraction of two 'mvmfd' objects

Usage

```
## S3 method for class 'mvmfd'
obj1 - obj2 = NULL
```

Arguments

`obj1`	An 'mvmfd' object
`obj2`	An optional 'mvmfd' object

Value

An ‘mvmfd‘ object

See Also

mvmfd,mvbasismfd

---

basismfd *Define a Set of Multidimensional Functional Basis*

Description

The ‘basismfd‘ class represents a set of multidimensional basis functions. This class utilizes basis objects from the ‘fda‘ package, such as B-splines and Fourier bases.

Constructor for ‘basismfd‘ objects (same as Basismfd(...))

Usage

```
Basismfd(...)

Basismfd(...)
```

Arguments

...	A list of ‘basisfd‘ objects

Active bindings

`basis` A list of basis objects from the ‘fda‘ package.

`dimSupp` The dimension of the support domain of the ‘basismfd‘ object.

`supp` The matrix representing the ranges of the dimensions.

`gram` The Gram matrix.

`nbasis` A numeric vector containing the number of bases.

Methods

Public methods:

- basismfd$new()
- basismfd$eval()
- basismfd$print()
- basismfd$clone()

Method `new()`**:** The constructor function for objects of the class ‘basismfd‘ (same as Basismfd(...))

Usage:

```
basismfd$new(...)
```

Arguments:

`...` A list of ‘basisfd‘ objects

Method `eval()`**:** Evaluate the ‘basismfd‘ object at given argument values

Usage:

```
basismfd$eval(evalarg)
```

Arguments:

`evalarg` A list of numeric vectors of argument values at which the ‘basismfd‘ is to be evaluated

Returns: A list of evaluated values

Method `print()`**:** Print method for ‘basismfd‘ objects

Usage:

```
basismfd$print(...)
```

Arguments:

`...` Additional arguments to be passed to ‘print‘ Getter and setter for ‘basis‘ field Getter and setter for ‘dimSupp‘ field Getter and setter for ‘nbasis‘ field Getter and setter for ‘supp‘ field Getter and setter for ‘gram‘ field

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
basismfd$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
require(fda)
bs1 <- create.fourier.basis(c(0, 2 * pi), 5)
bs2 <- create.bspline.basis(c(0, 1), 7)
bs3 <- create.exponential.basis(c(0, 2), 3)
# 1-D Basis ######## (similar to the fd features)
mdbs1 <- Basismfd(bs1)
mdbs1$basis
mdbs1$dimSupp
mdbs1$nbasis
mdbs1$supp
mdbs1$gram
mdbs1$eval(1:7 / 10)
image(as.matrix(mdbs1$gram))

####### 2-D Basis ######## (fd cannot handle this)
mdbs2 <- Basismfd(bs1, bs2)
mdbs2$basis
mdbs2$dimSupp
mdbs2$nbasis
mdbs2$supp
```

```
dim(mdbs2$gram)
arg_mdbs <- list(1:10, 1:9 / 10)
mdbs2$eval(arg_mdbs)
image(as.matrix(mdbs2$gram))
```

`bimfdplot` *Bivariate plot for 'mvmfd' objects*

Description

Bivariate plot for 'mvmfd' objects

Usage

```
bimfdplot(mvmfd_obj, type = "l", lty = 1, xlab = "", ylab = "", main = "", ...)
```

Arguments

`mvmfd_obj`	An 'mvmfd' object
`type`	Type of plot ('l' for lines, 'p' for points, etc.)
`lty`	Line type
`xlab`	Label for the x-axis
`ylab`	Label for the y-axis
`main`	Main title
`...`	Additional arguments for the matplot function

See Also

`mvmfd`, `mvbasismfd`

`electrical_power_data` *Electrical Power Dataset*

Description

This dataset contains measurements of voltage and electrical power consumption recorded from a household between December 2006 and November 2010.

Format

A bivariate functional data object of class 'mvmfd' with the following fields:

voltage Voltage measurements in volts.

power Calculated power consumption in watts.

Source

The dataset was collected from https://weather.com.

Examples

```
## Not run:
# Load the Electrical Power Dataset
data("electrical_power_data")
head(electrical_power_data)

## End(Not run)
```

`inprod_mfd` *Compute the inner product between two objects of class 'mfd'*

Description

Compute the inner product between two objects of class 'mfd'

Usage

```
inprod_mfd(mfd_obj1, mfd_obj2)
```

Arguments

`mfd_obj1`	An 'mfd' object
`mfd_obj2`	An 'mfd' object

Value

The inner products matrix between the two 'mfd' objects

See Also

`basismfd`, `mfd`

inprod_mvmfd *Compute the inner product between two objects of class 'mvmfd'*

Description

Compute the inner product between two objects of class 'mvmfd'

Usage

```
inprod_mvmfd(mvmfd_obj1, mvmfd_obj2)
```

Arguments

mvmfd_obj1	An 'mvmfd' object
mvmfd_obj2	An 'mvmfd' object

Value

The inner products matrix between the two 'mvmfd' objects

See Also

mvmfd,mvbasismfd

is.basismfd *Check if an object is of class 'basismfd'*

Description

Check if an object is of class 'basismfd'

Usage

```
is.basismfd(fdobj)
```

Arguments

fdobj	The object to check.

Value

TRUE if the object is of class 'basismfd', FALSE otherwise.

See Also

is.mvbasismfd, is.mfd, is.mvmfd

is.mfd *Check if an object is of class 'mfd'*

Description

Check if an object is of class 'mfd'

Usage

```
is.mfd(fdobj)
```

Arguments

fdobj	The object to check.

Value

TRUE if the object is of class 'mfd', FALSE otherwise.

See Also

is.mvbasismfd, is.basismfd, is.mvmfd

is.mvbasismfd *Check if an object is of class 'mvbasismfd'*

Description

Check if an object is of class 'mvbasismfd'

Usage

```
is.mvbasismfd(fdobj)
```

Arguments

fdobj	The object to check.

Value

TRUE if the object is of class 'mvbasismfd', FALSE otherwise.

See Also

is.basismfd, is.mfd, is.mvmfd

is.mvmfd *Check if an object is of class 'mvmfd'*

Description

Check if an object is of class 'mvmfd'

Usage

```
is.mvmfd(fdobj)
```

Arguments

fdobj	The object to check.

Value

TRUE if the object is of class 'mvmfd', FALSE otherwise.

See Also

is.mvbasismfd, is.mfd, is.basismfd

length *Length of an object of classes 'mfd'or 'mvmfd'.*

Description

Length of an object of an object of classes 'mfd' or 'mvmfd'.

Usage

```
length(x, ...)
```

Arguments

x	An object of classes 'mfd' or 'mvmfd'.
...	all 'length' function arguments.

mean *mean of an object of classes 'mfd'or 'mvmfd'.*

Description

mean of an object of classes 'mfd'or 'mvmfd'.

Usage

```
mean(x, ...)
```

Arguments

x	An object of classes 'mfd' or 'mvmfd'.
...	all 'mean' function arguments.

Value

An object of class 'mfd'

mfd *Define a Set of Multidimensional Functional Data objects*

Description

The 'mfd' class represents a set of multidimensional functional data with 'basismfd' object. Functional data objects are constructed by specifying a set of basis functions and a set of coefficients defining a linear combination of these basis functions.

Constructor for 'mfd' objects (same as Mfd(...))

Usage

```
Mfd(argval = NULL, X, mdbs, method = "data")
```

Arguments

argval	A list of numeric vectors of argument values at which the 'mfd' object is to be evaluated
X	A numeric matrix corresponds to basis expansion coefficients if 'method="coefs"' and discrete observations if 'method="data"'.
mdbs	a basismfd object
method	determine the 'X' matrix type as "coefs" and "data".

Active bindings

`basis` an object of the class ‘basismfd‘.

`coefs` a matrix of the coefficients.

`nobs` number of the observation

Methods

Public methods:

- `mfd$new()`
- `mfd$eval()`
- `mfd$print()`
- `mfd$clone()`

Method `new()`**:** Constructor for ‘mfd‘ objects (same as Mfd(...))

Usage:

```
mfd$new(argval = NULL, X, mdbs, method = "data")
```

Arguments:

`argval` A list of numeric vectors of argument values at which the ‘mfd‘ object is to be evaluated

`X` A numeric matrix corresponds to basis expansion coefficients if ‘method="coefs"‘ and discrete observations if ‘method="data"‘.

`mdbs` a basismfd object

`method` determine the ‘X‘ matrix type as "coefs" and "data".

Method `eval()`**:** Evaluation an ‘mfd‘ object in some arguments.

Usage:

```
mfd$eval(evalarg)
```

Arguments:

`evalarg` a list of numeric vector of argument values at which the `mfd` is to be evaluated.

Returns: A matrix of evaluated values

Method `print()`**:** Print method for ‘mfd‘ objects

Usage:

```
mfd$print(...)
```

Arguments:

`...` Additional arguments to be passed to ‘print‘

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
mfd$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

basismfd

Examples

```
require(fda)
bs1 <- create.fourier.basis(c(0,2*pi),5)
bs2 <- create.bspline.basis(c(0,1),7)
bs3 <- create.exponential.basis(c(0,2),3)

#1-D mfd :_____________________________
argval <- seq(0,2*pi,length.out=100)
nobs <- 10;
X <- outer(sin(argval),seq(0.5,1.5,length.out=nobs))
mdbs1 <- Basismfd(bs1)
mfd1 <- Mfd(X=X, mdbs = mdbs1)
inprod_mfd(mfd1,mfd1)
norm_mfd(mfd1)
mfd0 <- 2.5*mfd1
mfd1-mfd0
mfd1[1:3]

mfd1$eval(argval)
mfd1c <- Mfd(X=mfd1$coefs, mdbs = mdbs1, method = "coefs")
all.equal(c(mfd1$basis,mfd1$coefs,mfd1$nobs),c(mfd1c$basis,mfd1c$coefs,mfd1c$nobs))
length(mfd1)
mean(mfd1)
plot(mfd1)
```

`motion_sense_data` *Motion Sense Dataset: Measurements of user acceleration and pitch attitude collected by smartphones from 24 individuals performing four distinct activities: jogging, walking, sitting, and standing.*

Description

This dataset includes time-series data generated by accelerometer and gyroscope sensors. A total of 24 participants in a range of gender, age, weight, and height performed 4 activities in the same environment and conditions: downstairs, upstairs, walking, jogging, sitting, and standing.

Format

A bivariate functional data object of class ‘mvmfd‘ with the following fields:

user_acceleration Time-series data of user acceleration.

pitch_attitude Time-series data of pitch attitude.

Source

The MotionSense dataset collected by https://github.com/mmalekzadeh.

Examples

```
## Not run:
# Load the Motion Sense Dataset
data("motion_sense_data")

## End(Not run)
```

`mvbasismfd` *Define a Set of Multivariate Multidimensional Functional Basis*

Description

The 'mvbasismfd' a set of multivariate multidimensional basis functions. This class utilizes basis objects 'basismfd'.

Constructor for 'mvbasismfd' objects (same as 'Mvbasismfd')

Usage

```
Mvbasismfd(basis)

Mvbasismfd(basis)

## S3 method for class 'mvbasismfd'
mvbasismfd_obj[i = "index"]
```

Arguments

`basis`	A list of basisfd objects
`mvbasismfd_obj`	An 'mvmfd' object
`i`	An index or indices specifying the subsets to extract for the first dimension

Value

An 'mvbasismfd' object containing the specified subsets

Active bindings

`nvar` number of variables

`basis` A list of 'mvbasisfd' objects

`dimSupp` A sequence of positive integers specifying support domain of the 'mvbasismfd' object.

`nbasis` A list of integers specifying the number of basis functions

`supp` A list of matrices specifying the support of basis functions

`gram` The Gram matrix.

Methods

Public methods:

- mvbasismfd$new()
- mvbasismfd$eval()
- mvbasismfd$clone()

Method `new()`**:** Constructor for ‘mvbasismfd‘ objects (same as Mvbasismfd(...))

Usage:

```
mvbasismfd$new(basis)
```

Arguments:

`basis` A list of ‘basismfd‘ objects

Method `eval()`**:** Evaluate the ‘mvbasismfd‘ object at given argument values

Usage:

```
mvbasismfd$eval(evalarg)
```

Arguments:

`evalarg` A list of numeric vectors of argument values at which the ‘mvbasismfd‘ is to be evaluated

Returns: A list of evaluated values

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
mvbasismfd$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

mvmfd, basismfd

`mvmfd` *Define a Set of Multivariate Multidimensional Functional Data objects*

Description

The ‘mvmfd‘ class represents functional data ...

Constructor for ‘mvmfd‘ objects (same as ‘Mvmfd‘)

Usage

```
Mvmfd(...)
```

Arguments

...	A ‘mfd‘ objects which have separated by comma

Active bindings

`basis` A ‘mvbasismfd‘ object

`coefs` a matrix of the coefficients.

`nobs` number of observation

`nvar` number of variables

Methods

Public methods:

- `mvmfd$new()`
- `mvmfd$eval()`
- `mvmfd$print()`
- `mvmfd$clone()`

Method `new()`**:** Constructor for ‘mvmfd‘ objects (same as ’Mvmfd’)

Usage:

```
mvmfd$new(...)
```

Arguments:

`...` A ‘mfd‘ objects which have separated by comma

Method `eval()`**:** Eval method for ‘mvmfd‘ objects

Usage:

```
mvmfd$eval(evalarg)
```

Arguments:

`evalarg` A list of numeric vectors of argument values at which the ‘mvmfd‘ is to be evaluated.

Returns: A list of evaluated values

Method `print()`**:** Print method for ‘mvmfd‘ objects

Usage:

```
mvmfd$print(...)
```

Arguments:

`...` Additional arguments to be passed to ‘print‘

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
mvmfd$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

mvbasismfd, mfd

Examples

```
require(fda)
bs1 <- create.fourier.basis(c(0, 2 * pi), 5)
bs2 <- create.bspline.basis(c(0, 1), 7)
bs3 <- create.exponential.basis(c(0, 2), 3)
nobs <- 10
argval1 <- seq(0, 2 * pi, length.out = 12)
X1 <- outer(sin(argval1), seq(0.5, 1.5, length.out = nobs))
mdbs1 <- Basismfd(bs1)
mfd1 <- Mfd(argval1, X1, mdbs1)
mdbs2 <- Basismfd(bs1)
argval2 <- argval1
X2 <- outer(cos(argval2), seq(0.2, 1.5, length.out = nobs))
mfd2 <- Mfd(argval2, X2, mdbs1)
mvmfd1 <- Mvmfd(mfd1, mfd2)
mvmfd1[1]
mvmfd1[1, 1]
mvmfd1[1:5, 2]
mvmfd1[, 1]
mvmfd1[1:5, ]
evalarg <- list(argval1, argval2)
mvmfd1$eval(evalarg)
mvmfd1 + mvmfd1
mean(mvmfd1)
inprod_mvmfd(mvmfd1, mvmfd1)
norm_mvmfd(mvmfd1)
plot(mvmfd1)
bimfdplot(mvmfd1)
```

---

norm_mfd *Compute the norm of an object of class 'mfd'*

Description

Compute the norm of an object of class 'mfd'

Usage

```
norm_mfd(mfd_obj)
```

Arguments

mfd_obj	An object of class 'mfd'

Value

The norm vector of the an object of class ‘mfd‘

See Also

basismfd, mfd

---

norm_mvmfd *Compute the norm of an object of class ‘mvmfd‘*

Description

Compute the norm of an object of class ‘mvmfd‘

Usage

```
norm_mvmfd(mvmfd_obj)
```

Arguments

mvmfd_obj	An ‘mvmfd‘ object

Value

The norm vector of the an object of class ‘mvmfd‘

See Also

mvmfd,mvbasismfd

---

pen_fun *Penalty Function*

Description

Calculate the penalty matrix for ‘mvmfd‘ objects.

Usage

```
pen_fun(data, devorder = 2, type)
```

Arguments

data	an object of class ‘mvmfd‘.
devorder	The order of the derivative.
type	The type of penalty. The types "coefpen" and "basispen" is supported.

Value

The penalty matrix.

`plot` *plots an object of classes 'mfd', 'mvmfd' or 'remfpca'*

Description

plot an object of classes 'mfd', 'mvmfd' or 'remfpca'

Usage

```
plot(x, ...)
```

Arguments

`x`	An object of classes 'mfd', 'mvmfd' or 'remfpca'
`...`	all 'plot' function arguments.

`remfpca` *A Class for 'ReMFPCA' objects*

Description

The 'remfpca' class represents regularized functional principal components components.

The 'remfpca' class represents regularized functional principal components ('ReMFPCs') components.

Usage

```
Remfpca(
  mvmfd_obj,
  method = "power",
  ncomp,
  smooth_tuning = NULL,
  sparse_tuning = NULL,
  centerfns = TRUE,
  alpha_orth = FALSE,
  smoothing_type = "basispen",
  sparse_type = "soft",
  K_fold = 30,
  sparse_CV = TRUE,
  smooth_GCV = TRUE
)
```

Arguments

mvmfd_obj	An ‘mvmfd‘ object representing the multivariate functional data.
method	A character string specifying the approach to be used for MFPCA computation. Options are "power" (the default), which uses the power algorithm, or "eigen", which uses the eigen decomposition approach.
ncomp	The number of functional principal components to retain.
smooth_tuning	A list or vector specifying the smoothing regularization parameter(s) for each variable. If NULL, non-smoothing MFPCA is estimated.
sparse_tuning	A list or vector specifying the sparsity regularization parameter(s) for each variable. If NULL, non-sparse MFPCA is estimated.
centerfns	Logical indicating whether to center the functional data before analysis. Default is TRUE.
alpha_orth	Logical indicating whether to perform orthogonalization of the regularization parameters. If ‘method‘ is "power", setting ‘alpha_orth = FALSE‘ (default) uses the sequential power approach, while setting ‘alpha_orth = TRUE‘ uses the joint power approach.
smoothing_type	The type of smoothing penalty to be applied on the estimated functional PCs. The types "basispen" and "coefpen" is supported. Default is "basispen".
sparse_type	The type of sparse penalty to be applied on the estimated functional PCs. The types "soft-threshold", "hard-threshold" and "SCAD" is supported. Default is "soft-threshold".
K_fold	An integer specifying the number of folds in the sparse cross-validation process. Default is 30.
sparse_CV	@param sparse_CV Logical indicating whether cross-validation should be applied to select the optimal sparse tuning parameter in sequential power approach. If ‘sparse_CV = TRUE‘, a series of tuning parameters should be provided as a vector with positive number with max equals to number of subjects. If ‘sparse_CV = FALSE‘, specific tuning parameters are given directly to each principal components. Tuning parameters should be provided as a vector with length equal to ‘ncomp‘. If the dimensions of input tuning parameters are incorrect, it will be converted to a list internally, and a warning will be issued.
smooth_GCV	@param smooth_GCV Logical indicating whether generalized cross-validation should be applied to select the optimal smooth tuning parameter. If ‘smooth_GCV = TRUE‘, a series of tuning parameters should be provided as a list with length equal to the number of variables. If a list with incorrect dimensions is provided, it will be converted to a correct list internally, and a warning will be issued. If ‘smooth_GCV = FALSE‘, specific tuning parameters are given directly. If ‘method‘ is "power" and ‘alpha_orth = FALSE‘ (sequential power), tuning parameters should be provided as a list with length equal to the number of variables, where each element is a vector of length ‘ncomp‘. If ‘method‘ is "power" and ‘alpha_orth = TRUE‘ (joint power), tuning parameters should be provided as a vector with length equal to the number of variables. If the dimensions of input tuning parameters are incorrect, it will be converted to a list internally, and a warning will be issued.

Active bindings

`pc_mfd` An object of class 'mvmfd' where the first indices (fields) represents harmonics and second indices represents variables

`lsv` = Left singular values vectors

`values` = The set of eigenvalues

`smooth_tuning` = The list of smoothing penalties parameters

`sparse_tuning` = The list of sparse penalties parameters

`GCVs` = Generalized cross validations scores of smoothing penalties parameters. If both smoothing and sparse tuning penalties are used in the ReMFPCA method, this represents the conditional generalized cross-validation scores, which means it is computed based on the optimal sparse tuning parameter selected via cross validation.

`CVs` = Cross validations scores of sparse penalties parameters

`mean_mfd` A multivariate functional data object giving the mean function

Methods

Public methods:

- `remfpca$new()`
- `remfpca$clone()`

Method `new()`**:** Initialize the 'remfpca' class.

Usage:

```
remfpca$new(
  mvmfd_obj,
  method = "power",
  ncomp,
  smooth_tuning = NULL,
  sparse_tuning = NULL,
  centerfns = TRUE,
  alpha_orth = FALSE,
  smoothing_type = "coefpen",
  sparse_type = "soft",
  K_fold = 30,
  sparse_CV,
  smooth_GCV
)
```

Arguments:

`mvmfd_obj` An 'mvmfd' object representing the multivariate functional data.

`method` A character string specifying the approach to be used for MFPCA computation. Options are "power" (the default) or "eigen".

`ncomp` The number of functional principal components to retain.

`smooth_tuning` A list or vector specifying the smoothing regularization parameter(s).

`sparse_tuning` A list or vector specifying the sparsity regularization parameter(s).

`centerfns` Logical. Whether to center the functional data before analysis.

`alpha_orth` Logical. Whether to perform orthogonalization of the regularization parameters.

`smoothing_type` The type of smoothing penalty to be applied.

`sparse_type` The type of sparse penalty to be applied.

`K_fold` An integer specifying the number of folds in cross-validation.

`sparse_CV` Logical. Whether cross-validation should be applied for sparse tuning.

`smooth_GCV` Logical. Whether generalized cross-validation should be applied for smoothing tuning.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
remfpca$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

`mvmfd`

Examples

```
require(fda)
# Brownian Bridge simulation on [0,1]
M <- 110 # number of components
N <- 20 # number of instances
n <- 100 # number of grides
t0 <- seq(0, 1, len = n)
j <- 1:M
alpha1 <- list(a1 = 2^seq(0, 1, length.out = 3), a2 = 2^seq(0, 1, length.out = 3))
sparse_tuning = as.integer(seq(1, N-1, length.out = 10))
psi_1 <- function(t, m) sin(m * pi * t) # eigenfunction of BB
psi_2 <- function(t, m) sin((2 * m - 1) * pi / 2 * t) # eigenfunction of BM
PC_1 <- outer(t0, j, FUN = psi_1) # n by M matrix
PC_2 <- outer(t0, j, FUN = psi_2) # n by M matrix
Z <- matrix(rnorm(N * M), nr = M)
lambda <- matrix(2 / (pi * (2 * j - 1)), nr = M, nc = N)
X_1t <- PC_1 %*% (lambda * Z)
X_2t <- PC_2 %*% (lambda * Z)
noise <- rnorm(n * N, 0, 0.1)
X_1 <- X_1t + noise
X_2 <- X_2t + noise
bs <- create.bspline.basis(c(0, 1), 51)
mdbs <- Basismfd(bs)
mfd1 <- Mfd(X = X_1, mdbs = mdbs)
mfd2 <- Mfd(X = X_2, mdbs = mdbs)
mvmfd_obj <- Mvmfd(mfd1, mfd2)
k <- 2
# Non Regularized MFPCA based on sequential power algorithm
Re0 <- Remfpca(mvmfd_obj, ncomp = k, smooth_GCV = FALSE, sparse_CV = FALSE)
fpc0 <- Re0$pc_mfd
scores0 <- inprod_mvmfd(mvmfd_obj, fpc0)
```

```
dim(scores0)
# Smooth MFPCA based on sequential power algorithm
Re1 <- Remfpca(mvmfd_obj, ncomp = k, smooth_tuning = alpha1)
# Smooth and sparse MFPCA based on sequential power algorithm
Re2 <- Remfpca(mvmfd_obj, ncomp = k, smooth_tuning = alpha1, sparse_tuning = sparse_tuning)
# Smooth MFPCA based on joint power algorithm
Re3 <- Remfpca(mvmfd_obj, ncomp = k, smooth_tuning = alpha1, alpha_orth = TRUE)
# Smooth MFPCA based on eigen decomposition algorithm
Re4 <- Remfpca(mvmfd_obj, ncomp = k, smooth_tuning = alpha1, method = "eigen")
```

---

sd *Standard deviation of an object of class 'mfd'.*

Description

Standard deviation an object of class 'mfd'.

Usage

```
sd(x, ...)
```

Arguments

x	An object of class 'mfd'
...	all 'sd' function arguments.

Value

An object of class 'mfd'

[.mfd *Extract subsets of an 'mfd' object*

Description

Extract subsets of an 'mfd' object

Usage

```
## S3 method for class 'mfd'
mfd_obj[i = "index"]
```

Arguments

mfd_obj	An 'mfd' object
i	An index or indices specifying the subsets to extract

Value

An ‘mfd‘ object containing the specified subsets

See Also

basismfd, mfd

`[.mvmfd` *Extract subsets of an ‘mvmfd‘ object*

Description

Extract subsets of an ‘mvmfd‘ object

Usage

```
## S3 method for class 'mvmfd'
mvmfd_obj[i = "index", j = "index"]
```

Arguments

mvmfd_obj	An ‘mvmfd‘ object
i	An index or indices specifying the subsets to extract for the first dimension
j	An index or indices specifying the subsets to extract for the second dimension

Value

An ‘mvmfd‘ object containing the specified subsets

See Also

mvmfd,mvbasismfd

Index