Package 'funData'

February 14, 2024

Type Package

Title An S4 Class for Functional Data

Version 1.3-9

Date 2024-02-10

Author Clara Happ-Kurz [aut, cre]

Maintainer Clara Happ-Kurz <chk_R@gmx.de>

Description S4 classes for univariate and multivariate functional data with
utility functions. See <doi:10.18637/jss.v093.i05> for a detailed description
of the package functionalities and its interplay with the MFPCA package for
multivariate functional principal component analysis
<https://CRAN.R-project.org/package=MFPCA>.

URL https://github.com/ClaraHapp/funData

License GPL-2

Depends methods

Imports abind, fields, foreach, graphics, grDevices, stats

Suggests covr, fda, ggplot2 (>= 3.0.0), gridExtra, reshape2, zoo,
testthat (>= 2.0.0)

RoxygenNote 7.3.1

Encoding UTF-8

Collate 'funDataClass.R' 'coerce.R' 'funDataMethods.R' 'get_set.R'
'names.R' 'plotMethods.R' 'simulation.R' 'str.R' 'subset.R'
'summary.R' 'zzz.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2024-02-14 16:10:02 UTC

R topics documented:

.intWeights   *Calculate weights for numerical integration*

Description

This function calculates the weights for numerical integration

Usage

```
.intWeights(argvals, method = "trapezoidal")
```

Arguments

argvals	A numeric vector of x-Values
method	A character string, giving the numerical integration method to use (default is trapezoidal, alternatively use midpoint)

Value

A vector of integration weights

See Also

[integrate](#)

.scalarProduct   *Generic method for scalar products, based on integrate*

Description

Generic method for scalar products, based on integrate

Usage

```
.scalarProduct(object1, object2, ...)
```

Arguments

object1, object2

           Generic objects

...   Further objects passed to [integrate](#)

---

addError   *Add Gaussian white noise to functional data objects*

Description

This function generates an artificial noisy version of a functional data object of class [funData](#) (univariate) or [multiFunData](#) (multivariate) by adding iid. realizations of Gaussian random variables $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ to the observations. The standard deviation σ can be supplied by the user.

Usage

```
addError(funDataObject, sd)
```

Arguments

funDataObject   A functional data object of class [funData](#) or [multiFunData](#).

sd   The standard deviation σ of the Gaussian white noise that is added to the data. Defaults to 1. See Description.

Value

An object of the same class as funDataObject, which is a noisy version of the original data.

See Also

[funData](#), [multiFunData](#), [simFunData](#), [simMultiFunData](#).

Examples

```
oldPar <- par(no.readonly = TRUE)
set.seed(1)

# Univariate functional data
plain <- simFunData(argvals = seq(0,1,0.01), M = 10, eFunType = "Fourier",
                    eValType = "linear", N = 1)$simData
noisy <- addError(plain , sd = 0.5)
veryNoisy <- addError(plain, sd = 2)

plot(plain, main = "Add error", ylim = range(veryNoisy@X))
plot(noisy, type = "p", pch = 20, add = TRUE)
plot(veryNoisy, type = "p", pch = 4, add = TRUE)
legend("topright", c("Plain", "Noisy", "Very Noisy"), lty = c(1, NA, NA), pch = c(NA, 20 ,4))

# Multivariate functional data
plain <- simMultiFunData(type = "split", argvals = list(seq(0,1,0.01), seq(-.5,.5,0.02)), M = 10,
                         eFunType = "Fourier", eValType = "linear", N = 1)$simData
noisy <- addError(plain , sd = 0.5)
veryNoisy <- addError(plain, sd = 2)

par(mfrow = c(1,2))
plot(plain[[1]], main = "Add error (multivariate)", ylim = range(veryNoisy[[1]]@X))
plot(noisy[[1]], type = "p", pch = 20, add = TRUE)
plot(veryNoisy[[1]], type = "p", pch = 4, add = TRUE)

plot(plain[[2]], main = "Add error (multivariate)", ylim = range(veryNoisy[[2]]@X))
plot(noisy[[2]], type = "p", pch = 20, add = TRUE)
plot(veryNoisy[[2]], type = "p", pch = 4, add = TRUE)
legend("topright", c("Plain", "Noisy", "Very Noisy"), lty = c(1, NA, NA), pch = c(NA, 20 ,4))

par(oldPar)
```

---

approxNA   *Approximate missing values for funData objects*

Description

This function approximates missing values for funData objects based on the na.approx interpolation method from the package **zoo**.

Usage

```
approxNA(object)
```

Arguments

object   An object of class funData with missing values (coded by NA).

Value

A funData object where missing values have been imputed.

Warning

This function requires the package **zoo** to be installed, otherwise it will throw a warning.

Examples

```
# Simulate some data
f <- simFunData(N = 10, M = 8, eVal = "linear", eFun = "Poly", argvals = seq(0, 1, 0.01))$simData

# Sparsify, i.e. generate artificial missings in the data
fSparse <- sparsify(f, minObs = 10, maxObs = 50)

# plot
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(1,3))
plot(f, main = "Original Data")
plot(fSparse, main = "Sparse Data")
plot(approxNA(fSparse), main = "Reconstructed Data")
# faster with plot(fSparse, plotNA = TRUE, main = "Reconstructed Data")

par(oldpar)
```

Arith.funData   *Arithmetics for functional data objects*

Description

These functions allow basic arithmetics (such as '+', '-', '*', 'sqrt') for functional data and numerics
based on `Arith`. The operations are made pointwise for each observation. See examples below.

Usage

```
## S4 method for signature 'funData,funData'
Arith(e1, e2)

## S4 method for signature 'funData,numeric'
Arith(e1, e2)

## S4 method for signature 'numeric,funData'
Arith(e1, e2)

## S4 method for signature 'multiFunData,multiFunData'
Arith(e1, e2)

## S4 method for signature 'multiFunData,numeric'
Arith(e1, e2)

## S4 method for signature 'numeric,multiFunData'
Arith(e1, e2)

## S4 method for signature 'irregFunData,numeric'
Arith(e1, e2)

## S4 method for signature 'numeric,irregFunData'
Arith(e1, e2)

## S4 method for signature 'irregFunData,irregFunData'
Arith(e1, e2)

## S4 method for signature 'irregFunData,funData'
Arith(e1, e2)

## S4 method for signature 'funData,irregFunData'
Arith(e1, e2)
```

Arguments

e1, e2   Objects of class `funData`, `irregFunData`, `multiFunData` or `numeric`. If two
                functional data objects are used, they must be of the same class, have the same
                domain and the same number of observations. For exceptions, see Details.

Details

If two objects of a functional data class (funData, irregFunData or multiFunData) are used, they normally must be of the same class, have the same domain and the same number of observations. Exceptions are accepted if

- one object has only one observation. In this case, the arithmetic operations ('+', '-', '*', ...) are done pairwise for this single function and all functions of the other object. A typical example would be when subtracting the mean function from all observations in a funData object. This single function must be defined on the same domain as the other functions (or, in case of irregFunData, on the union of all observation grids).
- one of the two objects is of class irregFunData. Then, the other object can be of class funData, too, if it is defined on the union of all observation grids. The result is an irregFunData object which is defined on the same observation grid as the original irregFunData object.

Value

An object of the same functional data class as e1 or e2, respectively.

Warning

Note that not all combinations of operations and classes make sense, e.g. e1 ^ e2 is sensible if e1 is of class funData, irregFunData or multiFunData and e2 is numeric. The reverse is not true.

See Also

[funData](), [irregFunData](), [multiFunData](), [Arith]()

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(3,2), mar = rep(2.1,4))

argvals <- seq(0, 2*pi, 0.01)
object1 <- funData(argvals, outer(seq(0.75, 1.25, by = 0.05), sin(argvals)))
object2 <- funData(argvals, outer(seq(0.75, 1.25, by = 0.05), cos(argvals)))

plot(object1, main = "Object1")
plot(object2, main = "Object2")

# Only functional data objects
plot(object1 + object2, main = "Sum")
plot(object1 - object2, main = "Difference")

# Mixed
plot(4 * object1 + 5,  main = "4 * Object1 + 5") # Note y-axis!
plot(object1^2 + object2^2, main = "Pythagoras")

### Irregular
ind <- replicate(11, sort(sample(1:length(argvals), sample(5:10, 1))))
i1 <- irregFunData(
  argvals = lapply(1:11, function(i, ind, x){x[ind[[i]]]}, ind = ind, x = object1@argvals[[1]]),
```

```
    X = lapply(1:11, function(i, ind, y){y[i, ind[[i]]]}, ind = ind, y = object1@X))
i2 <- irregFunData(
  argvals = lapply(1:11, function(i, ind, x){x[ind[[i]]]}, ind = ind, x = object2@argvals[[1]]),
    X = lapply(1:11, function(i, ind, y){y[i, ind[[i]]]}, ind = ind, y = object2@X))

plot(i1, main = "Object 1 (irregular)")
plot(i2, main = "Object 2 (irregular)")

# Irregular and regular functional data objects
plot(i1 + i2, main = "Sum")
plot(i1 - object2, main = "Difference")

# Mixed
plot(4 * i1 + 5,  main = "4 * i1 + 5") # Note y-axis!
plot(i1^2 + i2^2, main = "Pythagoras")
par(oldpar)
```

---

as.data.frame.funData   *Coerce functional data objects to a data.frame*

Description

Coerce objects of class `funData`, `multiFunData` and `irregFunData` to a data frame.

Usage

```
## S4 method for signature 'funData'
as.data.frame(x)

## S4 method for signature 'multiFunData'
as.data.frame(x)

## S4 method for signature 'irregFunData'
as.data.frame(x)
```

Arguments

x   The functional data object that is to be transformed to a `data.frame`

Value

A data frame with columns obs (gives index/name of observed curve), argvals1, ... argvalsd
with d the dimension of the support and X for the observed values. One-dimensional functions have
only argvals1, two-dimensional functions (images) have argvals1 and argvals2, etc.

See Also

[funData](), [irregFunData](), [multiFunData](), [data.frame]()

Examples

```
# one-dimensional domain
f1 <- funData(argvals = 1:5, X = matrix(1:20, nrow = 4))
head(as.data.frame(f1))

# two-dimensional domain
f2 <- funData(argvals = list(1:5, 1:6), X = array(1:120, c(4,5,6)))
head(as.data.frame(f2))

# multivariate functional data
m1 <- multiFunData(f1, f2)
str(as.data.frame(m1))

# irregular functional data
i1 <- irregFunData(argvals = list(1:5, 2:4, 3:5), X = list(1:5, 2:4, -(3:1)))
head(as.data.frame(i1))
```

---

as.funData   *Coerce an irregFunData object to class funData*

Description

This function coerces an object of class `irregFunData` to a `funData` object with missing values, which is defined on the union of all observation points.

Usage

```
as.funData(object)

## S4 method for signature 'irregFunData'
as.funData(object)
```

Arguments

object   The `irregFunData` object that is to be converted to a `funData` object with miss-
                ing values.

See Also

[funData](#), [irregFunData](#)

---

as.irregFunData   *Coerce a funData object to class irregFunData*

Description

This function coerces an object of class funData to a irregFunData object.

Usage

```
as.irregFunData(object)

## S4 method for signature 'funData'
as.irregFunData(object)
```

Arguments

object   The funData object that is to be converted to a irregFunData object.

See Also

funData, irregFunData

---

as.multiFunData   *Coerce a funData object to class multiFunData*

Description

Coerce a funData object to class multiFunData with one element.

Usage

```
as.multiFunData(object)

## S4 method for signature 'funData'
as.multiFunData(object)
```

Arguments

object   The funData object that is to be converted to a multiFunData object of length
                1.

See Also

funData, multiFunData

Examples

```
# create funData object with 5 observations
x <- seq(0,1,0.01)
f1 <- funData(argvals = x, X = 1:5 %o% x)
f1
class(f1)

# coerce to multiFunData object (of length 1)
m1 <- as.multiFunData(f1)
m1
class(m1)
```

---

autoplot.funData   *Visualize functional data objects using ggplot*

Description

This function allows to plot funData objects based on the **ggplot2** package. The function provides a wrapper that rearranges the data in a funData object on a one- or two-dimensional domain and provides a basic [ggplot](#) object, which can be customized using all functionalities of the **ggplot2** package.

Usage

```
autoplot.funData(
  object,
  obs = seq_len(nObs(object)),
  geom = "line",
  plotNA = FALSE,
  ...
)

autolayer.funData(
  object,
  obs = seq_len(nObs(object)),
  geom = "line",
  plotNA = FALSE,
  ...
)
```

Arguments

object	A funData object on a one- or two-dimensional domain.
obs	A vector of numerics giving the observations to plot. Defaults to all observations in object. For two-dimensional functions (images) obs must have length 1.
geom	A character string describing the geometric object to use. Defaults to "line". See **ggplot2** for details.

plotNA	Logical. If TRUE, missing values are interpolated using the [approxNA](#) function (only for one-dimensional functions). Defaults to FALSE. See Details.
...	Further parameters passed to [geom_line](#) (for one dimensional domains, e.g. alpha, color, fill, linetype, size) or to [geom_raster](#) (for two-dimensional domains, e.g. hjust, vjust, interpolate).

Details

If some observations contain missing values (coded via NA), the functions can be interpolated using the option plotNA = TRUE. This option relies on the [na.approx](#) function in package [zoo](#) and is currently implemented for one-dimensional functions only in the function [approxNA](#).

Value

A [ggplot](#) object that can be customized using all functionalities of the **ggplot2** package.

See Also

[funData](#), [ggplot](#), [plot.funData](#)

Examples

```
# Install / load package ggplot2 before running the examples
library("ggplot2")

# One-dimensional
argvals <- seq(0,2*pi,0.01)
object <- funData(argvals,
                  outer(seq(0.75, 1.25, length.out = 11), sin(argvals)))

g <- autoplot(object) # returns ggplot object
g # plot the object

# add the mean function in red
g + autolayer(meanFunction(object),  col = 2)

# Two-dimensional
X <- array(0, dim = c(2, length(argvals), length(argvals)))
X[1,,] <- outer(argvals, argvals, function(x,y){sin((x-pi)^2 + (y-pi)^2)})
X[2,,] <- outer(argvals, argvals, function(x,y){sin(2*x*pi) * cos(2*y*pi)})
object2D <- funData(list(argvals, argvals), X)


autoplot(object2D, obs = 1)
autoplot(object2D, obs = 2)
## Not run: autoplot(object2D) # must specify obs!

### More examples ###

par(mfrow = c(1,1))

# using plotNA (needs packages zoo and gridExtra)
```

```
objectMissing <- funData(1:5, rbind(c(1, NA, 5, 4, 3), c(10, 9, NA, NA, 6)))
g1 <- autoplot(objectMissing) # the default
g2 <- autoplot(objectMissing, plotNA = TRUE) # requires zoo

gridExtra::grid.arrange(g1 + ggtitle("plotNA = FALSE (default)"),
                        g2 + ggtitle("plotNA = TRUE")) # requires gridExtra

# Customizing plots (see ggplot2 documentation for more details)
# parameters passed to geom_line are passed via the ... argument
gFancy <- autoplot(object, color = "red", linetype = 2)
gFancy

# new layers can be added directly to the ggplot object
gFancy + theme_bw() # add new layers to the ggplot object
gFancy + ggtitle("Fancy Plot with Title and Axis Legends") +
         xlab("The x-Axis") + ylab("The y-Axis")

autoplot(object2D, obs = 1) + ggtitle("Customized 2D plot") + theme_minimal() +
          scale_fill_gradient(high = "green", low = "blue", name = "Legend here")
```

---

autoplot.irregFunData *Visualize irregular functional data objects using ggplot*

Description

This function allows to plot irregFunData objects on their domain based on the **ggplot2** package. The function provides a wrapper that returns a basic [ggplot](#) object, which can be customized using all functionalities of the **ggplot2** package.

Usage

```
autoplot.irregFunData(object, obs = seq_len(nObs(object)), geom = "line", ...)

autolayer.irregFunData(object, obs = seq_len(nObs(object)), geom = "line", ...)
```

Arguments

object	A irregFunData object.
obs	A vector of numerics giving the observations to plot. Defaults to all observations in object. For two-dimensional functions (images) obs must have length 1.
geom	A character string describing the geometric object to use. Defaults to "line". See **ggplot2** for details.
...	Further parameters passed to [stat_identity](#), e.g. alpha, color, fill, linetype, size).

Value

A [ggplot](#) object that can be customized using all functionalities of the **ggplot2** package.

See Also

[irregFunData](#), [ggplot](#), [plot.irregFunData](#)

Examples

```
# Install / load package ggplot2 before running the examples
library("ggplot2")

# Generate data
argvals <- seq(0,2*pi,0.01)
ind <- replicate(5, sort(sample(1:length(argvals), sample(5:10,1))))
object <- irregFunData(argvals = lapply(ind, function(i){argvals[i]}),
                  X = lapply(ind, function(i){sample(1:10,1) / 10 * argvals[i]^2}))

# Plot the data
autoplot(object)

 # Parameters passed to geom_line are passed via the ... argument
autoplot(object, color = "red", linetype = 3)

# Plot the data and add green dots for the 2nd function
autoplot(object) + autolayer(object, obs = 2, geom = "point", color = "green")

# New layers can be added directly to the ggplot object using functions from the ggplot2 package
g <- autoplot(object)
g + theme_bw() + ggtitle("Plot with minimal theme and axis labels") +
    xlab("The x-Axis") + ylab("The y-Axis")
```

---

autoplot.multiFunData   *Visualize multivariate functional data objects using ggplot*

Description

This function allows to plot multiFunData objects based on the **ggplot2** package. The function applies the [autoplot.funData](#) function to each element and returns either a combined plot with all elements plotted in one row or a list containing the different subplots as [ggplot](#) objects. The individual objects can be customized using all functionalities of the **ggplot2** package.

Usage

```
autoplot.multiFunData(
  object,
  obs = seq_len(nObs(object)),
  dim = seq_len(length(object)),
  plotGrid = FALSE,
```

```
   ...
)
```

Arguments

object	A multiFunData object that is to be plotted.
obs	A vector of numerics giving the observations to plot. Defaults to all observations in object. For two-dimensional functions (images) obs must have length 1.
dim	The dimensions to plot. Defaults to length(object), i.e. all functions in object are plotted.
plotGrid	Logical. If TRUE, the data is plotted using grid.arrange and the list of ggplot objects is returned invisibly. If FALSE, only the list of objects is returned. Defaults to FALSE.
...	Further parameters passed to the univariate autoplot.funData functions for funData objects.

Value

A list of ggplot objects that are also printed directly as a grid if plotGrid = TRUE.

Warning

Currently, the function does not accept different parameters for the univariate elements.

See Also

multiFunData, ggplot, plot.multiFunData

Examples

```
# Load packages ggplot2 and gridExtra before running the examples
library("ggplot2"); library("gridExtra")

# One-dimensional elements
argvals <- seq(0, 2*pi, 0.01)
f1 <- funData(argvals, outer(seq(0.75, 1.25, length.out = 11), sin(argvals)))
f2 <- funData(argvals, outer(seq(0.75, 1.25, length.out = 11), cos(argvals)))

m1 <- multiFunData(f1, f2)

g <- autoplot(m1) # default
g[[1]] # plot first element
g[[2]] # plot second element
gridExtra::grid.arrange(grobs = g, nrow = 1) # requires gridExtra package

autoplot(m1, plotGrid = TRUE) # the same directly with plotGrid = TRUE


# Mixed-dimensional elements
X <- array(0, dim = c(11, length(argvals), length(argvals)))
```

```
X[1,,] <- outer(argvals, argvals, function(x,y){sin((x-pi)^2 + (y-pi)^2)})
f2 <- funData(list(argvals, argvals), X)

m2 <- multiFunData(f1, f2)

autoplot(m2, obs = 1, plotGrid = TRUE)

# Customizing plots (see ggplot2 documentation for more details)
g2 <- autoplot(m2, obs = 1)
g2[[1]] <- g2[[1]] + ggtitle("First element") + theme_bw()
g2[[2]] <- g2[[2]] + ggtitle("Second element") +
                     scale_fill_gradient(high = "green", low = "blue")
gridExtra::grid.arrange(grobs = g2, nrow = 1) # requires gridExtra package
```

---

dimSupp   *Support dimension of functional data*

Description

This function returns the support dimension of an object of class funData, irregFunData or multiFunData.

Usage

```
dimSupp(object)
```

Arguments

object   An object of class funData, irregFunData or multiFunData.

Value

If object is univariate (i.e. of class funData or irregFunData), the function returns the dimension of the support of object. If object is multivariate (i.e. of class multiFunData), the function returns a vector, giving the support dimension of each element.

See Also

[funData](#), [irregFunData](#), [multiFunData](#)

Examples

```
# Univariate (one-dimensional)
object1 <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
dimSupp(object1)

# Univariate (two-dimensional)
object2 <- funData(argvals = list(1:10, 1:5), X = array(rnorm(100), dim = c(2,10,5)))
```

```
dimSupp(object2)

# Univariate (irregular)
irregObject <- irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
dimSupp(irregObject)

# Multivariate
multiObject <- multiFunData(object1, object2)
dimSupp(multiObject)
```

---

eFun   *Generate orthonormal eigenfunctions*

Description

This function calculates M (orthonormal) basis functions on a given interval, that can be interpreted as the first M eigenfunctions of an appropriate data generating process of functional data.

Usage

```
eFun(argvals, M, ignoreDeg = NULL, type)
```

Arguments

argvals   A vector of numerics, defining a (fine) grid on the interval for which the basis
               functions are computed.

M   An integer, specifying the number of functions that are calculated.

ignoreDeg   A vector of numerics, specifying the degrees to be ignored for type "PolyHigh".
               Defaults to NULL. See Details.

type   A character string, specifying the type of functions that are calculated. See
               Details.

Details

The function implements three families of orthonormal basis functions plus variations of them. The parameter type, that specifies the functions to be calculated, can have the following values:

- "Poly": Calculate orthonormal Legendre polynomials of degree 0,...,M-1.

- "PolyHigh": Calculate M orthonormal Legendre Polynomials of higher degree. The vector of indices ignoreDeg specifies the functions to be ignored. If ignoreDeg is not specified, the function returns an error.

- "Fourier": Calculate the first M Fourier basis functions.

- "FourierLin": Calculate the first $M - 1$ Fourier basis functions plus the linear function, orthonormalized to the previous functions via Gram-Schmidts method. This type is currently implemented for functions on the unit interval $[0, 1]$ only. If the function is called with other argvals, an error is thrown.

- "Wiener": Calculate the first M orthonormal eigenfunctions of the Wiener process.

Value

A univariate functional data object of class [funData](#) containing the basis functions on the given interval.

See Also

[funData](#), [simFunData](#), [simMultiFunData](#)

Examples

```
oldPar <- par(no.readonly = TRUE)

argvals <- seq(0,1,0.01)

par(mfrow = c(3,2))
plot(eFun(argvals, M = 4, type = ″Poly″), main = ″Poly″, ylim = c(-3,3))
plot(eFun(argvals, M = 4, ignoreDeg = 1:2, type = ″PolyHigh″), main = ″PolyHigh″, ylim = c(-3,3))
plot(eFun(argvals, M = 4, type = ″Fourier″), main = ″Fourier″, ylim = c(-3,3))
plot(eFun(argvals, M = 4, type = ″FourierLin″), main = ″FourierLin″, ylim = c(-3,3))
plot(eFun(argvals, M = 4, type = ″Wiener″), main = ″Wiener″,  ylim = c(-3,3))
par(oldPar)
```

---

eVal   *Generate a sequence of simulated eigenvalues*

Description

This function generates M decreasing eigenvalues.

Usage

```
eVal(M, type)
```

Arguments

M	An integer, the number of eigenvalues to be generated.
type	A character string specifying the type of eigenvalues that should be calculated. See Details.

Details

The function implements three types of eigenvalues:

- ″linear″: The eigenvalues start at 1 and decrease linearly towards 0:

$$\nu_m = \frac{M + 1 - m}{m}.$$

- "exponential": The eigenvalues start at 1 and decrease exponentially towards 0:

$$\nu_m = \exp\left(-\frac{m-1}{2}\right).$$

- "wiener": The eigenvalues correspond to the eigenvalues of the Wiener process:

$$\nu_m = \frac{1}{(\pi/2 \cdot (2m-1))^2}.$$

Value

A vector containing the M decreasing eigenvalues.

Examples

```
oldpar <- par(no.readonly = TRUE)

# simulate M = 10 eigenvalues
M <- 10
eLin <- eVal(M = M, type = "linear")
eExp <- eVal(M = M, type = "exponential")
eWien <- eVal(M = M, type = "wiener")

par(mfrow = c(1,1))
plot(1:M, eLin, pch = 20, xlab = "m", ylab = expression(nu[m]), ylim = c(0,1))
points(1:M, eExp, pch = 20, col = 3)
points(1:M, eWien, pch = 20, col = 4)
legend("topright", legend = c("linear", "exponential", "wiener"), pch = 20, col = c(1,3,4))

par(oldpar)
```

---

extractObs   *Extract observations of functional data*

Description

This function extracts one or more observations and/or observations on a part of the domain from a funData, irregFunData or multiFunData object.

Usage

```
extractObs(
  object,
  obs = seq_len(nObs(object)),
  argvals = funData::argvals(object)
)

## S4 method for signature 'funData'
```

```
subset(x, obs = seq_len(nObs(x)), argvals = funData::argvals(x))

## S4 method for signature 'multiFunData'
subset(x, obs = seq_len(nObs(x)), argvals = funData::argvals(x))

## S4 method for signature 'irregFunData'
subset(x, obs = seq_len(nObs(x)), argvals = funData::argvals(x))

## S4 method for signature 'funData,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'multiFunData,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'irregFunData,ANY,missing,missing'
x[i = seq_len(nObs(x)), j, ..., drop = TRUE]
```

Arguments

object	An object of class funData, irregFunData or multiFunData.
obs	A numeric vector, giving the indices of the observations to extract (default: all observations).
argvals	The part of the domain to be extracted (default: the whole domain object@argvals). Must be a list or a numeric vector (only for one-dimensional domains, see also the definition of [funData](#), [multiFunData](#)).
x	An object of class funData, irregFunData or multiFunData (for subset).
i	A numeric vector, giving the indices of the observations to extract when using x[i]. Defaults to all observations.
j, drop	not used
...	Used to pass further arguments to extractObs. Here only usable for argvals.

Details

In case of an irregFunData object, some functions may not have observation points in the given part of the domain. In this case, the functions are removed from the extracted dataset and a warning is thrown.

If only observations are to be extracted, the usual notation object[1:3] is equivalent to extractObs(object, obs = 1:3). This works only if the domain remains unchanged.

Value

An object of class funData, irregFunData or multiFunData containing the desired observations.

Functions

- x[i:

Warning

The function is currently implemented only for functional data with up to three-dimensional domains.

Alias

The function subset is an alias for extractObs.

See Also

funData, irregFunData, multiFunData

Examples

```
# Univariate - one-dimensional domain
object1 <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
extractObs(object1, obs = 1)
extractObs(object1, argvals = 1:3)
extractObs(object1, argvals = list(1:3)) # the same as the statement before
# alias
subset(object1, argvals = 1:3)

# Univariate - two-dimensional domains
object2 <- funData(argvals = list(1:5, 1:6), X = array(1:60, dim = c(2, 5, 6)))
extractObs(object2, obs = 1)
extractObs(object2, argvals = list(1:3, c(2,4,6))) # argvals must be supplied as list

# Univariate - irregular
irregObject <- irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
extractObs(irregObject, obs = 2)
extractObs(irregObject, argvals = 1:3)
extractObs(irregObject, argvals = c(1,5)) # throws a warning, as second function has no observations

# Multivariate
multiObject <- multiFunData(object1, object2)
extractObs(multiObject, obs = 2)
multiObject[2] # shorthand
extractObs(multiObject, argvals = list(1:3, list(1:3, c(2,4,6))))


### Shorthand via "[]"
object1[1]
object1[argvals = 1:3]
object2[1]
object2[argvals = list(1:3, c(2,4,6))]
irregObject[2]
irregObject[argvals = 1:3]
```

---

fd2funData   *Convert an fd object to funData*

Description

This function converts an object of class [fd](#) (from package **fda**) to an object of class [funData](#). It heavily builds on the function [eval.fd](#) from the **fda** package. The [fd](#) representation assumes a basis representation for the observed functions and therefore implicitly smoothes the data. In funData objects, the data is saved in 'raw' format.

Usage

```
fd2funData(fdobj, argvals, ...)
```

Arguments

fdobj	An fd object
argvals	A vector or a list of length one, containing a vector with argument values at which the functions in fdobj should be evaluated.
...	Other parameters passed to [eval.fd](#).

Value

An object of class [funData](#).

Warning

Time names in fdobj$fdnames$time are not preserved.

See Also

[funData](#), [fd](#), [eval.fd](#)

Examples

```
# Install / load package fda before running the examples
library("fda")

# from Data2fd help
daybasis <- create.fourier.basis(c(0, 365), nbasis=65)
# fd object of daily temperatures
tempfd <- Data2fd(argvals = day.5, y = CanadianWeather$dailyAv[,,"Temperature.C"], daybasis)
# convert to funData
tempFun <- fd2funData(tempfd, argvals = day.5)

# plot to compare
par(mfrow = c(1,2))
plot(tempfd, main = "fd object")
plot(tempFun, main = "funData object")
```

---

flipFuns | *Flip functional data objects*

Description

This function flips an object newObject of class funData, irregFunData or multiFunData with respect to a reference object refObject of the same class (or of class funData, if newObject is irregular). This is particularly useful when dealing with functional principal components, as they are only defined up to a sign change. For details, see below.

Usage

```
flipFuns(refObject, newObject, ...)
```

Arguments

refObject	An object of class funData, irregFunData or multiFunData that serves as reference. It must have the same number of observations as newObject or have only one observation. In this case, all observations in newObject are flipped with respect to this single observation.
newObject	An object of class funData, irregFunData or multiFunData that is to be flipped with respect to refObject.
...	Further parameters passed to [norm](#).

Details

Functional principal component analysis is an important tool in functional data analysis. Just as eigenvectors, eigenfunctions (or functional principal components) are only defined up to a sign change. This may lead to difficulties in simulation studies or when bootstrapping pointwise confidence bands, as in these cases one wants the estimates to have the same "orientation" as the true function (in simulation settings) or the non-bootstrapped estimate (when calculating bootstrap confidence bands). This function allows to flip (i.e. multiply by -1) all observations in newObject that have a different orientation than their counterparts in refData.

Technically, the function compares the distance between newObject and refObject

$$|||f_{\mathrm{new}} - f_{\mathrm{ref}}|||$$

and the distance between newObject and -1 * refObject

$$|||f_{\mathrm{new}} + f_{\mathrm{ref}}|||.$$

If newObject is closer to -1 * refObject, it is flipped, i.e. multiplied by -1.

Value

An object of the same class as newData with flipped observations.

Warning

The function is currently implemented only for functional data with one- and two-dimensional domains.

See Also

funData, irregFunData, multiFunData, Arith.funData

Examples

```
### Univariate
argvals <- seq(0,2*pi,0.01)
refData <- funData(argvals, rbind(sin(argvals))) # one observation as reference
newData <- funData(argvals, outer(sample(c(-1,1), 11, replace = TRUE) * seq(0.75, 1.25, by = 0.05),
                                  sin(argvals)))

oldpar <- par(no.readonly = TRUE)
par(mfrow = c(1,2))

plot(newData, col = "grey", main = "Original data")
plot(refData, col = "red", lwd = 2, add = TRUE)

plot(flipFuns(refData, newData), col = "grey", main = "Flipped data")
plot(refData, col = "red", lwd = 2, add = TRUE)

### Univariate (irregular)
ind <- replicate(11, sort(sample(1:length(argvals), sample(5:10,1)))) # sample observation points
argvalsIrreg <- lapply(ind, function(i){argvals[i]})
argvalsIrregAll <- unique(sort(unlist(argvalsIrreg)))
 # one observation as reference (fully observed)
refDataFull <- funData(argvals, rbind(sin(argvals)))
 # one observation as reference (irregularly observed)
refDataIrreg <- irregFunData(argvals = list(argvalsIrregAll), X = list(sin(argvalsIrregAll)))
newData <- irregFunData(argvals = argvalsIrreg, X = mapply(function(x, a, s){s * a * sin(x)},
    x = argvalsIrreg, a = seq(0.75, 1.25, by = 0.05), s = sample(c(-1,1), 11, replace = TRUE)))

plot(newData, col = "grey", main = "Original data (regular reference)")
plot(refDataFull, col = "red", lwd = 2, add = TRUE)

plot(flipFuns(refDataFull, newData), col = "grey", main = "Flipped data")
plot(refDataFull, col = "red", lwd = 2, add = TRUE)

plot(newData, col = "grey", main = "Original data (irregular reference)")
plot(refDataIrreg, col = "red", lwd = 2, add = TRUE)

plot(flipFuns(refDataIrreg, newData), col = "grey", main = "Flipped data")
plot(refDataIrreg, col = "red", lwd = 2, add = TRUE)

### Multivariate
refData <- multiFunData(funData(argvals, rbind(sin(argvals))), # one observation as reference
                        funData(argvals, rbind(cos(argvals))))
```

```
sig <- sample(c(-1,1), 11, replace = TRUE)
newData <- multiFunData(funData(argvals, outer(sig * seq(0.75, 1.25, by = 0.05), sin(argvals))),
                        funData(argvals, outer(sig * seq(0.75, 1.25, by = 0.05), cos(argvals))))

par(mfrow = c(2,2))

plot(newData[[1]], col = topo.colors(11), main = "Original data")
plot(refData[[1]], col = "red", lwd = 2, add = TRUE)

plot(newData[[2]], col = topo.colors(11), main = "Original data")
plot(refData[[2]], col = "red", lwd = 2, add = TRUE)

plot(flipFuns(refData, newData)[[1]], col = topo.colors(11), main = "Flipped data")
plot(refData[[1]], col = "red", lwd = 2, add = TRUE)

plot(flipFuns(refData, newData)[[2]], col = topo.colors(11), main = "Flipped data")
plot(refData[[2]], col = "red", lwd = 2, add = TRUE)

par(oldpar)
```

---

funData-class   *A class for (univariate) functional data*

Description

The funData class represents functional data on d-dimensional domains. The two slots represent the domain (x-values) and the values of the different observations (y-values).

Usage

```
funData(argvals, X)

## S4 method for signature 'list,array'
funData(argvals, X)

## S4 method for signature 'numeric,array'
funData(argvals, X)

## S4 method for signature 'funData'
show(object)

## S4 method for signature 'funData'
names(x)

## S4 replacement method for signature 'funData'
names(x) <- value

## S4 method for signature 'funData'
```

```
str(object, ...)

## S4 method for signature 'funData'
summary(object, ...)
```

Arguments

argvals	A list of numeric vectors or a single numeric vector, giving the sampling points in the domains. See Details.
X	An array of dimension $N \times M$ (for one-dimensional domains, or $N \times M_1 \times \ldots \times M_d$ for higher-dimensional domains), giving the observed values for N individuals. Missing values can be included via NA. See Details.
object	A funData object.
x	The funData object.
value	The names to be given to the funData curves.
...	Other parameters passed to summary.

Details

Functional data can be seen as realizations of a random process

$$X : \mathcal{T} \to \mathrm{IR}$$

on a d-dimensional domain $\mathcal{T}$. The data is usually sampled on a fine grid $T \subset \mathcal{T}$, which is represented in the argvals slot of a funData object. All observations are assumed to be sampled over the same grid T, but can contain missing values (see below). If $\mathcal{T}$ is one-dimensional, argvals can be supplied either as a numeric vector, containing the x-values or as a list, containing such a vector. If $\mathcal{T}$ is higher-dimensional, argvals must always be supplied as a list, containing numeric vectors of the x-values in dimensions $1, \ldots, d$.

The observed values are represented in the X slot of a funData object, which is an array of dimension $N \times M$ (for one-dimensional domains, or $N \times M_1 \times \ldots \times M_d$ for higher-dimensional domains). Here N equals the number of observations and M denotes the number of sampling points (for higher dimensional domains M_i denotes the number of sampling points in dimension $i, i = 1, \ldots, d$). Missing values in the observations are allowed and must be marked by NA. If missing values occur due to irregular observation points, the data can be stored alternatively as an object of class [irregFunData](#).

Generic functions for the funData class include a print method, [plotting](#) and [basic arithmetics](#). Further methods for funData:

- [dimSupp](#), [nObs](#): Informations about the support dimensions and the number of observations,

- [getArgvals](#), [extractObs](#): Getting/Setting slot values (instead of accessing them directly via funData@argvals, funData@X) and extracting single observations or data on a subset of the domain,

- [integrate](#), [norm](#): Integrate all observations over their domain or calculating the L^2 norm.

A funData object can be coerced to a multiFunData object using as.multiFunData(funDataObject).

Methods (by generic)

- funData(argvals = list, X = array): Constructor for functional data objects with argvals given as list.

- funData(argvals = numeric, X = array): Constructor for functional data objects with argvals given as vector of numerics (only valid for one-dimensional domains).

- show(funData): Print basic information about the funData object in the console. The default console output for funData objects.

- names(funData): Get the names of the funData object.

- names(funData) <- value: Set the names of the funData object.

- str(funData): A str method for funData objects, giving a compact overview of the structure.

- summary(funData): A summary method for funData objects.

Functions

- funData(): Constructor for functional data objects, first argument (argvals) passed as list or vector of numerics

Slots

argvals The domain $\mathcal{T}$ of the data. See Details.

X The functional data samples. See Details.

See Also

[irregFunData](#), [multiFunData](#)

Examples

```
### Creating a one-dimensional funData object with 2 observations
# Basic
f1 <- new("funData", argvals = list(1:5), X = rbind(1:5,6:10))
# Using the constructor with first argument supplied as array
f2 <- funData(argvals = list(1:5), X = rbind(1:5, 6:10))
# Using the constructor with first argument supplied as numeric vector
f3 <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
# Test if all the same
all.equal(f1,f2)
all.equal(f1,f3)
# Display funData object in the console
f3

# A more realistic object
argvals <- seq(0,2*pi,0.01)
object <- funData(argvals, outer(seq(0.75, 1.25, by = 0.05), sin(argvals)))
# Display / summary give basic information
object
summary(object)
```

```
# Use the plot function to get an impression of the data
plot(object)


### Higher-dimensional funData objects with 2 observations
# Basic
g1 <- new("funData", argvals = list(1:5, 1:3),
                     X = array(1:30, dim = c(2,5,3)))
# Using the constructor
g2 <- funData(argvals = list(1:5, 1:3),
              X = array(1:30, dim = c(2,5,3)))
# Test if the same
all.equal(g1,g2)
# Display funData object in the console
g2
# Summarize information
summary(g2)
```

---

funData2fd   *Convert a funData object to fd*

Description

This function converts an object of class funData to an object of class [fd](fd) (from package **fda**). It heavily builds on the function Data2fd from the **fda** package. The [fd](fd) representation assumes a basis representation for the observed functions and therefore implicitly smoothes the data. In funData objects, the data is saved in 'raw' format.

Usage

```
funData2fd(object, ...)
```

Arguments

object   A funData object

...   Other parameters passed to Data2fd.

Value

An object of class [fd](fd).

Warning

This function works only for funData objects on one-dimensional domains.

See Also

funData, [fd](fd), Data2fd, fd2funData

Examples

```
# Install / load package fda before running the examples
library("fda")

# from Data2fd help
daybasis <- create.fourier.basis(c(0, 365), nbasis=65)
# funData object with temperature
tempFun <- funData(day.5, t(CanadianWeather$dailyAv[, , "Temperature.C"]))
# convert to fd
tempfd <- funData2fd(tempFun, daybasis)

# plot to compare
par(mfrow = c(1,2))
plot(tempFun, main = "funData object (raw data)")
plot(tempfd, main = "fd object (smoothed)")
```

---

ggplot   *ggplot Graphics for Functional Data Objects*

Description

This function is deprecated. Use [autoplot.funData](#) / [autolayer.funData](#) for funData objects, [autoplot.multiFunData](#) for multiFunData objects and [autoplot.irregFunData](#) / [autolayer.irregFunData](#) for irregFunData objects instead.

Usage

```
ggplot(data, ...)

## S4 method for signature 'funData'
ggplot(data, add = FALSE, ...)

## S4 method for signature 'multiFunData'
ggplot(data, ...)

## S4 method for signature 'irregFunData'
ggplot(data, add = FALSE, ...)
```

Arguments

data	A funData, multiFunData or irregFunData object.
...	Further parameters passed to the class-specific methods.
add	Logical. If TRUE, add to current plot (only for one-dimensional functions). Defaults to FALSE.

Details

In the default case, this function calls [ggplot](#) (if available).

Value

A [ggplot](#) object

See Also

[ggplot](#), [autoplot](#), [autolayer](#) from package **ggplot2**

---

integrate   *Integrate functional data*

Description

Integrate all observations of a funData, irregFunData or multiFunData object over their domain.

Usage

```
integrate(object, ...)
```

Arguments

object	An object of class funData, irregFunData or multiFunData.
...	Further parameters (see Details).

Details

Further parameters passed to this function may include:

- method: Character string. The integration rule to be used, passed to the internal function .intWeights. Defaults to "trapezoidal" (alternative: "midpoint").
- fullDom: Logical. If object is of class irregFunData, setting fullDom = TRUE extrapolates all functions linearly to the full domain before calculating the integrals. Defaults to FALSE. For details on the extrapolation, see [extrapolateIrreg](#).

Value

A vector of numerics, containing the integral values for each observation.

Warning

The function is currently implemented only for functional data with up to three-dimensional domains. In the default case, this function calls [integrate](#).

See Also

[funData](#), [irregFunData](#), [multiFunData](#)

Examples

```
# Univariate
object <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
integrate(object)

# Univariate (irregular)
irregObject <-irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
integrate(irregObject) # fullDom = FALSE
integrate(irregObject, fullDom = TRUE)

# Multivariate
multiObject <- multiFunData(object, funData(argvals = 1:3, X = rbind(3:5, 6:8)))
integrate(multiObject)
```

---

irregFunData-class   *A class for irregularly sampled functional data*

Description

The `irregFunData` class represents functional data that is sampled irregularly on one-dimensional domains. The two slots represent the observation points (x-values) and the observed function values (y-values).

Usage

```
irregFunData(argvals, X)

## S4 method for signature 'list,list'
irregFunData(argvals, X)

## S4 method for signature 'irregFunData'
show(object)

## S4 method for signature 'irregFunData'
names(x)

## S4 replacement method for signature 'irregFunData'
names(x) <- value

## S4 method for signature 'irregFunData'
str(object, ...)

## S4 method for signature 'irregFunData'
summary(object, ...)
```

Arguments

argvals	A list of numerics, corresponding to the observation points for each realization X_i (see Details).
X	A list of numerics, corresponding to the observed functions X_i (see Details).
object	An irregFunData object.
x	The irregFunData object.
value	The names to be given to the irregFunData curves.
...	Other parameters passed to summary.

Details

Irregular functional data are realizations of a random process

$$X : \mathcal{T} \to \mathrm{IR},$$

where each realization X_i of X is given on an individual grid $T_i \subset \mathcal{T}$ of observation points. As for the [funData](#) class, each object of the irregFunData class has two slots; the argvals slot represents the observation points and the X slot represents the observed data. In contrast to the regularly sampled data, both slots are defined as lists of vectors, where each entry corresponds to one observed function:

- argvals[[i]] contains the vector of observation points T_i for the i-th function,
- X[[i]] contains the corresponding observed data $X_i(t_{ij}), t_{ij} \in T_i$.

Generic functions for the irregFunData class include a print method, [plotting](#) and [basic arithmetics](#). Further methods for irregFunData:

- [dimSupp](#), [nObs](#): Informations about the support dimensions and the number of observations,
- [getArgvals](#), [extractObs](#): Getting/setting slot values (instead of accessing them directly via irregObject@argvals, irregObject@X) and extracting single observations or data on a subset of the domain,
- [integrate](#), [norm](#): Integrate all observations over their domain or calculating the L^2 norm.

An irregFunData object can be coerced to a funData object using as.funData(irregObject). The regular functional data object is defined on the union of all observation grids of the irregular object. The value of the new object is marked as missing (NA) for observation points that are in the union, but not in the original observation grid.

Methods (by generic)

- irregFunData(argvals = list, X = list): Constructor for irregular functional data objects.
- show(irregFunData): Print basic information about the irregFunData object in the console. The default console output for irregFunData objects.
- names(irregFunData): Get the names of the irregFunData object.
- names(irregFunData) <- value: Set the names of the irregFunData object.
- str(irregFunData): A str method for irregFunData objects, giving a compact overview of the structure.
- summary(irregFunData): A summary method for irregFunData objects.

Functions

- irregFunData(): Constructor for irregular functional data objects

Slots

argvals A list of numerics, representing the observation grid T_i for each realization X_i of X.

X A list of numerics, representing the values of each observation X_i of X on the corresponding observation points T_i.

Warning

Currently, the class is implemented only for functional data on one-dimensional domains $\mathcal{T} \subset \mathrm{IR}$.

See Also

funData, multiFunData

Examples

```
# Construct an irregular functional data object
i1 <- irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
# Display in the console
i1
# Summarize
summary(i1)

# A more realistic object
argvals <- seq(0,2*pi, 0.01)
ind <- replicate(11, sort(sample(1:length(argvals), sample(5:10,1)))) # sample observation points
argvalsIrreg <- lapply(ind, function(i){argvals[i]})
i2 <- irregFunData(argvals = argvalsIrreg, X = mapply(function(x, a){a * sin(x)},
            x = argvalsIrreg, a = seq(0.75, 1.25, by = 0.05)))
# Display/summary gives basic information
i2
summary(i2)
# Use the plot function to get an impression of the data
plot(i2)
```

Math.funData *Mathematical operations for functional data objects*

Description

These functions allow to apply mathematical operations (such as $exp(), log(), sin(), cos()$ or $abs()$ to functional data objects based on Math. The operations are made pointwise for each observation.

Usage

```
## S4 method for signature 'funData'
Math(x)

## S4 method for signature 'multiFunData'
Math(x)

## S4 method for signature 'irregFunData'
Math(x)
```

Arguments

x   An object of class funData, irregFunData or multiFunData.

Value

An object of the same functional data class as x.

See Also

funData, irregFunData, multiFunData, Math

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(1,2))

# simulate a funData object on 0..1 with 10 observations
argvals <- seq(0, 1, 0.01)
f <- simFunData(argvals = argvals, N = 10,
                M = 5, eFunType = "Fourier", eValType = "linear")$simData

### FunData
plot(f, main = "Original data")
plot(abs(f), main = "Absolute values")

### Irregular
# create an irrgFunData object by sparsifying f
i <- as.irregFunData(sparsify(f, minObs = 5, maxObs = 10))

plot(i, main = "Sparse data")
plot(cumsum(i), main = "'cumsum' of sparse data")

### Multivariate
m <- multiFunData(f, -1*f)
plot(m, main = "Multivariate Data")
plot(exp(m), main = "Exponential")

par(oldpar)
```

---

meanFunction   *Mean for functional data*

Description

This function calculates the pointwise mean function for objects of class funData, irregFunData or multiFunData.

Usage

```
meanFunction(object, na.rm = FALSE)
```

Arguments

object   An object of class funData, irregFunData or multiFunData.

na.rm   Logical. If TRUE, NA values are removed before computing the mean. Defaults
              to FALSE.

Value

An object of the same class as object with one observation that corresponds to the pointwise mean function of the functions in object.

Warning

If object is of class irregFunData, the option na.rm = TRUE is not implemented and throws an error. If na.rm = FALSE, the functions must be observed on the same domain.

See Also

[funData](), [irregFunData](), [multiFunData](), [Arith.funData]()

Examples

```
### Univariate (one-dimensional support)
x <- seq(0, 2*pi, 0.01)
f1 <- funData(x, outer(seq(0.75, 1.25, 0.05), sin(x)))

plot(f1)
plot(meanFunction(f1), col = 1, lwd = 2, add = TRUE)

### Univariate (two-dimensional support)
f2 <- funData(list(1:5, 1:3), array(rep(1:5,each = 11, times = 3), dim = c(11,5,3)))
all.equal(f2[1], meanFunction(f2)) # f2 has 11 identical observations

### Multivariate
m1 <- multiFunData(f1,f2)
all.equal(m1[6], meanFunction(m1)) # observation 6 equals the pointwise mean
```

```
### Irregular
i1 <- irregFunData(argvals = list(1:3,1:3,1:3), X = list(1:3,2:4,3:5))
all.equal(meanFunction(i1), i1[2])
# don't run: functions are not defined on the same domain
## Not run: meanFunction(irregFunData(argvals = list(1:3,1:5), X = list(1:3,1:5)))
```

---

multiFunData-class   *A class for multivariate functional data*

Description

The `multiFunData` class represents multivariate functional data on (potentially) different domains, i.e. a multivariate functional data object is a vector of (univariate) functional data objects, just as a vector in IR^n is a vector of n scalars. In this implementation, a `multiFunData` object is represented as a list of univariate `funData` objects, see Details.

Usage

```
multiFunData(...)

## S4 method for signature 'ANY'
multiFunData(...)

## S4 method for signature 'multiFunData'
names(x)

## S4 replacement method for signature 'multiFunData'
names(x) <- value

## S4 method for signature 'multiFunData'
str(object, ...)

## S4 method for signature 'multiFunData'
summary(object, ...)
```

Arguments

...	A list of funData objects or several funData objects passed as one argument, each. See Details.
x	The `multiFunData` object.
value	The names to be given to the `multiFunData` curves.
object	A `multiFunData` object.

Details

A `multiFunData` object is represented as a list of univariate `funData` objects, each having a `argvals` and X slot, representing the x-values and the observed y-values (see the [funData](#) class). When constructing a `multiFunData` object, the elements can be supplied as a list of `funData` objects or can be passed directly as arguments to the constructor function.

Most functions implemented for the [funData](#) class are also implemented for `multiFunData` objects. In most cases, they simply apply the corresponding univariate method to each element of the multivariate object and return it as a vector (if the result of the univariate function is scalar, such as [dimSupp](#)) or as a `multiFunData` object (if the result of the univariate function is a `funData` object, such as [extractObs](#)).

The norm of a multivariate functional data $f = (f_1, \ldots, f_p)$ is defined as

$$|||f||| := \left(\sum_{j=1}^{p} ||f_j||^2 \right)^{1/2}.$$

A `funData` object can be coerced to a `multiFunData` object with one element using `as.multiFunData(funDataObject)`.

Methods (by generic)

- `multiFunData(ANY)`: Constructor for multivariate functional data objects.
- `names(multiFunData)`: Get the names of the `multiFunData` object.
- `names(multiFunData) <- value`: Set the names of the `multiFunData` object.
- `str(multiFunData)`: A `str` method for `multiFunData` objects, giving a compact overview of the structure.
- `summary(multiFunData)`: A summary method for `multiFunData` objects.

Functions

- `multiFunData()`: Constructor for multivariate functional data objects

See Also

[funData](#)

Examples

```
### Creating a multifunData object with 2 observations on the same domain
# Univariate elements
x <- 1:5
f1 <- funData(x, rbind(x, x+1))
f2 <- funData(x,rbind(x^2, sin(x)))
# Basic
m1 <- new("multiFunData", list(f1,f2))
# Using the constructor, passing the elements as list
m2 <- multiFunData(list(f1,f2))
# Using the constructor, passing the elements directly
m3 <- multiFunData(f1,f2)
```

```
# Test if all the same
all.equal(m1,m2)
all.equal(m1,m3)
# Display multiFunData object in the console
m3
# Summarize
summary(m3)

### Creating a multifunData object with 2 observations on different domains (both 1D)
# A new element
y <- 1:3
g1 <- funData(y, rbind(3*y, y+4))
# Create the multiFunData object
m4 <- multiFunData(f1,g1)
# Display multiFunData object in the console
m4

### Creating a multifunData object with 2 observations on different domains (1D and 2D)
# A new element
y <- 1:3; z <- 1:4
g2 <- funData(list(y,z), array(rnorm(24), dim = c(2,3,4)))
# Create the multiFunData object
m5 <- multiFunData(f1,g2)
# Display multiFunData object in the console
m5

### A more realistic object
# element 1
x <- seq(0,2*pi, 0.01)
f1 <- funData(x, outer(seq(0.75, 1.25, length.out = 6), sin(x)))
# element 2
y <- seq(-1,1, 0.01); z <- seq(-0.5, 0.5, 0.01)
X2 <- array(NA, c(6, length(y), length(z)))
for(i in 1:6) X2[i,,] <- outer(y, z, function(x,y){sin(i*pi*y)*cos(i*pi*z)})
f2 <- funData(list(y,z), X2)
# MultiFunData Object
m6 <- multiFunData(f1,f2)
# Display multiFunData object in the console for basic information
m6
# Summarize
summary(m6)
# Use the plot function to get an impression of the data
## Not run: plot(m6) # m6 has 2D element, must specify one observation for plotting
plot(m6, obs = 1, main = c("1st element (obs 1)", "2nd element (obs 1)"))
plot(m6, obs = 6, main = c("1st element (obs 6)", "2nd element (obs 6)"))
```

---

nObs   *Get the number of observations*

Description

This functions returns the number of observations in a `funData`, `irregFunData` or `multiFunData` object.

Usage

```
nObs(object)
```

Arguments

object   An object of class `funData`, `irregFunData` or `multiFunData`.

Value

The number of observations in `object`.

See Also

[funData](), [irregFunData](), [multiFunData]()

Examples

```
# Univariate
object <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
nObs(object)

# Univariate (irregular)
irregObject <- irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
nObs(irregObject)

# Multivariate
multiObject <- multiFunData(object, funData(argvals = 1:3, X = rbind(3:5, 6:8)))
nObs(multiObject)
```

---

nObsPoints   *Get the number of observation points*

Description

This functions returns the number of observation points in an object of class `funData`, `multiFunData` or `irregFunData`.

Usage

```
nObsPoints(object)
```

Arguments

object   An object of class `funData`, `multiFunData` or `irregFunData`.

Details

Depending on the class of object, the function returns different values:

- If object is of class funData, the function returns a vector of length dimSupp(object), giving the number of observations in each dimension.
- If object is of class multiFunData, the function returns a list of the same length as object, where the j-th entry is a vector, corresponding to the observations point of object[[j]].
- If object is of class irregFunData, the function returns an array of length nObs(object), where the j-th entry corresponds to the number of observations in the j-th observed function.

Value

The number of observation points in object. See Details.

Warning

Do not confound with [nObs](#), which returns the number of observations (i.e. the number of observed functions) in an object of a functional data class.

See Also

[irregFunData](#), [extractObs](#)

Examples

```
# Univariate (one-dimensional)
object1 <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
nObsPoints(object1)

# Univariate (two-dimensional)
object2 <- funData(argvals = list(1:5, 1:6), X = array(1:60, dim = c(2, 5, 6)))
nObsPoints(object2)

# Multivariate
multiObject <- multiFunData(object1, object2)
nObsPoints(multiObject)

# Univariate (irregular)
irregObject <- irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
nObsPoints(irregObject)
```

---

norm   *Calculate the norm of functional data*

Description

This function calculates the norm for each observation of a funData, irregFunData or multiFunData object.

Arguments

`object`	An object of class `funData`, `irregFunData` or `multiFunData`.
`...`	Further parameters (see Details).

Details

For `funData` objects, the standard L^2 norm is calculated:

$$||f|| = \left(\int_{\mathcal{T}} f(t)^2 dt \right)^{1/2} .$$

For `irregFunData` objects, each observed function is integrated only on the observed grid points (unless `fullDom = TRUE`).

The (weighted) norm of a multivariate functional data object $f = (f_1, \ldots, f_p)$ is defined as

$$|||f||| := \left(\sum_{j=1}^{p} w_j ||f_j||^2 \right)^{1/2} .$$

Further parameters passed to this function may include:

- `squared`: Logical. If `TRUE` (default), the function calculates the squared norm, otherwise the result is not squared.
- `obs`: A numeric vector, giving the indices of the observations, for which the norm is to be calculated. Defaults to all observations.
- `method`: A character string, giving the integration method to be used. See [integrate](#) for details.
- `weight`: An optional vector of weights for the scalar product; particularly useful for multivariate functional data, where each entry can be weighted in the scalar product / norm. Defaults to 1 for each element.
- `fullDom`: Logical. If `object` is of class [irregFunData](#) and `fullDom = TRUE`, all functions are extrapolated to the same domain. Defaults to `FALSE`. See [integrate](#) for details.

Value

A numeric vector representing the norm of each observation.

Warning

The function is currently implemented only for functional data with one- and two-dimensional domains.

See Also

[funData](#), [irregFunData](#), [multiFunData](#), [integrate](#)

Examples

```
# Univariate
object <- funData(argvals = 1:5, X = rbind(1:5, 6:10))
norm(object)

# Univariate (irregular)
irregObject <- irregFunData(argvals = list(1:5, 2:4), X = list(2:6, 3:5))
norm(irregObject) # no extrapolation
norm(irregObject, fullDom = TRUE) # extrapolation (of second function)

# Multivariate
multiObject <- multiFunData(object, funData(argvals = 1:3, X = rbind(3:5, 6:8)))
norm(multiObject)
norm(multiObject, weight = c(2,1)) # with weight vector, giving more weight to the first element
```

---

plot.funData   *Plotting univariate functional data*

Description

This function plots observations of univariate functional data on their domain.

Usage

```
## S3 method for class 'funData'
plot(
  x,
  y,
  obs = seq_len(nObs(x)),
  type = "l",
  lty = 1,
  lwd = 1,
  col = NULL,
  xlab = "argvals",
  ylab = "",
  legend = TRUE,
  plotNA = FALSE,
  add = FALSE,
  ...
)

## S4 method for signature 'funData,missing'
plot(x, y, ...)
```

Arguments

x   An object of class funData.

y	Missing.
obs	A vector of numerics giving the observations to plot. Defaults to all observations in x. For two-dimensional functions (images) obs must have length 1.
type	The type of plot. Defaults to "l" (line plot). See [plot](#) for details.
lty	The line type. Defaults to 1 (solid line). See [par](#) for details.
lwd	The line width. Defaults to 1. See [par](#) for details.
col	The color of the functions. If not supplied (NULL, default value), one-dimensional functions are plotted in the [rainbow](#) palette and two-dimensional functions are plotted using [tim.colors](#) from package [fields-package](#).
xlab, ylab	The titles for x- and y-axis. Defaults to "argvals" for the x-axis and no title for the y-axis. See [plot](#) for details.
legend	Logical. If TRUE, a color legend is plotted for two-dimensional functions (images). Defaults to TRUE.
plotNA	Logical. If TRUE, missing values are interpolated using the [approxNA](#) function (only for one-dimensional functions). Defaults to FALSE.
add	Logical. If TRUE, add to current plot (only for one-dimensional functions). Defaults to FALSE.
...	Additional arguments to [matplot](#) (one-dimensional functions) or [image.plot](#)/ [image](#) (two-dimensional functions).

Details

If some observations contain missing values (coded via NA), the functions can be interpolated using the option plotNA = TRUE. This option relies on the [na.approx](#) function in package [zoo](#) and is currently implemented for one-dimensional functions only in the function [approxNA](#).

Warning

The function is currently implemented only for functional data with one- and two-dimensional domains.

See Also

[funData](#), [matplot](#), [image.plot](#), [image](#)

Examples

```
oldpar <- par(no.readonly = TRUE)

# One-dimensional
argvals <- seq(0,2*pi,0.01)
object <- funData(argvals,
                  outer(seq(0.75, 1.25, length.out = 11), sin(argvals)))

plot(object, main = "One-dimensional functional data")

# Two-dimensional
```

```
X <- array(0, dim = c(2, length(argvals), length(argvals)))
X[1,,] <- outer(argvals, argvals, function(x,y){sin((x-pi)^2 + (y-pi)^2)})
X[2,,] <- outer(argvals, argvals, function(x,y){sin(2*x*pi) * cos(2*y*pi)})
object2D <- funData(list(argvals, argvals), X)

plot(object2D, main = "Two-dimensional functional data (obs 1)", obs = 1)
plot(object2D, main = "Two-dimensional functional data (obs 2)", obs = 2)
## Not run: plot(object2D, main = "Two-dimensional functional data") # must specify obs!


### More examples ###
par(mfrow = c(1,1))

# using plotNA
if(requireNamespace("zoo", quietly = TRUE))
{
objectMissing <- funData(1:5, rbind(c(1, NA, 5, 4, 3), c(10, 9, NA, NA, 6)))
par(mfrow = c(1,2))
plot(objectMissing, type = "b", pch = 20, main = "plotNA = FALSE") # the default
plot(objectMissing, type = "b", pch = 20, plotNA = TRUE, main = "plotNA = TRUE") # requires zoo
}

# Changing colors
plot(object, main = "1D functional data in grey", col = "grey")
plot(object, main = "1D functional data in heat.colors", col = heat.colors(nObs(object)))

plot(object2D, main = "2D functional data in topo.colors", obs = 1, col = topo.colors(64))
par(oldpar)
```

---

plot.irregFunData   *Plotting irregular functional data*

Description

This function plots observations of irregular functional data on their domain.

Usage

```
## S3 method for class 'irregFunData'
plot(
  x,
  y,
  obs = seq_len(nObs(x)),
  type = "b",
  pch = 20,
  col = grDevices::rainbow(length(obs)),
  xlab = "argvals",
  ylab = "",
```

```
    xlim = range(x@argvals[obs]),
    ylim = range(x@X[obs]),
    log = "",
    add = FALSE,
    ...
)

## S4 method for signature 'irregFunData,missing'
plot(x, y, ...)
```

Arguments

x	An object of class irregFunData.
y	Missing.
obs	A vector of numerics giving the observations to plot. Defaults to all observations in x.
type	The type of plot. Defaults to "b" (line and point plot). See [plot](#) for details.
pch	The point type. Defaults to 20 (solid small circles). See [par](#) for details.
col	The color of the functions. Defaults to the [rainbow](#) palette.
xlab, ylab	The titles for x- and y-axis. Defaults to "argvals" for the x-axis and no title for the y-axis. See [plot](#) for details.
xlim, ylim	The limits for x- and y-axis. Defaults to the total range of the data that is to plot. See [plot](#) for details.
log	A character string, specifying the axis that is to be logarithmic. Can be "" (non-logarithmic axis, the default), "x", "y", "xy" or "yx". See [plot.default](#) for details. This parameter is ignored, if add = TRUE.
add	Logical. If TRUE, add to current plot (only for one-dimensional functions). Defaults to FALSE.
...	Additional arguments to [plot](#).

See Also

[plot.funData](#), [irregFunData](#), [plot](#)

Examples

```
oldpar <- par(no.readonly = TRUE)

# Generate data
argvals <- seq(0,2*pi,0.01)
ind <- replicate(5, sort(sample(1:length(argvals), sample(5:10,1))))
object <- irregFunData(argvals = lapply(ind, function(i){argvals[i]}),
                X = lapply(ind, function(i){sample(1:10,1) / 10 * argvals[i]^2}))

plot(object, main = "Irregular functional data")

par(oldpar)
```

---

plot.multiFunData   *Plotting multivariate functional data*

Description

This function plots observations of multivariate functional data on their domain. The graphic device
is split in a number of subplots (specified by dim) via mfrow ([par](#)) and the univariate elements are
plotted using plot.

Usage

```
## S3 method for class 'multiFunData'
plot(
  x,
  y,
  obs = seq_len(nObs(x)),
  dim = seq_len(length(x)),
  par.plot = NULL,
  main = names(x),
  xlab = "argvals",
  ylab = "",
  log = "",
  ylim = NULL,
  ...
)

## S4 method for signature 'multiFunData,missing'
plot(x, y, ...)
```

Arguments

x	An object of class multiFunData.
y	Missing.
obs	A vector of numerics giving the observations to plot. Defaults to all observations in x. For two-dimensional functions (images) obs must have length 1.
dim	The dimensions to plot. Defaults to length(x), i.e. all functions in x are plotted.
par.plot	Graphic parameters to be passed to the plotting regions. The option mfrow is ignored. Defaults to NULL. See [par](#) for details.
main	A string vector, giving the title of the plot. Can have the same length as dim (different titles for each dimension) or length 1 (one title for all dimensions). Defaults to names(x).
xlab, ylab	The titles for x- and y-axis. Defaults to "argvals" for the x-axis and no title for the y-axis for all elements. Can be supplied as a vector of the same length as dim (one x-/y-lab for each element) or a single string that is applied for all elements. See [plot](#) for details.

log A character string, specifying the axis that is to be logarithmic. Can be `""` (non-logarithmic axis), `"x"`, `"y"`, `"xy"` or `"yx"`. Defaults to `""` for all plots. Can be supplied as a vector of the same length as `dim` (one log-specification for each element) or a single string that is applied for all elements. See `plot.default` for details.

ylim Specifies the limits of the y-Axis. Can be either NULL (the default, limits are chosen automatically), a vector of length 2 (giving the minimum and maximum range for all elements at the same time) or a list of the same length as `dim` (specifying the limits for each element separately).

... Additional arguments to `plot`.

Warning

The function is currently implemented only for functional data with one- and two-dimensional domains.

See Also

[funData](), [multiFunData](), [plot.funData]()

Examples

```
oldpar <- par(no.readonly = TRUE)
argvals <- seq(0, 2*pi, 0.1)

# One-dimensional elements
f1 <- funData(argvals, outer(seq(0.75, 1.25, length.out = 11), sin(argvals)))
f2 <- funData(argvals, outer(seq(0.75, 1.25, length.out = 11), cos(argvals)))

m1 <- multiFunData(f1, f2)
plot(m1, main = c("1st element", "2nd element")) # different titles
plot(m1, main = "Multivariate Functional Data") # one title for all

# Mixed-dimensional elements
X <- array(0, dim = c(11, length(argvals), length(argvals)))
X[1,,] <- outer(argvals, argvals, function(x,y){sin((x-pi)^2 + (y-pi)^2)})
g <- funData(list(argvals, argvals), X)

m2 <- multiFunData(f1, g)
# different titles and labels
plot(m2, main = c("1st element", "2nd element"), obs = 1,
     xlab = c("xlab1", "xlab2"),
     ylab = "one ylab for all")
# one title for all
plot(m2, main = "Multivariate Functional Data", obs = 1)

## Not run: plot(m2, main = c("1st element", "2nd element")) # must specify obs!

par(oldpar)
```

scalarProduct   *Calculate the scalar product for functional data objects*

Description

This function calculates the scalar product between two objects of the class funData, irregFunData and multiFunData. For univariate functions f, g on a domain $\mathcal{T}$, the scalar product is defined as

$$\int_{\mathcal{T}} f(t)g(t)\mathrm{d}t$$

and for multivariate functions f, g on domains $\mathcal{T}_1, \ldots, \mathcal{T}_p$, it is defined as

$$\sum_{j=1}^{p} \int_{\mathcal{T}_j} f^{(j)}(t)g^{(j)}(t)\mathrm{d}t.$$

As seen in the formula, the objects must be defined on the same domain. The scalar product is calculated pairwise for all observations, thus the objects must also have the same number of observations or one object may have only one observation (for which the scalar product is calculated with all observations of the other object)). Objects of the classes funData and irregFunData can be combined, see integrate for details.

Usage

```
scalarProduct(object1, object2, ...)
```

Arguments

object1, object2

> Two objects of class funData, irregFunData or multiFunData, for that the scalar product is to be calculated.

... Additional parameters passed to integrate. For multiFunData objects, one can also pass a weight argument. See Details.

Details

For multiFunData one can pass an optional vector weight for calculating a weighted scalar product. This vector must have the same number of elements as the multiFunData objects and have to be non-negative with at least one weight that is different from 0. Defaults to 1 for each element. See also norm.

Value

A vector of length nObs(object1) (or nObs(object2), if object1 has only one observation), containing the pairwise scalar product for each observation.

See Also

integrate, norm,

Examples

```
# create two funData objectw with 5 observations on [0,1]
f <- simFunData(N = 5, M = 7, eValType = "linear",
                eFunType = "Fourier", argvals = seq(0,1,0.01))$simData
g <- simFunData(N = 5, M = 4, eValType = "linear",
                eFunType = "Poly", argvals = seq(0,1,0.01))$simData

# calculate the scalar product
scalarProduct(f,g)

# the scalar product of an object with itself equals the squared norm
all.equal(scalarProduct(f,f), norm(f, squared = TRUE))

# This works of course also for multiFunData objects...
m <- multiFunData(f,g)
all.equal(scalarProduct(m,m), norm(m, squared = TRUE))

# ...and for irregFunData objects
i <- as.irregFunData(sparsify(f, minObs = 5, maxObs = 10))
all.equal(scalarProduct(i,i), norm(i, squared = TRUE))

# Scalar product between funData and irregFunData objects
scalarProduct(i,f)

# Weighted scalar product for multiFunData objects
scalarProduct(m,m, weight = c(1,2))
```

---

simFunData	*Simulate univariate functional data*

Description

This functions simulates (univariate) functional data $f_1, \ldots, f_N$ based on a truncated Karhunen-Loeve representation:

$$f_i(t) = \sum_{m=1}^{M} \xi_{i,m} \phi_m(t).$$

on one- or higher-dimensional domains. The eigenfunctions (basis functions) $\phi_m(t)$ are generated using eFun, the scores $\xi_{i,m}$ are simulated independently from a normal distribution with zero mean and decreasing variance based on the eVal function. For higher-dimensional domains, the eigenfunctions are constructed as tensors of marginal orthonormal function systems.

Usage

```
simFunData(argvals, M, eFunType, ignoreDeg = NULL, eValType, N)
```

Arguments

argvals   A numeric vector, containing the observation points (a fine grid on a real in-
               terval) of the functional data that is to be simulated or a list of the marginal
               observation points.

M   An integer, giving the number of univariate basis functions to use. For higher-
               dimensional data, M is a vector with the marginal number of eigenfunctions. See
               Details.

eFunType   A character string specifying the type of univariate orthonormal basis functions
               to use. For data on higher-dimensional domains, eFunType can be a vector,
               specifying the marginal type of eigenfunctions to use in the tensor product. See
               [eFun](#) for details.

ignoreDeg   A vector of integers, specifying the degrees to ignore when generating the uni-
               variate orthonormal bases. Defaults to NULL. For higher-dimensional data, ignoreDeg
               can be supplied as list with vectors for each marginal. See [eFun](#) for details.

eValType   A character string, specifying the type of eigenvalues/variances used for the
               generation of the simulated functions based on the truncated Karhunen-Loeve
               representation. See [eVal](#) for details.

N   An integer, specifying the number of multivariate functions to be generated.

Value

simData   A [funData](#) object with N observations, representing the simulated functional
               data.

trueFuns   A [funData](#) object with M observations, representing the true eigenfunction basis
               used for simulating the data.

trueVals   A vector of numerics, representing the true eigenvalues used for simulating the
               data.

See Also

[funData](#), [eFun](#), [eVal](#), [addError](#), [sparsify](#)

Examples

```
oldPar <- par(no.readonly = TRUE)

# Use Legendre polynomials as eigenfunctions and a linear eigenvalue decrease
test <- simFunData(seq(0,1,0.01), M = 10, eFunType = "Poly", eValType = "linear", N = 10)

plot(test$trueFuns, main = "True Eigenfunctions")
plot(test$simData, main = "Simulated Data")

# The use of ignoreDeg for eFunType = "PolyHigh"
test <- simFunData(seq(0,1,0.01), M = 4, eFunType = "Poly", eValType = "linear", N = 10)
test_noConst <-  simFunData(seq(0,1,0.01), M = 4, eFunType = "PolyHigh",
                            ignoreDeg = 1, eValType = "linear", N = 10)
test_noLinear <-  simFunData(seq(0,1,0.01), M = 4, eFunType = "PolyHigh",
                             ignoreDeg = 2, eValType = "linear", N = 10)
```

```
test_noBoth <-  simFunData(seq(0,1,0.01), M = 4, eFunType = "PolyHigh",
                            ignoreDeg = 1:2, eValType = "linear", N = 10)

par(mfrow = c(2,2))
plot(test$trueFuns, main = "Standard polynomial basis (M = 4)")
plot(test_noConst$trueFuns, main = "No constant basis function")
plot(test_noLinear$trueFuns, main = "No linear basis function")
plot(test_noBoth$trueFuns, main = "Neither linear nor constant basis function")

# Higher-dimensional domains
simImages <- simFunData(argvals = list(seq(0,1,0.01), seq(-pi/2, pi/2, 0.02)),
             M = c(5,4), eFunType = c("Wiener","Fourier"), eValType = "linear", N = 4)
for(i in 1:4)
   plot(simImages$simData, obs = i, main = paste("Observation", i))

par(oldPar)
```

---

simMultiFunData   *Simulate multivariate functional data*

Description

This function provides a unified simulation structure for multivariate functional data $f_1, \ldots, f_N$ on one- or two-dimensional domains, based on a truncated multivariate Karhunen-Loeve representation:

$$f_i(t) = \sum_{m=1}^{M} \rho_{i,m} \psi_m(t).$$

The multivariate eigenfunctions (basis functions) ψ_m are constructed from univariate orthonormal bases. There are two different concepts for the construction, that can be chosen by the parameter type: A split orthonormal basis (split, only one-dimensional domains) and weighted univariate orthonormal bases (weighted, one- and two-dimensional domains). The scores $\rho_{i,m}$ in the Karhunen-Loeve representation are simulated independently from a normal distribution with zero mean and decreasing variance. See Details.

Usage

```
simMultiFunData(type, argvals, M, eFunType, ignoreDeg = NULL, eValType, N)
```

Arguments

type	A character string, specifying the construction method for the multivariate eigen-functions (either "split" or "weighted"). See Details.
argvals	A list, containing the observation points for each element of the multivariate functional data that is to be simulated. The length of argvals determines the number of elements in the resulting simulated multivariate functional data. See Details.

M	An integer (type = "split") or a list of integers (type = "weighted"), giving the number of univariate basis functions to use. See Details.
eFunType	A character string (type = "split") or a list of character strings (type = "weighted"), specifying the type of univariate orthonormal basis functions to use. See Details.
ignoreDeg	A vector of integers (type = "split") or a list of integer vectors (type = "weighted"), specifying the degrees to ignore when generating the univariate orthonormal bases. Defaults to NULL. See Details.
eValType	A character string, specifying the type of eigenvalues/variances used for the simulation of the multivariate functions based on the truncated Karhunen-Loeve representation. See eVal for details.
N	An integer, specifying the number of multivariate functions to be generated.

Details

The parameter type defines how the eigenfunction basis for the multivariate Karhunen-Loeve representation is constructed:

- type = "split": The basis functions of an underlying 'big' orthonormal basis are split in M parts, translated and possibly reflected. This yields an orthonormal basis of multivariate functions with M elements. This option is implemented only for one-dimensional domains.

- type = "weighted": The multivariate eigenfunction basis consists of weighted univariate orthonormal bases. This yields an orthonormal basis of multivariate functions with M elements. For data on two-dimensional domains (images), the univariate basis is constructed as a tensor product of univariate bases in each direction (x- and y-direction).

Depending on type, the other parameters have to be specified as follows:

Split 'big' orthonormal basis: The parameters M (integer), eFunType (character string) and ignoreDeg (integer vector or NULL) are passed to the function eFun to generate a univariate orthonormal basis on a 'big' interval. Subsequently, the basis functions are split and translated, such that the j-th part of the split function is defined on the interval corresponding to argvals[[j]]. The elements of the multivariate basis functions are given by these split parts of the original basis functions multiplied by a random sign $\sigma_j \in \{-1, 1\}, j = 1, \ldots, p.$

Weighted orthonormal bases: The parameters argvals, M, eFunType and ignoreDeg are all lists of a similar structure. They are passed element-wise to the function eFun to generate orthonormal basis functions for each element of the multivariate functional data to be simulated. In case of bivariate elements (images), the corresponding basis functions are constructed as tensor products of orthonormal basis functions in each direction (x- and y-direction).

If the j-th element of the simulated data should be defined on a one-dimensional domain, then

- argvals[[j]] is a list, containing one vector of observation points.
- M[[j]] is an integer, specifying the number of basis functions to use for this entry.
- eFunType[[j]] is a character string, specifying the type of orthonormal basis functions to use for this entry (see eFun for possible options).
- ignoreDeg[[j]] is a vector of integers, specifying the degrees to ignore when constructing the orthonormal basis functions. The default value is NULL.

If the j-th element of the simulated data should be defined on a two-dimensional domain, then

- `argvals[[j]]` is a list, containing two vectors of observation points, one for each direction (observation points in x-direction and in y-direction).
- `M[[j]]` is a vector of two integers, giving the number of basis functions for each direction (x- and y-direction).
- `eFunType[[j]]` is a vector of two character strings, giving the type of orthonormal basis functions for each direction (x- and y-direction, see `eFun` for possible options). The corresponding basis functions are constructed as tensor products of orthonormal basis functions in each direction.
- `ignoreDeg[[j]]` is a list, containing two integer vectors that specify the degrees to ignore when constructing the orthonormal basis functions in each direction. The default value is `NULL`.

The total number of basis functions (i.e. the product of `M[[j]]` for all `j`) must be equal!

Value

`simData`	A `multiFunData` object with `N` observations, representing the simulated multivariate functional data.
`trueFuns`	A `multiFunData` object with `M` observations, representing the multivariate eigenfunction basis used for simulating the data.
`trueVals`	A vector of numerics, representing the eigenvalues used for simulating the data.

References

C. Happ, S. Greven (2018): Multivariate Functional Principal Component Analysis for Data Observed on Different (Dimensional) Domains. Journal of the American Statistical Association, 113(522): 649-659.

See Also

`multiFunData`, `eFun`, `eVal`, `simFunData`, `addError`, `sparsify`.

Examples

```
oldPar <- par(no.readonly = TRUE)

# split
split <- simMultiFunData(type = "split", argvals = list(seq(0,1,0.01), seq(-0.5,0.5,0.02)),
                 M = 5, eFunType = "Poly", eValType = "linear", N = 7)

par(mfrow = c(1,2))
plot(split$trueFuns, main = "Split: True Eigenfunctions", ylim = c(-2,2))
plot(split$simData, main = "Split: Simulated Data")

# weighted (one-dimensional domains)
weighted1D <- simMultiFunData(type = "weighted",
                 argvals = list(list(seq(0,1,0.01)), list(seq(-0.5,0.5,0.02))),
                 M = c(5,5), eFunType = c("Poly", "Fourier"), eValType = "linear", N = 7)

plot(weighted1D$trueFuns, main = "Weighted (1D): True Eigenfunctions", ylim = c(-2,2))
```

```
plot(weighted1D$simData, main = "Weighted (1D): Simulated Data")

# weighted (one- and two-dimensional domains)
weighted <- simMultiFunData(type = "weighted",
             argvals = list(list(seq(0,1,0.01), seq(0,10,0.1)), list(seq(-0.5,0.5,0.01))),
                M = list(c(5,4), 20), eFunType = list(c("Poly", "Fourier"), "Wiener"),
                eValType = "linear", N = 7)

plot(weighted$trueFuns, main = "Weighted: True Eigenfunctions (m = 2)", obs = 2)
plot(weighted$trueFuns, main = "Weighted: True Eigenfunctions (m = 15)", obs = 15)
plot(weighted$simData, main = "Weighted: Simulated Data (1st observation)", obs = 1)
plot(weighted$simData, main = "Weighted: Simulated Data (2nd observation)", obs = 2)

par(oldPar)
```

---

sparsify   *Generate a sparse version of functional data objects*

Description

This function generates an artificially sparsified version of a functional data object of class `funData` (univariate) or `multiFunData` (multivariate). The minimal and maximal number of observation points for all observations can be supplied by the user.

Usage

```
sparsify(funDataObject, minObs, maxObs)
```

Arguments

funDataObject   A functional data object of class `funData` or `multiFunData`.

minObs, maxObs The minimal/maximal number of observation points. Must be a scalar for uni-
                variate functional data (`funData` class) or a vector of the same length as `funDataObject`
                for multivariate functional data (`multiFunData` class), giving the minimal/maximal
                number of observations for each element. See Details.

Details

The technique for artificially sparsifying the data is as described in Yao et al. (2005): For each element $x_i^{(j)}$ of an observed (multivariate) functional data object x_i, a random number $R_i^{(j)} \in \{\mathrm{minObs}, \ldots, \mathrm{maxObs}\}$ of observation points is generated. The points are sampled uniformly from the full grid $\{t_{j,1}, \ldots, t_{j,S_j}\} \subset \mathcal{T}_j$, resulting in observations

$$x_{i,r}^{(j)} = x_i^{(j)}(t_{j,r}), \quad r = 1, \ldots, R_i^{(j)}, \, j = 1, \ldots, p.$$

Value

An object of the same class as `funDataObject`, which is a sparse version of the original data.

Warning

This function is currently implemented for 1D data only.

References

Yao, F., H.-G. Mueller and J.-L. Wang (2005): Functional Data Analysis for Sparse Longitudinal Data. Journal of the American Statistical Association, 100 (470), 577–590.

See Also

funData, multiFunData, simFunData, simMultiFunData, addError.

Examples

```
oldPar <- par(no.readonly = TRUE)
par(mfrow = c(1,1))
set.seed(1)

# univariate functional data
full <- simFunData(argvals = seq(0,1, 0.01), M = 10, eFunType = "Fourier",
                   eValType = "linear", N = 3)$simData
sparse <- sparsify(full, minObs = 4, maxObs = 10)

plot(full, main = "Sparsify")
plot(sparse, type = "p", pch = 20, add = TRUE)
legend("topright", c("Full", "Sparse"), lty = c(1, NA), pch = c(NA, 20))

# Multivariate
full <- simMultiFunData(type = "split", argvals = list(seq(0,1, 0.01), seq(-.5,.5, 0.02)),
                        M = 10, eFunType = "Fourier", eValType = "linear", N = 3)$simData
sparse <- sparsify(full, minObs = c(4, 30), maxObs = c(10, 40))

par(mfrow = c(1,2))
plot(full[[1]], main = "Sparsify (multivariate)", sub = "minObs = 4, maxObs = 10")
plot(sparse[[1]], type = "p", pch = 20, add = TRUE)

plot(full[[2]], main = "Sparsify (multivariate)", sub = "minObs = 30, maxObs = 40")
plot(sparse[[2]], type = "p", pch = 20, add = TRUE)
legend("bottomright", c("Full", "Sparse"), lty = c(1, NA), pch = c(NA, 20))

par(oldPar)
```

---

tensorProduct   *Tensor product for univariate functions on one-dimensional domains*

Description

This function calculates tensor product functions for up to three objects of class `funData` defined on one-dimensional domains.

Usage

```
tensorProduct(...)
```

Arguments

    ...   Two or three objects of class funData, that must be defined on a one-dimensional domain, each.

Value

An object of class as funData that corresponds to the tensor product of the input functions.

Warning

The function is only implemented for up to three functions on one-dimensional domains.

See Also

[funData](#)

Examples

```
### Tensor product of two functional data objects
x <- seq(0, 2*pi, 0.1)
f1 <- funData(x, outer(seq(0.75, 1.25, 0.1), sin(x)))
y <- seq(-pi, pi, 0.1)
f2 <- funData(y, outer(seq(0.25, 0.75, 0.1), sin(y)))

plot(f1, main = "f1")
plot(f2, main = "f2")

tP <- tensorProduct(f1, f2)
dimSupp(tP)
plot(tP, obs = 1)

### Tensor product of three functional data objects
z <- seq(-1, 1, 0.05)
f3 <- funData(z, outer(seq(0.75, 1.25, 0.1), z^2))

plot(f1, main = "f1")
plot(f2, main = "f2")
plot(f3, main = "f3")

tP2 <- tensorProduct(f1, f2, f3)
dimSupp(tP2)
```

Index