Package: DistributionalDDA (via r-universe)

October 20, 2026

Type Package

Title Distributional Data Analysis in Bayes Spaces

Version 0.1.1

Maintainer K. Gerald van den Boogaart <support@boogaart.de>

Description Statistical analysis of data sets whose observations are whole probability distributions, using Bayes space calculus. Distributions (provided as samples, histograms or density functions) are treated as elements of a Hilbert space, so that operations such as mean, variance, distance, principal component analysis and clustering carry over to densities. Reference measures defining the geometry of the space can be chosen and combined. The methodology was first published with peer review by van den Boogaart, Egozcue and Pawlowsky-Glahn (2014) <doi:10.1111/anzs.12074>.

License LGPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1), compositions, splines

Imports fda, methods, graphics, stats, utils, rlang, digest, MASS, Matrix, grDevices

Suggests knitr, igraph, ggraph, rmarkdown, testthat (>= 3.0.0)

Collate DistributionalDDA-package.R topics.R helpers.R integration.R ListData.R classes.R converters.R Bayes_spaces.R Cartesian.R operations.R summaries.R plots.R simulations.R datasets.R linear_models.R PCA.R bulding_blocks.R statistics.R

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

RoxygenNote 7.3.3

NeedsCompilation no

Author Ivana Pavlu [aut] (ORCID: <https://orcid.org/0000-0001-9322-246X>), Raimon Tolosana-Delgado [aut] (ORCID: <https://orcid.org/0000-0001-9847-0462>), Matthias Templ [aut] (ORCID: <https://orcid.org/0000-0002-8638-5276>), K. Gerald van den Boogaart [aut, cre] (ORCID: <https://orcid.org/0000-0003-4646-943X>)

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

Date/Publication 2026-10-20 15:10:02 UTC

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

RemoteRef HEAD

RemoteSha 981e91890e4a9a20dc0aaca4634ce35c025d3dc2

Contents

.DensityFromHist . . . 5
.LogDensityFromHist . . . 6
[.BDataHistograms . . . 7
%.% . . . 8
+.BData . . . 9
Add . . . 10
addFrequency . . . 13
age . . . 15
as.BayesSpaceBasis . . . 17
as.BDataDensities . . . 18
as.BDataDensityHistograms . . . 20
as.BDataDensityHistograms.list . . . 21
as.BDataFrequencyHistograms . . . 22
as.BDataFrequencyHistograms.list . . . 23
as.BDataHistograms . . . 24
as.BDataLogDensities . . . 25
as.BDataLogPDFs . . . 27
as.BDataPDFs . . . 29
as.BDataSamples . . . 31
as.BDataSplines . . . 32
B2DistHyper . . . 34
B2GaussianDistribution . . . 35
B2N01 . . . 37
basisEval . . . 38
BayesSpace . . . 39
BayesSpaceBasis . . . 39
BayesSpaceBasis.BayesSpaceCartesianProduct . . . 41
BayesSpaceBasis.function . . . 42
BayesSpaceBasis.ReferenceMeasureDiscrete . . . 44
BayesSpaceBasis.ReferenceMeasureTransformed . . . 45
BayesSpaceBasis.ReferenceMeasureUniform . . . 46

BayesSpaceBasisCartesianProduct . . . 47
BayesSpaceCartesianProduct . . . 47
BData . . . 48
BDataDensities . . . 49
BDataDensityHistograms . . . 51
BDataFrequencyHistograms . . . 52
BDataGeneral . . . 53
BDataLogDensities . . . 54
BDataSamples . . . 56
BDataSplines . . . 57
biplot.princomp.BData . . . 58
boxplot.BData . . . 61
cartesianProduct . . . 63
caseE . . . 64
caseLogRatios . . . 65
caseMeans . . . 66
caseQuantiles . . . 66
caseSds . . . 67
caseVars . . . 68
cdf2density . . . 69
cdf2Support . . . 70
cen . . . 71
centeringConstant . . . 72
centralityQuantile . . . 73
clo . . . 75
clo.BDataHistograms . . . 76
closingConstant . . . 77
clr . . . 78
computeL22BayesSpacesBasisMatrix . . . 79
computeL22BayesSpacesBasisMatrix.basisfd . . . 80
computeLimit . . . 81
CovarianceOperator . . . 81
DiscretizedFunction . . . 83
dist.BData . . . 85
distanceToCenter . . . 86
Div . . . 87
DTD . . . 89
ensureType . . . 90
expectation . . . 91
expectation.ReferenceMeasureUniform . . . 92
findJumps . . . 93
fx . . . 94
getBayesSpace . . . 95
getBayesSpace.ReferenceMeasureCartesianProduct . . . 96
getBayesSpaceBasis . . . 97
getBayesSpaceBasisDimension . . . 98
getBreaks . . . 99
getCartesianFactors . . . 100

getDimension . . . 100
getDiscretizer . . . 101
getL2Space . . . 102
getMetricTensor . . . 104
getPDF . . . 105
getReferenceMeasure . . . 106
getSample . . . 107
getSplineCoefs . . . 109
hasSameBayesSpace . . . 110
hasSameBayesSpaceBasis . . . 111
idt.BData . . . 111
isClosed . . . 112
isInDomain . . . 114
isInDomain.ReferenceMeasureUniform . . . 115
isValidBayesSpaceBasis . . . 116
isValidBayesSpaceElement . . . 116
kdeDensity . . . 118
L2Space . . . 120
listDataElementAccess . . . 122
ListDataEnv . . . 122
Ln . . . 124
makeFunctionOuter . . . 126
makeLimitStable . . . 126
mapToRealLineWithSupport . . . 127
maxFunc . . . 127
mean.BDataSplines . . . 128
median.BData . . . 129
Mul . . . 130
mvar.BData . . . 133
nCases . . . 134
norm . . . 135
numberOfCases . . . 136
pdf2Support . . . 137
plot.BayesSpaceBasis . . . 138
plot.BDataHistograms . . . 138
plot.BDataSamples . . . 139
plot.BDataSplines . . . 140
plot.princomp.BData . . . 141
plot.princomp.BData.grouped . . . 145
plot.princomp.BData.multivariate . . . 147
plot.varKernel . . . 149
plotCoefs . . . 151
plotFuncs . . . 153
predict.princomp.BData . . . 157
princomp.BData . . . 158
print.BayesSpace . . . 165
print.BDataLogDensities . . . 166
print.BDataLogPDFs . . . 167

print.BDataSplines . . . 167
print.L2Space . . . 168
print.princomp.BData.grouped . . . 168
print.princomp.BData.multivariate . . . 169
print.ReferenceMeasure . . . 170
ReferenceMeasure . . . 171
ReferenceMeasureCartesianProduct . . . 171
ReferenceMeasureDiscrete . . . 172
ReferenceMeasureTransformed . . . 173
ReferenceMeasureUniform . . . 174
sample2Support . . . 176
scalarProduct . . . 177
scalarProductOfLogDensities . . . 178
scale.BData . . . 179
screeplot.princomp.BData . . . 179
selectIntegrationMethod . . . 181
simulate.B2DistHyper . . . 182
simulate.BData . . . 183
simulate.ReferenceMeasureCartesianProduct . . . 184
simulate.ReferenceMeasureDiscrete . . . 185
simulate.ReferenceMeasureTransformed . . . 185
simulate.ReferenceMeasureUniform . . . 186
spatialDepth . . . 187
StdReferenceMeasureUnif01 . . . 188
Sub . . . 189
subBasisIndices . . . 192
summary.princomp.BData . . . 193
Support . . . 194
SupportCartesianProduct . . . 196
supportDiscretization . . . 197
toHistogram . . . 198
variationKernel . . . 200
varKernel . . . 201
weightedStats . . . 203

Index **205**

`.DensityFromHist` *Compute the density relativ to a measure from histogram weights.*

Description

The function does not normalize the density.

Usage

```
.DensityFromHist(
  weights,
  breaks,
  background = 0,
  dmu = function(x) rep(1, numberOfCases(x))
)
```

Arguments

weights	a value proportional to the density in a histogram
breaks	the breaks of the histogram in accending order. One more than weights.
background	the value for the density to be chosen outside of break range.
dmu	a vectorized function representing the Lebesgue density of the measure relativ to which the density should be valid. The default computes Lebesgue densities (probability density functions, PDFs)

Value

a function that can be evaluated on the support of the underlying measure (given by dmu) and provides the a density at every point.

See Also

Other .DensitFromHist: `.LogDensityFromHist()`

Examples

```
f = .DensityFromHist(1:5,0:5)
x = seq(-1,6,by=0.01)
plot(x,f(x),type="b")
```

.LogDensityFromHist *Compute the log density relativ to a measure from histogram weights.*

Description

The function does not normalize the density.

Usage

```
.LogDensityFromHist(
  weights,
  breaks,
  background = mean(weights)/(length(weights) * 10),
  dmu = function(x) rep(1, numberOfCases(x))
)
```

Arguments

weights	a value proportional to the density in a histogram
breaks	the breaks of the histogram in accending order. One more than weights.
background	a value to be added to ensure that the density is nowhere 0 and thus the log exists.
dmu	a vectorized function representing the Lebesgue density of the measure relativ to which the density should be valid. The default computes Lebesgue densities (probability density functions, PDFs)

Value

a function that can be evaluated on the support of the underlying measure (given by dmu) and provides the a density at every point.

See Also

Other .DensitFromHist: `.DensityFromHist()`

Examples

```
f = .LogDensityFromHist(1:5,0:5)
x = seq(-1,6,by=0.01)
plot(x,f(x),type="b")
```

[.BDataHistograms *Subsetting BDataHistograms*

Description

A subsetting function selecting only a part of the histograms

Usage

```
## S3 method for class 'BDataHistograms'
x[i, j, ...]
```

Arguments

x	a BDataHistograms object
i	index of rows to be extracted
j	index of the columns, typically missing for the subsetting
...	further generic parameters such das drop=FALSE

Value

A BDataHistograms object with the subset of histograms described by the basis

Examples

```
set.seed(47)
breaks = seq(-3,3,by=1)
inRange = function(x) -3<x & x<3
X = t(replicate(4,hist(Filter(inRange,rnorm(1000)),breaks=breaks,plot=FALSE)$counts))
hists = BDataFrequencyHistograms(X,breaks=breaks,basis=B2N01Basis5)
hists[1:2, ]
```

`%.%` *Operator to act on a list*

Description

The operator acts on all elements of a list and returns a list of the results. The right hand side can be a name in which case the '$' operator is used to e

Usage

```
X %.% expr

X %..% expr
```

Arguments

`X`	the list on which to operate
`expr`	an expression, which is either a single name, in which case a the correponding '$'-operator is applied to the list, or a full expression, in which '.' takes the role of the selected element.

Value

a list, with results of the operations on the individual list elements.

Functions

- `X %..% expr`: same as '

Examples

```
data(DTD)
DTD$data %.% shape
DTD$data %.% summary(.$shape)
DTD$data %..% summary(.$shape)
```

`+.BData` *Operations for BData objects*

Description

BData objects are considered a elements of a BayesSpace and operated accordingly to Bayes space rules, using the functions [Add], [Sub], [Mul], [Div]. For operating with a scalar, please use the BData object first.

Usage

```
## S3 method for class 'BData'
x + y

## S3 method for class 'BData'
x - y

## S3 method for class 'BData'
x * y

## S3 method for class 'BData'
x / y
```

Arguments

`x`	a BData value to be operated
`y`	another value, like e.g. another BData value or a scalar

Value

A BData object of the same class as ‘x‘, resulting from the corresponding Bayes-space operation (addition, subtraction, scalar multiplication, or scalar division).

Functions

- `-` : Bayes space subtraction divides densities
- `*` : Bayes space multiplication works only with scalars
- `/` : Bayes space division only works with scalars and leads to do the y-th root

Examples

```
X <- BDataSplines(t(1:5),B2Unif01Basis5)
X+X
```

Add *Adds functional objects*

Description

This additional is also able to add functions and BayesSpace objects. The sum of two functions is a function that results in the sum of the two results. The sum of two BayesSpace objects (in BData), is given by the BayesSpace operations, which is given by the product of the densities with respect to the reference measure.

Usage

```
Add(x, y)

## S3 method for class '`function`'
Add(x, y)

## S3 method for class 'list'
Add(x, y)

## S3 method for class 'numeric'
Add(x, y)

## S3 method for class '`function`'
Add.function(x, y)

## S3 method for class 'list'
Add.function(x, y)

## S3 method for class 'numeric'
Add.function(x, y)

## S3 method for class '`function`'
Add.numeric(x, y)

## S3 method for class 'numeric'
Add.numeric(x, y)

## S3 method for class 'numeric'
Add.list(x, y)

## S3 method for class 'list'
Add.numeric(x, y)

## S3 method for class 'list'
Add.list(x, y)
```

```
## S3 method for class '`function`'
Add.list(x, y)

## S3 method for class 'BDataDensities'
Add(x, y)

## S3 method for class 'BDataLogDensities'
Add(x, y)

## S3 method for class 'BDataSplines'
Add(x, y)

## S3 method for class 'BDataHistograms'
Add(x, y)

## S3 method for class 'BDataDensities'
Add.BDataDensities(x, y)

## S3 method for class 'BDataLogDensities'
Add.BDataLogDensities(x, y)

## Default S3 method:
Add.BDataLogDensities(x, y)

## Default S3 method:
Add.BDataDensities(x, y)

## S3 method for class 'BDataSplines'
Add.BDataSplines(x, y)

## Default S3 method:
Add.BDataSplines(x, y)

## S3 method for class 'BDataHistograms'
Add.BDataHistograms(x, y)

## S3 method for class 'BDataDensities'
Add(x, y)

## S3 method for class 'BDataSplines'
Add(x, y)

## S3 method for class 'BDataLogDensities'
Add(x, y)

## S3 method for class 'BDataPDFs'
Add(x, y)
```

```
## S3 method for class 'BDataLogPDFs'
Add(x, y)
```

Arguments

x	a functional object
y	a functional object

Value

A BData object of the same subclass as the first argument (the Bayes-space sum / difference / scalar multiple / quotient).

Methods (by class)

- `Add(`function`)`: method for functions
- `Add(list)`: method for lists, iterating over them
- `Add(numeric)`: method for numerics, considering them as constant functions
- `Add(BDataDensities)`: method for BDataDensities (effectively multiplying the densities as addition in the BayesSpace)
- `Add(BDataLogDensities)`: method for BDataLogDensities
- `Add(BDataSplines)`: method for BDataSplines
- `Add(BDataHistograms)`: method for BDataHistograms
- `Add(BDataDensities)`: BayesSpaceAddition for BDataDensities
- `Add(BDataSplines)`: BayesSpaceAddition for BDataSplines
- `Add(BDataLogDensities)`: BayesSpaceAddition for BDataLogDensities
- `Add(BDataPDFs)`: BayesSpaceAddition for BDataPDFs
- `Add(BDataLogPDFs)`: BayesSpaceAddition for BDataLogPDFs

Functions

- `Add.function(`function`)`: method for adding two functions, doing it by creating a function adding their values.
- `Add.function(list)`: method adding a function to a list, creating a list with entries for each list
- `Add.function(numeric)`: method adding a constant to a function
- `Add.numeric(`function`)`: method for adding a constant to a function
- `Add.numeric(numeric)`: method for two numerics, simply adding the values
- `Add.list(numeric)`: method for adding a numeric to every value of the list
- `Add.numeric(list)`: method for adding a numeric to every value of the list
- `Add.list(list)`: method for vectorized addition of list elements
- `Add.list(`function`)`: method for adding a function to every element of a list
- `Add.BDataDensities(BDataDensities)`: method for two BDataDensities

- `Add.BDataLogDensities(BDataLogDensities)`: method for adding two BDataLogDensities
- `Add.BDataLogDensities(default)`: method converting the second argument to a BDataLogDensity and then adding
- `Add.BDataDensities(default)`: method converting the second argument to a BDataDensity object and then adding.
- `Add.BDataSplines(BDataSplines)`: method for two BDataSplines
- `Add.BDataSplines(default)`: method for BDataSplines default
- `Add.BDataHistograms(BDataHistograms)`: method for BDataHistograms

See Also

Other Add: `Div()`, `Mul()`, `Sub()`

Examples

```
Add(sin,cos)
Add(list(sin,cos),sin)
Add(cos,list(sin,cos))
Add(list(sin,cos),list(sin,cos))
Add(list(sin,cos),10)
```

`addFrequency` *Pool absolute frequencies*

Description

Sum together the absolute frequencies of two representations with this information

Usage

```
addFrequency(x, y, ...)

## S3 method for class 'BDataFrequencyHistograms'
addFrequency.BDataFrequencyHistograms(x, y, ...)

## S3 method for class 'BDataUnivariateSamples'
addFrequency.BDataUnivariateSamples(x, y, ...)

## S3 method for class 'BDataUnivariateSamples'
addFrequency.BDataFrequencyHistograms(
  x,
  y,
  .baseSpaceBasis = getBayesSpaceBasis(x),
  ...
```

```
)

## S3 method for class 'BDataFrequencyHistograms'
addFrequency.BDataUnivariateSamples(
  x,
  y,
  .baseSpaceBasis = getBayesSpaceBasis(x),
  ...
)
```

Arguments

x	a compatible BData representation
y	another compatible BData representation
...	additional arguments (mostly internal)
.baseSpaceBasis	which base space basis to use, defaults to the one specified by the first argument ‘x‘

Value

An object of the least informative kind from the two pooled together, which contains the added absolute frequencies of both input objects. The Bayes Space basis is controlled by the first argument.

Functions

- `addFrequency.BDataFrequencyHistograms(BDataFrequencyHistograms)`: method for two "BDataFrequencyHistograms" object. Returns a "BDataFrequencyHistograms" where the frequencies are summed (if breaks are equivalent), or where they are distributed uniformly along the interval bins and rebinned by summing with a break set resulting from the combination of both input bin sets.
- `addFrequency.BDataUnivariateSamples(BDataUnivariateSamples)`: method for two "BDataUnivariateSamples" object. Returns a "BDataUnivariateSamples" where the observations are pooled together (in the order they are given), pair by pair. It can recycle a "BDataUnivariateSamples" of one single population with a multi-population "BDataUnivariateSamples", but will not work with any other combination of "BDataUnivariateSamples" of different lengths.
- `addFrequency.BDataFrequencyHistograms(BDataUnivariateSamples)`: method for a "BDataFrequencyHistograms" and a "BDataUnivariateSamples" objects; classifies the data sets from "BDataUnivariateSamples" into the bins of "BDataFrequencyHistograms". As with other methods of this function, it only works for ‘x‘ and ‘y‘ of the same length, or else either ‘x‘ or ‘y‘ must have length 1.
- `addFrequency.BDataUnivariateSamples(BDataFrequencyHistograms)`: method for a "BDataUnivariateSamples" and a "BDataFrequencyHistograms" objects; same as in the reverse order

Examples

```
xh = lapply(rep(1000,4), runif) |> as.BDataFrequencyHistograms(basis=B2Unif01Basis5)
yh = lapply(rep(1000,4), runif) |> as.BDataFrequencyHistograms(basis=B2Unif01Basis5)
```

```
addFrequency(xh,yh)
xh1 = list(runif(300)) |> as.BDataFrequencyHistograms(basis=B2Unif01Basis5)
addFrequency(xh,xh1)
xh1 = list(runif(300)) |> as.BDataFrequencyHistograms(basis=B2Unif01Basis5, breaks=c(0:4)/4)
addFrequency(xh,xh1)
xs = lapply(rep(200,4), runif) |> BDataSamples(B2Unif01Basis5)
ys = lapply(rep(10,4), runif) |> BDataSamples(B2Unif01Basis5)
addFrequency(xs,ys) |> summary()
x1 = BDataSamples(list(runif(100)), B2Unif01Basis5)
addFrequency(xs, x1)  |> summary()
xs = lapply(rep(200,4), runif) |> BDataSamples(B2Unif01Basis5)
yh = lapply(rep(1000,4), runif) |> as.BDataFrequencyHistograms(basis=B2Unif01Basis5)
addFrequency(xs,yh)
```

---

age *Age Distribution Data for Upper Austrian Communes*

Description

Age distributions for 57 communes in Upper Austria (Oberoesterreich), separated by sex. Each observation represents the age distribution of either the male or female population in a commune, given as proportions across 19 five-year age groups.

Usage

```
age
```

Format

A list with three components:

data A data frame with 114 rows (57 communes x 2 sexes) and 22 columns:

- **commune** Character. Four-digit commune code.
- **municipality** Character. Name of the municipality (e.g., "Linz", "Steyr"). German umlauts have been transliterated to ASCII (ue, oe, ae, ss).
- **sex** Factor with levels "male" and "female".
- **AGE_2, AGE_7, ..., AGE_92** Numeric. Proportions for each age group. Column names indicate the midpoint of the 5-year age interval (e.g., AGE_2 = ages 0-4, AGE_7 = ages 5-9, ..., AGE_92 = ages 90+).

ages Numeric vector of length 19. The midpoint ages: 2, 7, 12, ..., 92.

distributions A 114 x 19 matrix of age proportions. Row names are formatted as "municipality_sex" (e.g., "Linz_male"). Each row sums to 1.

Details

The data originates from the Austrian Register Census 2021 (Registerzaehlung 2021), which provides population counts by gender, age, and citizenship at the municipality level. The original counts have been converted to proportions (probability distributions) for distributional data analysis.

This dataset is useful for demonstrating PCA on distributional data, as it contains real-world probability distributions that vary in shape (e.g., rural vs urban communes have different age structures).

License

Creative Commons Attribution 4.0 International (CC BY 4.0). https://creativecommons.org/licenses/by/4.0/

Note

Municipality names have been transliterated from German to ASCII for cross-platform compatibility, following the convention ae / oe / ue / ss for the umlauts and sharp-s (so e.g. the German "Altmuenster" and "Schoenegg" are stored with ue / oe). For an unambiguous key independent of spelling, use the four-digit codes in the ‘commune‘ column.

Source

Data from the Austrian Register Census 2021 (Registerzaehlung 2021), provided by the State of Upper Austria (Land Oberoesterreich) via the Open Data Portal Austria.

References

Land Oberoesterreich (2021). Bevoelkerung nach Geschlecht und Alter. Open Data Oesterreich.
Old broken link: https://www.data.gv.at/katalog/dataset/land-ooe_bevolkerung-geschlecht-und-alter
Current dataset: https://www.data.gv.at/katalog/datasets/cd33f19a-7bcc-4454-987c-9f2d76542872

Examples

```
data("age")

# Examine the structure
str(age)
head(age$data)

# The age midpoints
age$ages

# Plot a few age distributions
matplot(age$ages, t(age$distributions[1:6, ]), type = "l",
        xlab = "Age", ylab = "Proportion",
        main = "Age distributions for first 6 observations")

# Convert to BData for distributional analysis
# First, create a basis for age distributions (support 0-100)
rmAge <- ReferenceMeasureUniform(0, 100)
```

```
B2Age <- BayesSpace(rmAge)
basisAge <- BayesSpaceBasis(B2Age, nbasis = 7)

# Create histograms from the proportional data
breaks <- c(5*(0:18), 100)
ageHist <- BDataDensityHistograms(
    age$distributions, breaks = breaks, basis = basisAge
    )

# Convert to splines: histograms -> densities -> splines
ageDensities <- as.BDataDensities(ageHist)
ageSplines <- as.BDataSplines(ageDensities)
pcaAge <- princomp(ageSplines)
summary(pcaAge)

# Visualize PCA results
plot(pcaAge, type = "screeplot")
plot(pcaAge, type = "scores")

# Score plot with municipality labels
plot(pcaAge, type = "scores", labels = age$data$municipality, cex = 0.6)

# Color by sex
sexColors <- ifelse(age$data$sex == "male", "blue", "red")
plot(pcaAge, type = "scores", col = sexColors)
legend("topright", c("Male", "Female"), col = c("blue", "red"), pch = 19)
```

`as.BayesSpaceBasis` *Finds a BayesSpaceBasis representation of the object*

Description

Finds a BayesSpaceBasis representation of the object

Usage

```
as.BayesSpaceBasis(x, ...)

## S3 method for class 'BayesSpaceBasis'
as.BayesSpaceBasis(x, ...)

## S3 method for class 'BayesSpace'
as.BayesSpaceBasis(x, ...)
```

Arguments

`x`	the object to be converted
`...`	further parameters to methods

Value

a [BayesSpaceBasis] object

Methods (by class)

- `as.BayesSpaceBasis(BayesSpaceBasis)`: returns x
- `as.BayesSpaceBasis(BayesSpace)`: Calls 'BayesSpaceBasis(x,...)' to create an appropriate basis

See Also

[BayesSpaceBasis]

Examples

```
as.BayesSpaceBasis(B2Unif01Basis5)
as.BayesSpaceBasis(B2Unif01)
```

---

as.BDataDensities *Converts to a BDataDensities object*

Description

This represents BayesSpace data by a list of density functions, i.e. one density (with respect to the reference measure) per observation. Use this representation to inspect or plot the distributions as densities; for computation (PCA, distances) convert further to splines with [as.BDataSplines].

Usage

```
as.BDataDensities(x, ...)

## S3 method for class 'BDataDensities'
as.BDataDensities(x, ...)

## S3 method for class 'BDataSplines'
as.BDataDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogDensities'
as.BDataDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataPDFs'
as.BDataDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogPDFs'
as.BDataDensities(x, ..., basis = getBayesSpaceBasis(x))
```

```
## S3 method for class 'BDataHistograms'
as.BDataDensities(
  x,
  ...,
  background = 0,
  basis = getBayesSpaceBasis(x),
  dmu = getPDF(getReferenceMeasure(basis))
)

## S3 method for class 'BDataSamples'
as.BDataDensities(
  x,
  ...,
  method = c("Fisher", "kde"),
  basis = getBayesSpaceBasis(x)
)
```

Arguments

`x`	the data to be converted
`...`	generic arguments potentially used by methods, e.g. to define a method of computation.
`basis`	the BayesSpaceBasis to be assigned to the object. Necessary if non can be infered from the information presented to the function.
`background`	the valued for the density outside of the interval.
`dmu`	the Lebesgue density of the reference measure.
`method`	a character string defining the method of computation of the density form the data. "Fisher" stands for Fisher scoring and tries to fit an exponential family model given by the subspace of the BayesSpace. It might fail for certain datasets. See [FisherScoringEstimation] for further method parameters. "kde" stands for kernel density estimation. See [kdeDensity] for further possible parameters.

Value

A BDataDensities object (densities w.r.t. the reference measure).

Methods (by class)

- `as.BDataDensities(BDataDensities)`: method for the class itself das nothing
- `as.BDataDensities(BDataSplines)`: constructs the projected functions
- `as.BDataDensities(BDataLogDensities)`: calls [BDataDensities.BDataLogDensities]
- `as.BDataDensities(BDataPDFs)`: divides PDFs by PDF of reference measure
- `as.BDataDensities(BDataLogPDFs)`: divides PDFs by PDF of reference measure
- `as.BDataDensities(BDataHistograms)`: method for histograms (TODO: Uses a very crude heuristic to estimate the log density)
- `as.BDataDensities(BDataSamples)`: constructs a density from the samples

See Also

[.DensityFromHist]

Other BayesSpaceConverters: as.BDataDensityHistograms(), as.BDataFrequencyHistograms(), as.BDataHistograms(), as.BDataLogDensities(), as.BDataLogPDFs(), as.BDataPDFs(), as.BDataSamples(), as.BDataSplines(), idt.BData(), toHistogram()

Examples

```
X = BDataLogDensities(list(sin,cos),basis=B2Unif01Basis5)
as.BDataDensities(X)
```

---

as.BDataDensityHistograms

Convert to a BDataDensityHistograms dataset

Description

Density histograms do not require count information, but provide density information about the relative density in the bars.

Usage

```
as.BDataDensityHistograms(X, ...)

## S3 method for class 'BDataUnivariateSamples'
as.BDataDensityHistograms(
  X,
  ...,
  breaks = "Sturges",
  basis = getBayesSpaceBasis(X)
)
```

Arguments

X	the object to be converted to a BDataDensityHistorams dataset
...	further parameters for the method called
breaks	a numerical vector of the histogram breaks to be used
basis	the BayesSpaceBasis to be used for the resulting object

Value

a BDataDensityHistograms object

Methods (by class)

- `as.BDataDensityHistograms(BDataUnivariateSamples)`: method for samples computing the histogram

See Also

[as.BDataDensityHistograms.list]

Other BDataHistograms: `BDataFrequencyHistograms()`, `as.BDataDensityHistograms.list()`, `as.BDataFrequencyHistograms.list()`, `getBreaks()`, `plot.BDataHistograms()`

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataFrequencyHistograms()`, `as.BDataHistograms()`, `as.BDataLogDensities()`, `as.BDataLogPDFs()`, `as.BDataPDFs()`, `as.BDataSamples()`, `as.BDataSplines()`, `idt.BData()`, `toHistogram()`

Examples

```
s = BDataSamples(list(rnorm(300), rnorm(300,1)), basis=B2N01Basis5)
as.BDataDensityHistograms(s)
```

---

as.BDataDensityHistograms.list

Build a distributional dataset as density histograms from a list

---

Description

Constructor for a distributional data set of class "BDataDensityHistograms" out of a list of either precalculated histograms or of numeric vectors.

Usage

```
## S3 method for class 'list'
as.BDataDensityHistograms(X, breaks, basis, ...)
```

Arguments

`X`	"list" of histograms or of numeric vectors
`breaks`	breaks to be used; if ‘X‘ is a list of numeric vectors, it admits, Minimum and Maximum should fit to the support of ReferenceMeasure. the same possibilities as the same argument of [graphics::hist()]
`basis`	"BayesSpaceBasis" object to use
`...`	further arguments passed to [graphics::hist()]

Value

A distributional data container of class [‘BDataDensityHistograms‘].

See Also

Other BDataHistograms: BDataFrequencyHistograms(), as.BDataDensityHistograms(), as.BDataFrequencyHistogr
getBreaks(), plot.BDataHistograms()

Examples

```
x = lapply(rep(1000,4), runif)
(bddh = as.BDataDensityHistograms(x, breaks="Sturges", basis=B2Unif01Basis5) )
(bdfh = as.BDataFrequencyHistograms(x, breaks="Sturges", basis=B2Unif01Basis5) )
as.BDataDensityHistograms(bdfh)
```

---

as.BDataFrequencyHistograms

Converts to a BDataFrequencyHistograms

Description

Only samples and BDataFrequencyHistograms themselfs can be effectively converted to histograms, because all other representations lack information on the certainty

Usage

```
as.BDataFrequencyHistograms(X, ...)

## S3 method for class 'BDataUnivariateSamples'
as.BDataFrequencyHistograms(
  X,
  breaks = "Sturges",
  ...,
  basis = getBayesSpaceBasis(X)
)
```

Arguments

X	an object representing a dataset
...	further generic paramters for [hist]
breaks	the breaks parameters of the [hist] function for R
basis	the BayesSpaceBasis marking the BayesSpace and that shall be use for if a BayesSpaceBasis is needed for an operation.

Value

a BDataFrequencyHistograms object representing the same dataset

Methods (by class)

- as.BDataFrequencyHistograms(BDataUnivariateSamples): method for BDataUnivariateSamples

See Also

Other BayesSpaceConverters: as.BDataDensities(), as.BDataDensityHistograms(), as.BDataHistograms(), as.BDataLogDensities(), as.BDataLogPDFs(), as.BDataPDFs(), as.BDataSamples(), as.BDataSplines(), idt.BData(), toHistogram()

Examples

```
Xsamples = BDataSamples(lapply(1:10,function(i) runif(100)),basis=B2Unif01Basis5)
Xhist    = as.BDataFrequencyHistograms(Xsamples)
```

---

as.BDataFrequencyHistograms.list

Build a distributional dataset as histograms from a list

Description

Constructor for a distributional data set of class "BDataFrequencyHistograms" out of a list of either precalculated histograms or of numeric vectors.

Usage

```
## S3 method for class 'list'
as.BDataFrequencyHistograms(X, breaks, basis, ...)
```

Arguments

X	"list" of histograms or of numeric vectors
breaks	breaks to be used; if ‘X‘ is a list of numeric vectors, it admits the same possibilities as the same argument of [graphics::hist()]
basis	"BayesSpaceBasis" object to use
...	further arguments passed to [graphics::hist()]

Value

A distributional data container of class [‘BDataFrequencyHistograms‘].

See Also

Other BDataHistograms: BDataFrequencyHistograms(), as.BDataDensityHistograms(), as.BDataDensityHistogram getBreaks(), plot.BDataHistograms()

Examples

```
x = lapply(rep(1000,4), runif)
as.BDataFrequencyHistograms(x, breaks="Sturges",basis=B2Unif01Basis5)
```

`as.BDataHistograms` *Represent data as a BDataHistogram (BDataDensityHistogram or BDataFrequencyHistogram*

Description

The function tries to guess whether its datasets, distributions, Histograms,...

Usage

```
as.BDataHistograms(X, ...)

## S3 method for class 'list'
as.BDataHistograms(
  X,
  ...,
  breaks = NULL,
  basis = NULL,
  nsim = NULL,
  seed = NULL
)

## S3 method for class 'BDataHistograms'
as.BDataHistograms(
  X,
  ...,
  breaks = NULL,
  basis = NULL,
  nsim = NULL,
  seed = NULL
)

## S3 method for class 'histogram'
as.BDataHistograms(X, ..., breaks = NULL, basis, nsim = NULL, seed = NULL)

## S3 method for class 'BDataFrequencyHistograms'
as.BDataHistograms(X, ...)

## S3 method for class 'BDataDensityHistograms'
as.BDataHistograms(X, ...)
```

Arguments

`X`	a list of objects (or single object) that is a numeric sample, a histogram (with or without a counts), or a BData object
`...`	further arguments to [graphics::hist]

breaks	the common breaks. If the data is provided as histograms it should already be provide with these breaks
basis	the BayesSpaceBasis to be used in the resulting BayesSpace, if required
nsim	the number of points to simulate if the data is provided as a distribution. If NULL no simulation is done.
seed	the seed to used if data is simulated (use with care)

Value

either a BDataFrequencyHistograms (if counts are availalbe) , or a BDataDensityHistograms object if only densities are known

Methods (by class)

- `as.BDataHistograms(list)`: method for lists of possibly heterogenious data
- `as.BDataHistograms(BDataHistograms)`: checks breaks if given
- `as.BDataHistograms(histogram)`: checks breaks if given
- `as.BDataHistograms(BDataFrequencyHistograms)`: identity converter
- `as.BDataHistograms(BDataDensityHistograms)`: identity converter

See Also

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataLogDensities()`, `as.BDataLogPDFs()`, `as.BDataPDFs()`, `as.BDataSamples()`, `as.BDataSplines()`, `idt.BData()`, `toHistogram()`

Examples

```
myBreaks = seq(0,1,by=0.2)
as.BDataHistograms(replicate(10,runif(100),simplify=FALSE),
                   breaks=myBreaks,basis=B2Unif01Basis5)
```

---

as.BDataLogDensities *Converts to a BDataLogDensities object*

Description

This represents BayesSpace data by a list of log-density functions (the logarithm of the density with respect to the reference measure). Log-densities are the natural representation for Bayes-space operations, where adding distributions corresponds to adding their log-densities.

Usage

```
as.BDataLogDensities(x, ...)

## S3 method for class 'BDataDensities'
as.BDataLogDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogPDFs'
as.BDataLogDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataPDFs'
as.BDataLogDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataSplines'
as.BDataLogDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogDensities'
as.BDataLogDensities(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataSamples'
as.BDataLogDensities(
  x,
  ...,
  method = c("Fisher", "kde"),
  basis = getBayesSpaceBasis(x)
)

## S3 method for class 'list'
as.BDataLogDensities(x, ..., basis)

## S3 method for class 'BDataHistograms'
as.BDataLogDensities(x, ..., basis = getBayesSpaceBasis(x))
```

Arguments

`x`	the object to be converted to a log density
`...`	further typically unused generic arguments (might be used for density inference methods)
`basis`	the BayesSpaceBasis to be used if not availabel form the converted object.
`method`	the method to compute the density, see [as.BDataDensities] for the availabel options.

Value

A BDataLogDensities object.

Methods (by class)

- `as.BDataLogDensities(BDataDensities)`: method for BDataDensities

- `as.BDataLogDensities(BDataLogPDFs)`: method for BDataLogPDFs
- `as.BDataLogDensities(BDataPDFs)`: method for BDataPDFs
- `as.BDataLogDensities(BDataSplines)`: constructs the projected functions
- `as.BDataLogDensities(BDataLogDensities)`: calls [BDataLogDensities.BDataLogDensities]
- `as.BDataLogDensities(BDataSamples)`: constructs a density from the samples
- `as.BDataLogDensities(list)`: constructs a density from the samples
- `as.BDataLogDensities(BDataHistograms)`: constructs a density from the samples (TODO: Uses a very crude heuristic to estimate the log density)

See Also

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataHistograms()`, `as.BDataLogPDFs()`, `as.BDataPDFs()`, `as.BDataSamples()`, `as.BDataSplines()`, `idt.BData()`, `toHistogram()`

Examples

```
as.BDataLogDensities(list(sin,cos),basis=B2N01)
```

`as.BDataLogPDFs` *Converts to representations as Log Probability Density functions*

Description

A probability density function or Lebesgue density of a measure can only be computed for Lebesgue continuous distributions. As a dataset of BayesSpace elements (BData) they can be encoded as BDataLogPDFs objects. This function converts other representations to this representation.

Usage

```
as.BDataLogPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataDensities'
as.BDataLogPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogDensities'
as.BDataLogPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataSplines'
as.BDataLogPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataPDFs'
as.BDataLogPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogPDFs'
```

```
as.BDataLogPDFs(x, ..., basis)

## S3 method for class 'BDataDensityHistograms'
as.BDataLogPDFs(x, ..., background = 0, basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataFrequencyHistograms'
as.BDataLogPDFs(x, ..., background = 0, basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataSamples'
as.BDataLogPDFs(
  x,
  ...,
  method = c("Fisher", "kde"),
  basis = getBayesSpaceBasis(x)
)
```

Arguments

`x`	a BData object that shall be converted to probability density function (Lebesgue density representation)
`...`	further parameters for generic use
`basis`	the BayesSpaceBasis to be used with the new object
`background`	the background density to be added in order to avoid 0 densities e.g. in case of histograms for empty classes or regions outside of the histogram breaks.
`method`	a character string defining the method of computation of the density form the data. "Fisher" stands for Fisher scoring and tries to fit an exponential family model given by the subspace of the BayesSpace. It might fail for certain datasets. See [FisherScoringEstimation] for further method parameters. "kde" stands for kernel density estimation. See [kdeDensity] for further possible parameters.

Value

a BDataLogPDFs object

Methods (by class)

- `as.BDataLogPDFs(BDataDensities)`: method for BDataDensities multiplies with reference measure density
- `as.BDataLogPDFs(BDataLogDensities)`: method for BDataLogDensities adds the log density of the reference measure
- `as.BDataLogPDFs(BDataSplines)`: method for BDataSplines
- `as.BDataLogPDFs(BDataPDFs)`: method for BDataLogPDFs exponentiates and multiplies with reference measure density
- `as.BDataLogPDFs(BDataLogPDFs)`: method for BDataLogPDFs possibly changes the basis
- `as.BDataLogPDFs(BDataDensityHistograms)`: method for BDataDensityHistograms interprets histogram as density.

- `as.BDataLogPDFs(BDataFrequencyHistograms)`: method for BDataFrequencyHistograms
- `as.BDataLogPDFs(BDataSamples)`: method for BDataSamples

See Also

[BDataLogPDFs], [as.BDataPDFs]

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataHistograms()`, `as.BDataLogDensities()`, `as.BDataPDFs()`, `as.BDataSamples()`, `as.BDataSplines()`, `idt.BData()`, `toHistogram()`

Examples

```
X <- BDataDensities(list(\(x)(1),\(x)(x)),basis=B2N01Basis5)
Xlpdfs = as.BDataLogPDFs(X)
Xlpdfs
X1 = as.BDataLogPDFs(as.BDataLogDensities(X))
X2 = as.BDataLogPDFs(as.BDataLogPDFs(X))
X3 = as.BDataLogPDFs(Xlpdfs)
set.seed(47)
Xs = BDataSamples(replicate(7,runif(200),simplify=FALSE),B2Unif01Basis5)
X4 = as.BDataLogPDFs(Xs) # TODO
X5 = as.BDataLogPDFs(as.BDataFrequencyHistograms(Xs,breaks=seq(0,1,by=0.1)))
X6 = as.BDataLogPDFs(as.BDataDensityHistograms(Xs,breaks=seq(0,1,by=0.1)^2))
# plot(Xlpdfs)
```

`as.BDataPDFs` *Converts to representations as Probability Density functions*

Description

A probability density function or Lebesgue density of a measure can only be computed for Lebesgue continuous distributions. As a dataset of BayesSpace elements (BData) they can be encoded as BDataPDFs objects. This function converts other representations to this representation.

Usage

```
as.BDataPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataDensities'
as.BDataPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogDensities'
as.BDataPDFs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataLogPDFs'
as.BDataPDFs(x, ..., basis = getBayesSpaceBasis(x))
```

```
## S3 method for class 'BDataPDFs'
as.BDataPDFs(x, ..., basis)

## S3 method for class 'BDataDensityHistograms'
as.BDataPDFs(x, ..., background = 0, basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataFrequencyHistograms'
as.BDataPDFs(x, ..., background = 0, basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataSamples'
as.BDataPDFs(
  x,
  ...,
  method = c("Fisher", "kde"),
  basis = getBayesSpaceBasis(x)
)

## S3 method for class 'BDataSplines'
as.BDataPDFs(x, ..., basis = getBayesSpaceBasis(x))
```

Arguments

`x`	a BData object that shall be converted to probability density function (Lebesgue density representation)
`...`	further parameters for generic use
`basis`	the BayesSpaceBasis to be used with the new object
`background`	the background density to be added in order to avoid 0 densities e.g. in case of histograms for empty classes or regions outside of the histogram breaks.
`method`	a character string defining the method of computation of the density form the data. "Fisher" stands for Fisher scoring and tries to fit an exponential family model given by the subspace of the BayesSpace. It might fail for certain datasets. See [FisherScoringEstimation] for further method parameters. "kde" stands for kernel density estimation. See [kdeDensity] for further possible parameters.

Value

a BDataPDFs object

Methods (by class)

- `as.BDataPDFs(BDataDensities)`: method for BDataDensities multiplies with reference measure density
- `as.BDataPDFs(BDataLogDensities)`: method for BDataLogDensities exponentiates and multiplies with reference measure density
- `as.BDataPDFs(BDataLogPDFs)`: method for BDataLogPDFs exponentiates and multiplies with reference measure density
- `as.BDataPDFs(BDataPDFs)`: method for BDataPDFs possibly changes the basis

- `as.BDataPDFs(BDataDensityHistograms)`: method for BDataDensityHistograms interprets histogram as density.
- `as.BDataPDFs(BDataFrequencyHistograms)`: method for BDataFrequencyHistograms
- `as.BDataPDFs(BDataSamples)`: method for BDataSamples
- `as.BDataPDFs(BDataSplines)`: method for BDataSplines

See Also

[BDataPDFs]

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataHistograms()`, `as.BDataLogDensities()`, `as.BDataLogPDFs()`, `as.BDataSamples()`, `as.BDataSplines()`, `idt.BData()`, `toHistogram()`

Examples

```
X <- BDataDensities(list(\(x)(1),\(x)(x)),basis=B2N01Basis5)
Xpdfs = as.BDataPDFs(X)
Xpdfs
X1 = as.BDataPDFs(as.BDataLogDensities(X))
X2 = as.BDataPDFs(as.BDataLogPDFs(X))
X3 = as.BDataPDFs(Xpdfs)
set.seed(47)
Xs = BDataSamples(replicate(7,runif(200),simplify=FALSE),B2Unif01Basis5)
# X4 = as.BDataPDFs(Xs) # TODO
X5 = as.BDataPDFs(as.BDataFrequencyHistograms(Xs,breaks=seq(0,1,by=0.1)))
X6 = as.BDataPDFs(as.BDataDensityHistograms(Xs,breaks=seq(0,1,by=0.1)^2))
# plot(Xpdfs)
```

`as.BDataSamples` *Converts to representations as BDataSamples*

Description

Convert any representation to "BDataSamples", most commonly by simulating the data by means of [simulate.BData()].

Usage

```
as.BDataSamples(x, ...)

## S3 method for class 'BData'
as.BDataSamples(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataSamples'
as.BDataSamples(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BDataUnivariateSamples'
as.BDataSamples(x, ..., basis = getBayesSpaceBasis(x))
```

Arguments

x	a BData object that shall be converted to a collection of samples
...	further parameters for generic use, in particular ‘nsim‘ may be useful
basis	the BayesSpaceBasis to be used with the new object

Value

a BDataSamples object

Methods (by class)

- `as.BDataSamples(BData)`: for any "BData" object
- `as.BDataSamples(BDataSamples)`: for "BDataSamples" object, as identity or to change the basis
- `as.BDataSamples(BDataUnivariateSamples)`: for "BDataUnivariateSamples" object, as identity or to change the basis

See Also

[simulate.BData]

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataHistograms()`, `as.BDataLogDensities()`, `as.BDataLogPDFs()`, `as.BDataPDFs()`, `as.BDataSplines()`, `idt.BData()`, `toHistogram()`

---

as.BDataSplines *Convert distributional data to a spline representation*

Description

Returns a dataset of densities represented by zero-integral B-spline (ZB-spline) coefficients in a [BDataSplines] object. The spline representation is the compact form used for computation (PCA, distances, inner products). The conversion relies on a [BayesSpaceBasis] that is either taken from the object or supplied via ‘basis‘.

Usage

```
as.BDataSplines(X, ...)

## S3 method for class 'BDataLogDensities'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BDataLogPDFs'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BDataPDFs'
```

```
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BDataDensities'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BDataSplines'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BDataSamples'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BDataGeneral'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class '`function`'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'histogram'
as.BDataSplines(X, ..., basis = getBayesSpaceBasis(X))

## S3 method for class 'BayesSpaceBasis'
as.BDataSplines(X, ...)
```

Arguments

`X`	a dataset of densities/distributions in any form
`...`	further arguments passed to the method supporting conversion
`basis`	"BayesSpaceBasis" object to use

Value

A distributional data container of class [‘BDataSplines‘].

Methods (by class)

- `as.BDataSplines(BDataLogDensities)`: method for BDataLogDensities
- `as.BDataSplines(BDataLogPDFs)`: method for BDataLogPDFs
- `as.BDataSplines(BDataPDFs)`: method for BDataPDFs
- `as.BDataSplines(BDataDensities)`: method for BDataDensities
- `as.BDataSplines(BDataSplines)`: method for BDataSplines, eventually reprojects if basis given
- `as.BDataSplines(BDataSamples)`: method for BDataSamples
- `as.BDataSplines(BDataGeneral)`: method for BDataGeneral
- `as.BDataSplines(`function`)`: method for function interpreted as density with respect to the reference measure.
- `as.BDataSplines(histogram)`: method for function interpreted as density with respect to the reference measure. (TODO: Uses a very crude heuristic to estimate the log density)

- `as.BDataSplines(BayesSpaceBasis)`: the distributions corresponding to the basis elements.

Note

When converting from densities, log-densities or PDFs, the spline coefficients are the L2 projection onto the (non-orthonormal) ZB-spline basis: they solve the normal equations ‘S c = <f, zeta>‘ with the Gram matrix ‘S = getMetricTensor(basis)‘ (see [getMetricTensor]). The inner products ‘<f, zeta>‘ are estimated by Monte-Carlo integration over the reference-measure sample, so the coefficients carry a small sampling error that decreases as the sample size grows.

See Also

[as.BDataDensities], [BayesSpaceBasis], [getMetricTensor]

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataHistograms()`, `as.BDataLogDensities()`, `as.BDataLogPDFs()`, `as.BDataPDFs()`, `as.BDataSamples()`, `idt.BData()`, `toHistogram()`

Examples

```
X = BDataDensities(list(\(x) x, \(x) x^2),basis=B2Unif01Basis5)
as.BDataSplines(X)
as.BDataSplines(as.BDataSplines(X))
as.BDataSplines(as.BDataLogDensities(X))
as.BDataSplines(BDataSamples(replicate(10,runif(100),simplify=FALSE),basis=B2Unif01Basis5))
```

`B2DistHyper` *A distribution of distributions based a distribution family with random hyperparameters.*

Description

A distribution of distributions based a distribution family with random hyperparameters.

Usage

```
B2DistHyper(
  density,
  basis,
  paramSimulator,
  ...,
  simulator = NULL,
  .hyperparameters = list(...)
)
```

Arguments

density	A function that can be called with a vectorized x of data values of the BayesSpace and parameters, to calculate a density.
basis	A BayesSpaceBasis (or an object that can be converted to one) of BayesSpace containing the values of the distribution (of distributions).
paramSimulator	A function to be called without arguments to simulate one set of parameters.
...	parameters to be given to parametersimulator
simulator	A function that can be called with an integer n and the parameters to create n random simulations. This function creates a distribution, which can be used to simulate distributional objects according to the random hyperparameters.
.hyperparameters	the list of parametrs to be given to the the parameterSimulator

Value

A B2DistHyper object encoding the distribution.

Examples

```
D = B2DistHyper(dbeta,B2Unif01Basis5,function() rexp(2),simulator=rbeta)
simulate(D,4,seed=47)
```

B2GaussianDistribution

A Gaussian Process distribution for probabilities

Description

A Gaussian Process distribution for probabilities

Simulate a Gaussian Distribution on B2

Usage

```
B2GaussianDistribution(
 mean = BDataLogDensities(list(function(x) rep(0, length(x))), basis = B2Unif01Basis7),
  var = varKernel(function(x, y) exp(-abs(outer(x, y, "-"))), basis =
    getBayesSpaceBasis(mean)),
  nEval = 513
)

## S3 method for class 'B2GaussianDistribution'
simulate(object, nsim = 1, seed = NULL, ..., nEval = object$nEval)
```

Arguments

`mean`	something covertable to a single BDataLogDensity, providing the Bayes space expectation of the distribution, as well as BayesSpace and Support. Currently only finite continuous supports are possible. You might want to simulate on B2Unif01 and transform for other Transformed spaces.
`var`	a function of two arguments providing the covariance operator
`nEval`	the number of equispaced discretization points used on the interval.
`object`	the B2GaussianDistribution object
`nsim`	the number of simulations to be computed
`seed`	the optional seed, if used it will be reported as a seed attribution of the result (use with care)
`...`	unused arguments for generic use

Value

B2GaussianDistribution: a B2GaussianDistribution object which can be simulated with, simulate: BDataLogDensities-Object with nsim cases with the same BayesSpace and BayesSpaceBasis as the mean.

Methods (by generic)

- `simulate(B2GaussianDistribution)`: Simulaton method

See Also

[

[stats::simulate]

Examples

```
m = BDataPDFs(list(dunif),basis=B2Unif01Basis7)
G1 = B2GaussianDistribution(m,function(x,y) exp(-(4*abs(x-y)^1.7)))
plot(simulate(G1,10,seed=47),col=2:11)

GD = B2GaussianDistribution(mean=BDataPDFs(\(x)x,basis=B2Unif01Basis5),
                            var=\(x,y) exp(-abs(x-y)))
plot(simulate(GD,5))
GD = B2GaussianDistribution(mean=BDataPDFs(\(x) dnorm(x),basis=B2N01Basis5),
                            var=\(x,y) exp(-abs(x-y)))
plot(as.BDataPDFs(simulate(GD,5)))
```

B2N01 *The standard B2(N(0,1)) Bayes space.*

Description

B2N01 is the Bayes space B2(N(0,1)), and B2N01BasisN its B-Spline basis with N elements of order 2

Usage

```
B2N01

B2N01Basis5

B2N01Basis7

B2N01Basis20
```

Format

An object of class `BayesSpace` of length 1.

An object of class `BayesSpaceBasis` of length 5.

An object of class `BayesSpaceBasis` of length 5.

An object of class `BayesSpaceBasis` of length 5.

Functions

- `B2N01Basis5`: a basis for B2N01 with 5 elements
- `B2N01Basis7`: a basis for B2N01 with 7 elements
- `B2N01Basis20`: a basis for B2N01 with 20 elements

See Also

[BayesSpaceBasis]

Other ReferenceMeasures: `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `ReferenceMeasureUniform()`, `StdReferenceMeasureUnif01`, `expectation()`, `getSample()`, `isInDomain()`

Examples

```
getReferenceMeasure(B2N01)
getMetricTensor(B2N01Basis5)
```

basisEval *Evaluates a BayesSpaceBasis*

Description

The basis is represented by log-densities. The function is vectorized, i.e. multiple x values may be provided.

Usage

```
basisEval(basis, x, ...)

## S3 method for class 'BayesSpaceBasis'
basisEval(basis, x, ...)

## S3 method for class 'basisfd'
basisEval(basis, x, ...)

## S3 method for class 'BayesSpaceBasis'
as.function(x, ...)
```

Arguments

basis	a description of a basis, e.g. through a BayesSpaceBasis object
x	the data values on which the basis shall be evaluated
...	further parameters that a generic can use, e.g. describing method and precision of evaluation.

Value

a matrix with the evaluation for one each of the x in a row.

Methods (by class)

- `basisEval(BayesSpaceBasis)`: the standard method for BayesSpaceBasis
- `basisEval(basisfd)`: a method implementation for basisfd

Functions

- `as.function(BayesSpaceBasis)`: the basis can also be converted to a function or called directly

Examples

```
x = seq(0,1,by=0.1)
basisEval(B2Unif01Basis5,x)
```

BayesSpace *Creates the Bayes space for a reference measure*

Description

A Bayes Space is a Hilbert space of distribution and likelihoods having a non zero density with respect to the reference measure (which needs) to be a distribution itself.

Usage

```
BayesSpace(referenceMeasure = ReferenceMeasureUniform(0, 1))
```

Arguments

`referenceMeasure`
: a "ReferenceMeasure" object

Value

An object of class ‘"BayesSpace"‘: a list holding the reference measure.

See Also

[ReferenceMeasure]

Other BayesSpace: `L2Space()`

Examples

```
bs = BayesSpace(ReferenceMeasureUniform(0,10))
bs
X = BDataSamples(replicate(10,runif(100,0,10)),basis=BayesSpaceBasis(bs))
```

BayesSpaceBasis *Create a basis for a Bayes space*

Description

Creates the finite-dimensional basis in which distributional data are represented for computation. Besides the functional spline basis and its [BayesSpace], the returned object stores the transformation matrix ‘A‘ and the Gram matrix ‘S‘ (the inner products of the basis elements).

Usage

```
BayesSpaceBasis(basis, ...)

## S3 method for class 'basisfd'
BayesSpaceBasis(
  basis,
 bayesSpace = BayesSpace(ReferenceMeasureUniform(basis$rangeval[1], basis$rangeval[2])),
  A = computeL22BayesSpacesBasisMatrix(basis),
  S = A %*% fda::inprod(basis, basis) %*% t(A)/diff(basis$rangeval),
  orthonormalize = FALSE,
  ...
)

## S3 method for class 'fd'
BayesSpaceBasis(basis, ...)

## S3 method for class 'BayesSpace'
BayesSpaceBasis(basis, ...)
```

Arguments

basis	the object describing the basis (a function, an ‘fda‘ basis, a reference measure, or a [BayesSpace]).
...	further generic arguments
bayesSpace	the BayesSpace this basis is designed for
A	the transformation matrix mapping the result of the function to the basis values
S	the Gram (metric) matrix of the given basis, ‘S[i,j] = <psi_i, psi_j>‘. For an orthonormal basis ‘S‘ is the identity; for ZB-splines it is not. Used by all inner-product based operations.
orthonormalize	boolean if true creates an orthonormal basis, by a sigualr value decomposition based orthonormalization.

Details

The default basis (constructed from a reference measure or an ‘fda‘ B-spline basis) consists of **zero-integral B-splines (ZB-splines)**: spline functions that integrate to zero, which is required to represent centred-log-ratio (clr) transformed densities. The object stores

‘basis‘ a function evaluating the underlying (B-spline) basis,

‘A‘ the transformation matrix mapping those values to the ZB-spline basis, so the basis functions are ‘zeta = A %*% B‘,

‘S‘ the Gram (metric) matrix ‘S[i,j] = <zeta_i, zeta_j>‘,

‘bayesSpace‘, ‘centered‘ the space and a centring flag.

ZB-splines are **not orthonormal**, so in general ‘S‘ is not the identity; inner products, norms, variances and PCA all use ‘S‘ (see [getMetricTensor], [princomp.BData]). Pass an explicit ‘S‘ only if you have a better value than the internally computed one.

Value

An object of class ‘"BayesSpaceBasis"‘: a list with components ‘basis‘, ‘bayesSpace‘, ‘A‘, ‘S‘ and ‘centered‘.

Methods (by class)

- `BayesSpaceBasis(basisfd)`: Construct from an basisfd object from functional data
- `BayesSpaceBasis(fd)`: Construct from an fd object from functional data
- `BayesSpaceBasis(BayesSpace)`: Calls BayesSpaceBasis on the Reference measure

See Also

[getMetricTensor], [BayesSpace], [as.BDataSplines], [princomp.BData]

[fda::basisfd]

[fda::fd]

Examples

```
basis <- BayesSpaceBasis(B2Unif01, nbasis = 7)
getMetricTensor(basis)       # Gram matrix S (not the identity for ZB-splines)
```

BayesSpaceBasis.BayesSpaceCartesianProduct

Create a BayesSpaceBasis for a Cartesian Product of Bayes Spaces

---

Description

Create a BayesSpaceBasis for a Cartesian Product of Bayes Spaces

Usage

```
## S3 method for class 'BayesSpaceCartesianProduct'
BayesSpaceBasis(basis, ..., useBases = lapply(basis, BayesSpaceBasis, ...))

## S3 method for class 'ReferenceMeasureCartesianProduct'
BayesSpaceBasis(
  basis,
  ...,
  useBases = lapply(bayesSpace, BayesSpaceBasis, ...),
  bayesSpace = BayesSpaceCartesianProduct(lapply(basis, BayesSpace))
)
```

Arguments

basis	A cartesian product of Bayes Spaces (BayesSpaceCartesianProduct)
...	further parameter controling the BayesSpaceBasis generation, used by the BayesSpaces basis generation tools of the factors of the cartesian product.
useBases	a list of marginal BayesSpaceBases corresponding to the BayesSpaces.
bayesSpace	the BayesSpace to be used for this CartesianProduct. It has a good default, but should be given in case you want to create the basis for an already existing ProductSpace.

Value

A BayesSpaceBasisCartesianProduct of the bases in useBases, for the BayesSpace basis.

Functions

- `BayesSpaceBasis(ReferenceMeasureCartesianProduct)`: creates the BayesSpaceCartesian Product for this reference measure and the corresponding product basis.

See Also

BayesSpaceBasis

Other Cartesian: `BayesSpaceBasisCartesianProduct()`, `BayesSpaceCartesianProduct()`, `ReferenceMeasureCartesi` `SupportCartesianProduct()`, `cartesianProduct()`, `getBayesSpace.ReferenceMeasureCartesianProduct()`, `getCartesianFactors()`, `subBasisIndices()`

Examples

```
BayesSpaceBasis(BayesSpaceCartesianProduct(
    A=BayesSpace(StdReferenceMeasureUnif01),
    B=BayesSpace(ReferenceMeasureDiscrete(c("A","B","C"),c(0.2,0.3,0.5))),
    C=BayesSpace(StdReferenceMeasureN01)
    )
)
```

BayesSpaceBasis.function

Creates a BayesSpaceBasis from a function

Description

The function needs a vectorized function mapping the support of the Bayes space to real densities.

Usage

```
## S3 method for class '`function`'
BayesSpaceBasis(
  basis,
  bayesSpace = B2Unif01,
  A = diag(rep(1, ncol(basis(getSample(bayesSpace))))),
  S = A %*% scalarProductOfLogDensities(basis, space = bayesSpace) %*% t(A),
  orthonormalize = FALSE,
  ...,
  centered = FALSE
)
```

Arguments

basis	the function representing the log densities representing the basis.
bayesSpace	the BayesSpace the basis is defined for
A	the matrix mapping the results of f to the log densities.
S	the Gram (metric) matrix of this basis, ‘S[i,j] = <psi_i, psi_j>‘ (see [getMetricTensor]). Stored as the Gram matrix itself, not its inverse.
orthonormalize	boolean: if TRUE the basis is orthogonalized, using a singluar using a singular value decomposition based matrix square root.
...	further unused generic arguments
centered	a boolean, if true the basis is considered centered

Details

A and S only need to be provided if there is a better result availabel than from the internal computation of the Bayes space

Value

A BayesSpaceBasis

See Also

BayesSpaceBasis

Examples

```
BayesSpaceBasis(function(x) outer(x,1:5,function(a,b) cos(a*b*pi)))
BayesSpaceBasis(function(x) outer(x,1:5,function(a,b) cos(a*b*pi)),orthonormalize=TRUE)
```

BayesSpaceBasis.ReferenceMeasureDiscrete

Coded Basis for ReferenceMeasureDiscrete

Description

Coded Basis for ReferenceMeasureDiscrete

Usage

```
## S3 method for class 'ReferenceMeasureDiscrete'
BayesSpaceBasis(
  basis,
  bayesSpace = BayesSpace(basis),
  orthonormalize = FALSE,
  ...
)
```

Arguments

basis	the ReferenceMeasureDiscrete the basis is designed for
bayesSpace	the BayesSpace to be used for the BayesSpaceBasis
orthonormalize	boolean: if TRUE the basis is orthonormalized, using a singluar using a singular value decomposition based matrix square root.
...	further arguments for generic use only.

Value

A BayesSpaceBasis for that discrete reference measure. If a nonstandard A is used it might be nonorthogonal.

Examples

```
X=c("A","B","C")
rmd = ReferenceMeasureDiscrete(X)
basis = BayesSpaceBasis(rmd)
as.function(basis)(X)
weightedStats(rmd,as.function(basis),function(x) rep(1,length(x)))
```

BayesSpaceBasis.ReferenceMeasureTransformed

Generates a BayesSpaceBasis for transformed Bayes Spaces

Description

The basis is a transformation of the Basis that would be computed for the underlying untransformed BayesSpace.

Usage

```
## S3 method for class 'ReferenceMeasureTransformed'
BayesSpaceBasis(
  basis,
  bayesSpace = getBayesSpace(basis),
  orthonormalize = FALSE,
  ...
)
```

Arguments

basis	a ReferenceMeasureTransformed, for which the basis shall be computed.
bayesSpace	the BayesSpace of this ReferenceMeasure
orthonormalize	boolean: if TRUE the basis is orthonoramlized, using a singluar using a singular value decomposition based matrix square root.
...	further parameters for the BayesSpaceBasis method of the underlying untransformed ReferenceMeasure, typically ReferenceMeasureUniform. The breaks are than to be understood as quantiles.

Value

an object of class ‘"BayesSpaceBasis"‘, transforming the basis of the underlying untransformed BayesSpace.

Examples

```
rmt <- ReferenceMeasureTransformed("pnorm", mean=2, sd=2)
basis <- BayesSpaceBasis(rmt)
basis
```

BayesSpaceBasis.ReferenceMeasureUniform

create the ‘BayesSpaceBasis‘ object for ‘ReferenceMeasureUniform‘

Description

create the ‘BayesSpaceBasis‘ object for ‘ReferenceMeasureUniform‘

Usage

```
## S3 method for class 'ReferenceMeasureUniform'
BayesSpaceBasis(
  basis,
  bayesSpace = getBayesSpace(basis),
  orthonormalize = FALSE,
  nbasis = 7,
  order = 4,
  breaks = seq(basis$a, basis$b, length.out = (nbasis - order + 3)),
  ...
)
```

Arguments

basis	object of class ‘ReferenceMeasureUniform‘
bayesSpace	object of class ‘BayesSpace‘, of which the result is a basis.
orthonormalize	boolean if true creates and orthonormal basis
nbasis	number of basis functions for the B-spline basis
order	order of the polynomial of the basis functions
breaks	ascending nonrepetetive vector of knots
...	arguments to be passed to [fda::create.bspline.basis()]

Value

object of class ‘BayesSpaceBasis‘

Examples

```
basis <- BayesSpaceBasis(ReferenceMeasureUniform(0,1))
basis
#basis(seq(0,1,length.out=10))
```

BayesSpaceBasisCartesianProduct

Computes the product basis of BayesSpaceBasis for the product BayesSpace

Description

Computes the product basis of BayesSpaceBasis for the product BayesSpace

Usage

```
BayesSpaceBasisCartesianProduct(
  ...,
  .LIST = .listCatcher(...),
  bayesSpace = BayesSpaceCartesianProduct(.LIST = lapply(.LIST, getBayesSpace))
)
```

Arguments

`...`	the BayesSpacesBases to operate, either directly or as a simple list.
`.LIST`	the list of these bases, use either ... or .LIST
`bayesSpace`	the Product BayesSpace for the basis. The product space is computed automatically, if not given. Try to create this space only once.

Value

A Bayes Space Basis for the product space as defined the DDA Book.

See Also

[BayesSpaceBasis]

Other Cartesian: `BayesSpaceBasis.BayesSpaceCartesianProduct()`, `BayesSpaceCartesianProduct()`, `ReferenceMeasureCartesianProduct()`, `SupportCartesianProduct()`, `cartesianProduct()`, `getBayesSpace.ReferenceMeasureCartesianProduct()`, `getCartesianFactors()`, `subBasisIndices()`

BayesSpaceCartesianProduct

The Cartesian Product of Bayes Spaces

Description

The Cartesian Product of Bayes Spaces

Usage

```
BayesSpaceCartesianProduct(...)

L2SpaceCartesianProduct(...)
```

Arguments

...	the Bayes Spaces to be used in the product either given directly or as a simple list. Names will to used to name the variable they are operating on.

Value

the Product space of the BayesSpaces, a BayesSpace itself.

Functions

- `L2SpaceCartesianProduct()`: Product of L2 spaces

See Also

Other Cartesian: `BayesSpaceBasis.BayesSpaceCartesianProduct()`, `BayesSpaceBasisCartesianProduct()`, `ReferenceMeasureCartesianProduct()`, `SupportCartesianProduct()`, `cartesianProduct()`, `getBayesSpace.ReferenceMeasureCartesianProduct()`, `getCartesianFactors()`, `subBasisIndices()`

BData — *Distributional data containers (BData)*

Description

‘BData‘ is the base class for the data containers in DistributionalDDA. A ‘BData‘ object holds a data set of distributions together with the [BayesSpaceBasis] in which they are represented. Several subclasses store the same information in different forms; pick the one that matches how your data arrive, then convert to splines for computation.

Details

Which representation should I use?

Raw samples [BDataSamples] – one numeric sample per distribution.

Binned data [BDataFrequencyHistograms] (counts) or [BDataDensityHistograms] (densities).

Density functions w.r.t. the Lebesgue measure [BDataPDFs] / [BDataLogPDFs].

Density functions w.r.t. the reference measure [BDataDensities] / [BDataLogDensities].

Spline coefficients [BDataSplines] – the compact representation used for computation (PCA, distances, inner products).

Conversion pipeline The ‘as.BData*‘ converters move between representations. The usual flow is

```
Samples / Histograms  ->  Densities  ->  Splines  ->  PDFs
```

Converting towards densities/splines requires a basis and is an approximation (the data are projected onto the finite-dimensional spline space); extracting samples or re-expressing a representation as itself is lossless. For most analyses, convert to [BDataSplines] with [as.BDataSplines].

See Also

[BayesSpace], [BayesSpaceBasis], [ReferenceMeasure], [as.BDataSplines], [as.BDataDensities]

`BDataDensities` *Constructor of BDataDensities objects representing a Dataset of Densities in a BayesSpace*

Description

The dataset is represented by a list of functions, associated with a Bayes space. The basis is used for functions relying on projections, e.g. when the data is converted to a BDataSpline object.

Usage

```
BDataDensities(f, basis, ...)

BDataPDFs(f, basis, ...)

## S3 method for class 'list'
BDataLogDensities(
  f,
  basis,
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'list'
BDataLogPDFs(
  f,
  basis,
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class '`function`'
BDataDensities(
```

```
  f,
  basis,
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class '`function`'
BDataPDFs(
  f,
  basis,
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'list'
BDataDensities(
  f,
  basis,
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'list'
BDataPDFs(f, basis, ..., close = FALSE, closed = FALSE, centered = FALSE)
```

Arguments

f	a dataset of densities in some form
basis	the BayesSpaceBasis (or an object that can be converted to one) to be used for projections.
...	further generic parameters
close	boolean, if true a closed object will be returned
closed	boolean, if true it is assumed that the input is already closed
centered	boolean, if true it is assumed that the input is already centered

Details

BDataDensities represent densities (aka as Radon-Nikodym derivatives) with repect to the reference measure of the BayesSpace.

BDataPDFs for (Probablity Density function) represents densities with respect to the Lebesgue measure (Haar measure) of the support. I.e. the measure that is invariant under shift or permutation

of the space. BDataPDFs are only reasonable if the reference measure is absolutely continuous, i.e. itself has a finite Lebesgue density.

For the special case of uniform reference measures BDataDensities and BDataPDFs of the same function, represent the same measure.

Value

An object of class ‘c("BDataDensities","BData")‘ representing the densities with respect to the reference measure.

Methods (by class)

- `BDataDensities(`function`)`: Method for a single function
- `BDataDensities(list)`: Method for a list of functions

Functions

- `BDataPDFs()`: Object representing Lebesgue densities
- `BDataLogDensities(list)`: Method for list of functions interpreted as log densities.
- `BDataLogPDFs(list)`: Method for list of functions interpreted as log probability density functions (PDFs).
- `BDataPDFs(`function`)`: Method for a single function
- `BDataPDFs(list)`: Method for a list of functions

See Also

[BayesSpaceBasis]

Other BData: `BDataDensityHistograms()`, `BDataLogDensities()`

Examples

```
BDataDensities(list(function(x) x, function(x) 1-x),B2Unif01Basis5)
```

`BDataDensityHistograms`

Data container for distributions as histograms

Description

Simple S3 constructor of a data container for distributional data where each distribution is represented as a density histogram.

Usage

```
BDataDensityHistograms(matrixDensities, breaks, basis)
```

Arguments

`matrixDensities`
: a matrix (or some object convertible to one with ‘as.matrix()‘) containing densities, observations by row and histogram bins by column

`breaks`
: defining the histogram bins

`basis`
: the B-spline basis to be used

Value

An object of class ‘c("BDataDensityHistograms","BDataHistograms","BData")‘, with the matrix densities as main value and ‘breaks‘ and ‘basis‘ as attributes.

See Also

Other BData: `BDataDensities()`, `BDataLogDensities()`

Examples

```
set.seed(47)
breaks = seq(-3,3,by=1)
inRange = function(x) -3<x & x<3
X = t(replicate(10,hist(Filter(inRange,rnorm(1000)),breaks=breaks,plot=FALSE)$density))
hists = BDataDensityHistograms(X,breaks=breaks,basis=B2N01Basis5)
plot(hists)
```

BDataFrequencyHistograms

Data container for distributions as histograms

Description

Simple S3 constructor of a data container for distributional data where each distribution is represented as a (absolute) frequency histogram.

Usage

```
BDataFrequencyHistograms(matrixCounts, breaks, basis)
```

Arguments

`matrixCounts`
: a matrix (or some object convertible to one with ‘as.matrix()‘) containing absolute frequencies, observations by row and histogram bins by column

`breaks`
: defining the histogram bins

`basis`
: the B-spline basis to be used

Value

An object of class ‘c("BDataFrequencyHistograms","BDataHistograms","BData")‘, with the matrix counts as main value and ‘breaks‘ and ‘basis‘ as attributres.

See Also

Other BDataHistograms: `as.BDataDensityHistograms()`, `as.BDataDensityHistograms.list()`, `as.BDataFrequencyHistograms.list()`, `getBreaks()`, `plot.BDataHistograms()`

Examples

```
set.seed(47)
breaks = seq(-3,3,by=1)
inRange = function(x) -3<x & x<3
X = t(replicate(10,hist(Filter(inRange,rnorm(1000)),breaks=breaks)$counts))
hists = BDataFrequencyHistograms(X,breaks=breaks,basis=B2N01Basis5)
plot(hists)
```

BDataGeneral *BayesSpace data in different formats*

Description

A BDataGeneral object is essentially a list of different representations of BayesSpace elements of the same BayesSpace. They are treated individually as long as possible.

Usage

```
BDataGeneral(
  data,
  basis,
  ...,
  close = FALSE,
  closed = isClosed(data),
  centered = isCentered(data)
)
```

Arguments

`data`	the list of different BayesSpace elements
`basis`	the basis of the common BayesSpace that shall be used, when a basis is needed.
`...`	further arguments to clo
`close`	boolean, if true a closed object will be returned
`closed`	boolean, if true it is assumed that the input is already closed
`centered`	boolean, if true it is assumed that the input is already centered

Value

An object of class ‘c("BDataGeneral","BData")‘: a list bundling the several distributional representations for a common Bayes space.

Examples

```
X = BDataGeneral(list(sin,hist(runif(100)),BDataSamples(list(runif(100)),basis=B2Unif01Basis7)),
             basis=B2Unif01Basis5)
Xs= as.BDataSplines(X)
```

BDataLogDensities — *Constructor of BDataLogDensities objects representing a Dataset of log Densities in a BayesSpace*

Description

The dataset is represented by a list of functions, associated with a Bayes space. The basis is used for functions relying on projections, e.g. when the data is converted to a BDataSpline object

Usage

```
BDataLogDensities(f, basis, ...)

BDataLogPDFs(f, basis, ...)

## S3 method for class 'BDataLogDensities'
BDataDensities(
  f,
  basis = getBayesSpaceBasis(f),
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'BDataLogDensities'
BDataDensities(
  f,
  basis = getBayesSpaceBasis(f),
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'BDataDensities'
BDataLogDensities(
```

```
  f,
  basis = getBayesSpaceBasis(f),
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'BDataLogPDFs'
BDataPDFs(
  f,
  basis = getBayesSpaceBasis(f),
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)

## S3 method for class 'BDataPDFs'
BDataLogPDFs(
  f,
  basis = getBayesSpaceBasis(f),
  ...,
  close = FALSE,
  closed = isClosed(f),
  centered = isCentered(f)
)
```

Arguments

`f`	a dataset of densities in some form
`basis`	the BayesSpaceBasis (or an object that can be converted to one) to be used for projections.
`...`	further generic parameters
`close`	boolean, if true a closed object will be returned
`closed`	boolean, if true it is assumed that the input is already closed
`centered`	boolean, if true the input is assumed to be centered

Value

An object of class ‘c("BDataLogDensities","BData")‘ representing the log-densities with respect to the reference measure.

Methods (by class)

- `BDataLogDensities(BDataDensities)`: conversion method

Methods (by generic)

- `BDataDensities(BDataLogDensities)`: Method for a list of functions
- `BDataDensities(BDataLogDensities)`: conversion method

Functions

- `BDataLogPDFs()`: Object representing Lebesgue densities
- `BDataPDFs(BDataLogPDFs)`: conversion method
- `BDataLogPDFs(BDataPDFs)`: conversion method

See Also

[BayesSpaceBasis]

Other BData: `BDataDensities()`, `BDataDensityHistograms()`

Examples

```
BDataDensities(list(function(x) x, function(x) 1-x),B2Unif01Basis5)
```

BDataSamples *BData container with distributions as samples*

Description

Constructs a BData object where each distribution is represented as a sample. Such a sample can be a vector of values (for univariate cases) or some data structure convertible to a "data.frame" (for multivariate cases).

Usage

```
BDataSamples(datalist, basis)

## S3 method for class 'BDataUnivariateSamples'
summary(object, ...)

## S3 method for class 'BDataSamples'
summary(object, ...)

## S3 method for class 'BDataUnivariateSamples'
x[...]

## S3 method for class 'BDataSamples'
x[i, j]
```

Arguments

`datalist`	list of samples
`basis`	the B-spline basis to be used
`object`	object to be summarized
`...`	further arguments passed to summary.default for the individual summaries of each sample/variable, or to vector subsetting
`x`	object to be subsetted
`i`	index of samples to be selected/filtered out, or logical vector of length equal to ‘length(x)‘
`j`	index of variables to be selected/filtered out, or logical vector of length equal to ‘ncol(x[[1]])‘

Value

In multivariate cases, the outcome is an object of class ‘c("BDataSamples","BData")‘ encoding a list of data.frames of the same number of columns. In univariate cases, the outcome is an object of class ‘c("BDataUnivariateSamples","BDataSamples","BData")‘ encoding a list of vectors of samples of each individual distribution.

Methods (by generic)

- `summary(BDataSamples)`: summary method for multivariate cases
- `[`: subsetting method for multivariate samples

Functions

- `summary(BDataUnivariateSamples)`: summary method for univariate cases
- `[`: subsetting method for univariate samples

Examples

```
x = lapply(rep(1000,4), runif) |> BDataSamples(basis=B2Unif01Basis5)
summary(x)
```

BDataSplines — *create a ‘BDataSplines‘ object: a set of distributions represented as splines; possible representation of individual distribution through a vector of coefficients.*

Description

create a ‘BDataSplines‘ object: a set of distributions represented as splines; possible representation of individual distribution through a vector of coefficients.

Usage

```
BDataSplines(matrixCoefs, basis)

## S3 method for class 'BDataSplines'
x[i, j, ...]
```

Arguments

matrixCoefs	a matrix of spline coefficients (nrow: # of distributions; ncol: # of basis functions)
basis	a spline basis
x	a BDataSplines object to be subsetted
i	index of samples to be selected/filtered out, or logical vector of length equal to ‘length(x)‘
j	index of variables to be selected/filtered out, or logical vector of length equal to ‘ncol(x[[1]])‘
...	further generic arguments

Value

object of class ‘BDataSplines‘ describing a set of distributions represented as splines

Methods (by generic)

- [: subsetting method, keeps structure if only i is given.

See Also

BData

Other BDataSplines: plot.BDataSplines()

Examples

```
coefs=matrix(1:15,nrow=3)
bssplines=BDataSplines(coefs,B2Unif01Basis5)
bssplines
```

---

biplot.princomp.BData *Biplot for PCA of Distributional Data*

Description

Creates a biplot that simultaneously displays the principal component scores (observations) and loadings (variables/coefficients) in the same plot. This visualization helps understand relationships between observations and the underlying coefficient structure.

Usage

```
## S3 method for class 'princomp.BData'
biplot(
  x,
  choices = 1:2,
  scale = 0,
  pc.biplot = FALSE,
  main = NULL,
  col = c("black", "red"),
  cex = c(0.8, 0.8),
  xlabs = NULL,
  ylabs = NULL,
  expand = 1,
  arrow.len = 0.1,
  ...
)
```

Arguments

`x`	An object of class ‘princomp.BData‘ as returned by `princomp.BData`.
`choices`	Integer vector of length 2 specifying which principal components to display. Default is ‘c(1, 2)‘ for the first two components.
`scale`	Numeric scaling factor controlling the relative scaling of scores and loadings: **‘0‘** (Default) Distance biplot: Loadings unscaled, scores scaled. Distances between observation labels approximate Euclidean distances in the full space. Best for seeing relationships between observations. **‘1‘** Correlation biplot: Loadings scaled by standard deviations. Inner products between loading arrows approximate covariances. Best for seeing relationships between variables.
`pc.biplot`	Logical. If ‘TRUE‘, use Gabriel’s (1971) biplot scaling, which sets ‘scale=1‘. Default is ‘FALSE‘.
`main`	Character string for plot title. Default is ‘"Biplot"‘.
`col`	Vector of two colors: first for observation labels, second for loading arrows. Default is ‘c("black", "red")‘.
`cex`	Vector of two character expansion factors: first for observation labels, second for loading labels. Default is ‘c(0.8, 0.8)‘.
`xlabs`	Character vector of labels for observations. Default uses observation numbers (1, 2, 3, ...).
`ylabs`	Character vector of labels for loading arrows (coefficients). Default uses ‘"C1"‘, ‘"C2"‘, etc.
`expand`	Numeric expansion factor for loading arrows. Increase this if the arrows are too short to see clearly. Default is 1.
`arrow.len`	Length of arrow heads. Default is 0.1.
`...`	Additional graphical parameters passed to the underlying plot.

Details

Biplot Interpretation

In a biplot:

- **Observations** (numbers/text) represent individual distributions. Observations close together have similar distribution shapes.
- **Loading arrows** (red arrows labeled C1, C2, etc.) represent spline basis coefficients. The direction and length indicate how each coefficient contributes to the principal components.
- **Projecting** an observation onto a loading arrow approximates that observation's value on that coefficient (relative to the mean).
- **Arrow angles**: Arrows pointing in similar directions indicate positively correlated coefficients; opposite directions indicate negative correlation; perpendicular arrows indicate uncorrelated coefficients.

Note on Coefficient Interpretation

Unlike PCA on raw data where variables have direct meaning, the "variables" here are spline basis coefficients. These are abstract and their individual interpretation may not be intuitive. For understanding how distributions vary, the `plot.princomp.BData` function with 'type="eigenfunctions"' provides a more interpretable visualization showing how distribution shapes change along each PC direction.

Why Biplot Works for ZB-Splines

The biplot is meaningful for the default ZB-spline basis because ZB-splines have **local support**: each basis function is non-zero only in a specific region of the domain. This means coefficient i primarily reflects the behavior of the distribution near knot region i. When a loading arrow for coefficient Ci points strongly in a PC direction, it indicates that variation in that region of the domain is associated with that principal component.

Important: If orthonormal splines were used instead, the biplot would lose this interpretability because orthonormal basis functions have global support (they are non-zero across the entire domain), so each coefficient would mix information from everywhere rather than representing a local region.

Value

Invisibly returns a list containing the scaled 'scores' and 'loadings' matrices as plotted.

Author(s)

Matthias Templ

References

Gabriel, K. R. (1971). The biplot graphic display of matrices with application to principal component analysis. *Biometrika*, 58(3), 453-467.

See Also

princomp.BData for performing the PCA.

plot.princomp.BData for other visualization options, especially ‘type="loadings"‘ for functional interpretation.

biplot for the generic biplot function.

Other PCA: plot.princomp.BData(), plot.princomp.BData.grouped(), plot.princomp.BData.multivariate(), predict.princomp.BData(), princomp.BData(), screeplot.princomp.BData(), summary.princomp.BData()

Examples

```
## Create example PCA result
meanDist <- BDataPDFs(list(dunif), basis = B2Unif01Basis7)
gaussProc <- B2GaussianDistribution(meanDist, function(x,y) exp(-4*abs(x-y)^1.7))
pcaResult <- princomp(as.BDataSplines(simulate(gaussProc, 25, seed = 42)))

## Basic biplot
biplot(pcaResult)

## Biplot of PC2 vs PC3
biplot(pcaResult, choices = c(2, 3))

## Correlation biplot (scale=1)
biplot(pcaResult, scale = 1)

## Customized biplot
biplot(pcaResult,
       col = c("blue", "darkgreen"),
       cex = c(1, 0.7),
       expand = 1.5,
       main = "PCA Biplot of Distributional Data")
```

---

boxplot.BData *Functional Boxplot for BData objects*

Description

Currently this function used the [fda::fbplot] function on the evaluated functions in compbination with [depthRank] as depth measure.

Usage

```
## S3 method for class 'BData'
boxplot(
  x,
  ...,
  depth,
```

```
  depthType = c("depthquantile", "spatialdepth", "meandistance", "median"),
  support = getSupport(x),
  SD = supportDiscretization(support, nEval),
  nEval = 513,
  xlim = getSupport1DPlottingLimits(support)
)

## S3 method for class 'BDataSplines'
boxplot(x, ...)

## S3 method for class 'BDataSamples'
boxplot(x, ...)
```

Arguments

`x`	a BData object
`...`	further parameters for [graphics::boxplot] or [fda::fbplot]
`depth`	the depth used to identify the middle region, increasing inwards towards the median.
`depthType`	if depth is missing, an algorithm to compute it. The default is "depthquantile", which uses the spatial depth, but ensures a 50 used the spatialdepth directly, the others use the 1-[centralityQuantiles] with the given method.
`support`	the support on which the functions should be plotted
`SD`	support discretization of the support (see [supportDiscretization]
`nEval`	the number of datapoints
`xlim`	the plotting limits in x direction.

Value

‘NULL‘ invisibly, called for the side effect of producing a functional boxplot.

Functions

- `boxplot(BDataSplines)`: method for BDataSplines uses classical boxplot for the coefficients.
- `boxplot(BDataSamples)`: method for BDataSamples uses classical boxplot for the the datasets.

See Also

fda::fbplot

Examples

```
X = BDataSplines(matrix(rnorm(10*5),ncol=5),basis=B2Unif01Basis5)
boxplot(X)
boxplot(clr(X),depthType="meandistance")
boxplot(clr(X))
```

`cartesianProduct` *Compute the cartesian product*

Description

Compute the cartesian product

Usage

```
cartesianProduct(...)

## S3 method for class 'Support'
cartesianProduct(...)

## S3 method for class 'ReferenceMeasure'
cartesianProduct(...)

## S3 method for class 'BayesSpace'
cartesianProduct(...)

## S3 method for class 'BayesSpaceBasis'
cartesianProduct(...)
```

Arguments

`...`	objects of the same kind (space, measure, support, basis, ...) to be used for a cartesian product, and named parameters of the method functions. The first will determine the generic behaviour. if exactly one argument is given and is a pure list it is interpreted as the list of objects given. The individual spaces can be named.

Value

the Cartesian product of the objects given. Check [SupportCartesianProduct()] for details about this structure.

The cartesian product of a space (like a vector space or support) is the space of tuples, where each tuple element comes from the space of the same position or name. The cartesian product of measures is the product measure, which gives a cartesian product of sets the product of the measures. The cartesian product of functions(like basis) is a function on the cartesian product of supports and gives the product of the function values for each of the components.

Methods (by class)

- `cartesianProduct(Support)`: method for Supports
- `cartesianProduct(ReferenceMeasure)`: method for ReferenceMeasure
- `cartesianProduct(BayesSpace)`: method for BayesSpace
- `cartesianProduct(BayesSpaceBasis)`: method for BayesSpaceBasis

See Also

Other Cartesian: BayesSpaceBasis.BayesSpaceCartesianProduct(), BayesSpaceBasisCartesianProduct(), BayesSpaceCartesianProduct(), ReferenceMeasureCartesianProduct(), SupportCartesianProduct(), getBayesSpace.ReferenceMeasureCartesianProduct(), getCartesianFactors(), subBasisIndices()

---

caseE *Expected values of functions of distributions*

Description

Ideally this should always be called on the original samples.

Usage

```
caseE(x, X, ...)

## S3 method for class 'BDataSamples'
caseE(x, X, ...)

## S3 method for class 'BData'
caseE(x, X, ...)
```

Arguments

x	the dataset of distributions,
X	the real random variable, i.e. a real function define on whats random in the distributions
...	further arguments to called functions (e.g. quantile)

Value

a vector, matrix or with the Expected values for each element of x. if the expected value is multivariate the last dimension iterates over the cases in x.

Methods (by class)

- `caseE(BDataSamples)`: method for BDataSamples simply uses means
- `caseE(BData)`: method for distributions

See Also

Other caseStatistics: caseLogRatios(), caseMeans(), caseQuantiles(), caseSds(), caseVars(), centralityQuantile(), distanceToCenter(), median.BData(), spatialDepth()

Examples

```
data(DTD)
X = BDataSamples(DTD$data %.% shape,basis=B2Unif01Basis5)
caseE(X,X=function(x) x,probs=c(0.25,0.75))
caseE(X,X=function(x) x^2,probs=0.5)
```

---

caseLogRatios *Log-density ratios for each distribution (case)*

Description

Density log ratios independent of the reference measure and any other values.

Usage

```
caseLogRatios(x, numerator, denominator, ...)
```

Arguments

x	a BData dataset of distributions
numerator	numerator, i.e. the same number of possible values of the random varianble
denominator	denominator values, i.e. possible values of the random variable
...	further unused arguments for future generic use

Value

a dataset of the same number of cases as x of the corresponding log rations

See Also

Other caseStatistics: caseE(), caseMeans(), caseQuantiles(), caseSds(), caseVars(), centralityQuantile(), distanceToCenter(), median.BData(), spatialDepth()

Examples

```
data(DTD)
X  = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
caseLogRatios(X,0.8,0.9)
```

caseMeans — *Expected values (means) of real-valued distributions*

Description

Ideally this should always be called on the original samples.

Usage

```
caseMeans(x, ...)
```

Arguments

x	the dataset of distributions,
...	further arguments to called functions (e.g. caseE)

Value

a vector or a matrix (with different cases in columns) of the means of the distributions in the dataset. For multivariate (vector-valued) means, the last dimension iterates over the elements of x.

See Also

Other caseStatistics: caseE(), caseLogRatios(), caseQuantiles(), caseSds(), caseVars(), centralityQuantile(), distanceToCenter(), median.BData(), spatialDepth()

Examples

```
data(DTD)
X = BDataSamples(DTD$data %.% shape,basis=B2Unif01Basis5)
caseMeans(X)
X = BDataSamples(DTD$data,basis=B2Unif01Basis5)
caseMeans(X)
```

caseQuantiles — *Quantiles of distributions in a data set of distributions*

Description

Ideally this should always be called on the original samples.

Usage

```
caseQuantiles(X, ...)

## S3 method for class 'BDataSamples'
caseQuantiles(X, ...)

## S3 method for class 'BData'
caseQuantiles(X, ...)
```

Arguments

X	the dataset, typically a BData object
...	further arguments to called functions (e.g. quantile)

Value

a vector or a matrix (with different probabilities in colums) of quantiles of the distributions in the dataset.

Methods (by class)

- `caseQuantiles(BDataSamples)`: method for BDataSamples, simply calles [stats::quantile]
- `caseQuantiles(BData)`: method for BData not yet implemented

See Also

Other caseStatistics: `caseE()`, `caseLogRatios()`, `caseMeans()`, `caseSds()`, `caseVars()`, `centralityQuantile()`, `distanceToCenter()`, `median.BData()`, `spatialDepth()`

Examples

```
data(DTD)
X = BDataSamples(DTD$data %.% shape,basis=B2Unif01Basis5)
caseQuantiles(X,probs=c(0.25,0.75))
caseQuantiles(X,probs=0.5)
```

---

caseSds *Standard deviations of real-valued distributions*

Description

Ideally this should always be called on the original samples.

Usage

```
caseSds(x, ...)
```

Arguments

x	the dataset of distributions,
...	further arguments to called functions (e.g. caseE)

Value

a vector the standard deviations of the distributions in the dataset.

See Also

Other caseStatistics: caseE(), caseLogRatios(), caseMeans(), caseQuantiles(), caseVars(), centralityQuantile(), distanceToCenter(), median.BData(), spatialDepth()

Examples

```
data(DTD)
X = BDataSamples(DTD$data %.% shape,basis=B2Unif01Basis5)
caseSds(X)
```

---

caseVars *Variances of real-valued distributions*

Description

Ideally this should always be called on the original samples.

Usage

```
caseVars(x, ..., X = function(x) x)

## S3 method for class 'BDataSamples'
caseVars(x, ..., X = function(x) x)

## S3 method for class 'BData'
caseVars(x, ..., X = function(x) x)
```

Arguments

x	the dataset of distributions,
...	further arguments to called functions (e.g. caseE)
X	a function representing the real or vectorial random variable on the distributions in x

Value

A vector or tensor of variances. In case of multiple dimensional means, the last dimension iterates over the elements of x.

Methods (by class)

- `caseVars(BDataSamples)`: method for samples
- `caseVars(BData)`: method for distributions

See Also

Other caseStatistics: `caseE()`, `caseLogRatios()`, `caseMeans()`, `caseQuantiles()`, `caseSds()`, `centralityQuantile()`, `distanceToCenter()`, `median.BData()`, `spatialDepth()`

Examples

```
data(DTD)
X = BDataSamples(DTD$data %.% shape,basis=B2Unif01Basis5)
caseVars(X)
X = BDataSamples(DTD$data,basis=B2Unif01Basis5)
caseVars(X)
```

---

cdf2density *numerically compute a density from a cdf*

Description

The function tries to keep the (open) support, by using a one sided derivative at limits of the support, while using a two sided in the middle of the support.

Usage

```
cdf2density(cdf, ..., .args = list(...), eps = 1e-07, cdf2 = NULL)
```

Arguments

`cdf`	the cummulative distribution function to be derived
`...`	futher named parameters for cdf
`.args`	the further named parameters given as a list
`eps`	the distance to be used in the numerical derivative
`cdf2`	a function representing the cdf of the reference measure. NULL uses a simplification for Lebesque reference.

Value

a vectorized function giving the density obtained by numerically differentiating ‘cdf‘ (a one-sided difference at the boundaries of the support, a two-sided difference in the interior). If ‘cdf2‘ is supplied the result is the density relative to that reference measure instead of the Lebesgue density.

Examples

```
dUnif = cdf2density(punif,min=-0.2,0.4)
x=seq(-1,2,by=0.01)
plot(x,dUnif(x))
```

`cdf2Support` *Try to guess the support from cdf*

Description

This function essentially tries to infere the support from a cdf.

Usage

```
cdf2Support(cdf, alpha = 0.01, eps = 1e-07, digits = 5)
```

Arguments

`cdf`	the cummulative density function from which the support shall be guessed.
`alpha`	the support cutoff probability for effective supports if the support is infinite
`eps`	the computational precision for computing the boundaries
`digits`	the number of digits the boundaries and jumps shall be rounded to.

Value

a ‘Support‘ object describing the support inferred from the ‘cdf‘. Its ‘a‘ and ‘b‘ elements give the (possibly infinite) lower and upper support boundaries, its ‘lim‘ element the effective lower and upper boundaries (a numeric vector of length two, cut at probability ‘alpha/2‘ and ‘1 - alpha/2‘ when the support is unbounded), and ‘X‘ any detected points of positive probability mass.

See Also

Other Support: `Support()`, `pdf2Support()`, `plotFuncs()`, `sample2Support()`

Examples

```
cdf2Support(pnorm)
cdf2Support(punif)
cdf2Support(
  function(x){
  ifelse(x< -pi,0,((pmax(x,0)+pi)/pi)*0.5) +
      ifelse(x<0,0,0.25+(pmax(x,pi)/pi)*0.25)
  }
)
```

cen *Centers the representation*

Description

This centers an representation in a way that a direct converstion to a log Density has integral 0.

Usage

```
cen(X, ..., center = TRUE)

## S3 method for class '`function`'
cen(X, measure = getReferenceMeasure(X), ..., center)

## S3 method for class 'BDataDensities'
cen(X, ..., center)

## S3 method for class 'BDataPDFs'
cen(X, ..., center)

## S3 method for class 'BDataLogDensities'
cen(X, ..., center)

## S3 method for class 'BDataLogPDFs'
cen(X, ..., center)

## S3 method for class 'BayesSpaceBasis'
cen(X, ..., center)
```

Arguments

`X`	the object to be centered
`...`	further arguments for generic use
`center`	a boolean, if FALSE the object is returned directly, if true uncentered objects will be centered.
`measure`	the reference measure for which the closing constant shall be computed. This is especially required for functions

Details

Centring subtracts the integral mean (with respect to the reference measure) so that the result integrates to zero; equivalently it makes the clr representative of the distribution centred.

Value

and object of the same type as the incoming, but closed.

Methods (by class)

- `cen(`function`)`: method for functions
- `cen(BDataDensities)`: method for BDataDensities
- `cen(BDataPDFs)`: method for BDataPDFs
- `cen(BDataLogDensities)`: method for BDataLogDensities
- `cen(BDataLogPDFs)`: method for BDataLogPDFs
- `cen(BayesSpaceBasis)`: method for BayesSpaceBasis returns the centered logDensities

See Also

[clo]

Examples

```
cen(function(x) x^2,measure=ReferenceMeasureUniform(0,1))
```

---

centeringConstant *Finds the centering constant(s)*

Description

A centering constant is a real value, by which needs to be substracted from a function (of its log in case of densities) get an integral of 0.

Usage

```
centeringConstant(f, ...)

## S3 method for class '`function`'
centeringConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BayesSpaceBasis'
centeringConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataDensities'
centeringConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataLogDensities'
centeringConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataLogPDFs'
centeringConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataPDFs'
```

```
centeringConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataSplines'
centeringConstant(f, measure = getReferenceMeasure(f), ...)
```

Arguments

`f`	a representation of a measure
`...`	further arguments for generic use
`measure`	the ReferenceMeasure, needed especially for functions

Value

a (vector) of numeric centering constants. Might contain Inf in case of infinite measures and 0 in case of functions that are constant zeros

Methods (by class)

- `centeringConstant(` `function` `)`: method for function
- `centeringConstant(BayesSpaceBasis)`: method for BayesSpaceBasis
- `centeringConstant(BDataDensities)`: method for BDataDensities
- `centeringConstant(BDataLogDensities)`: method for BDataLogDensities
- `centeringConstant(BDataLogPDFs)`: method for BDataLogPDFs
- `centeringConstant(BDataPDFs)`: method for BDataPDFs
- `centeringConstant(BDataSplines)`: method for BDataSplines

Examples

```
# centering constant of f(x) = x^2 on Uniform(0,1)
centeringConstant(function(x) x^2, measure = ReferenceMeasureUniform(0,1))
```

`centralityQuantile` *centralityQuantile*

Description

Computes quantile of distanceToCenter of data in a dataset, 0 is the median, 1 is the farthest point

Usage

```
centralityQuantile(x, ...)

## Default S3 method:
centralityQuantile(
  x,
  ...,
  d = distanceToCenter(x, ..., distType = distType),
  distType = c("meandistance", "median", "spatialdepth"),
  ties.method = "first"
)

centralityRank(x, ...)

## Default S3 method:
centralityRank(
  x,
  ...,
  d = distanceToCenter(x, ..., distType = distType),
  distType = c("meandistance", "median", "spatialdepth"),
  ties.method = "first"
)
```

Arguments

`x`	the dataset
`...`	further parameters to [distanceToCenter]
`d`	the distances to median
`distType`	"meandistance" for mean distance to datapoints, "median" for distance to median, "spatialdepth" for the spatial depth.
`ties.method`	see [rank]: the method for breaking ties in the ranks

Value

A numeric vector in [0,1], 0 for the most central point and 1 for the most peripheral.

Methods (by class)

- `centralityQuantile(default)`: default method

Functions

- `centralityRank()`: returns the rank rather than the quantile: an integer vector of ranks, 1 for the most central point up to the number of observations for the most peripheral.
- `centralityRank(default)`: default method returning ranks

See Also

Other caseStatistics: caseE(), caseLogRatios(), caseMeans(), caseQuantiles(), caseSds(), caseVars(), distanceToCenter(), median.BData(), spatialDepth()

Examples

```
x = rnorm(100)
plot(x,centralityQuantile(x))
plot(x,centralityRank(x))
```

---

clo — *Closes to a given total measure*

Description

This is a generic version of clo overwriting the one in compositions

Usage

```
clo(X, ..., close = TRUE)

## Default S3 method:
clo(X, ..., total = 1, close)

## S3 method for class '`function`'
clo(X, measure = getReferenceMeasure(X), ..., total = 1, close)

## S3 method for class 'BDataDensities'
clo(X, ..., total = 1, close)

## S3 method for class 'BDataPDFs'
clo(X, ..., total = 1, close)

## S3 method for class 'BDataLogDensities'
clo(X, ..., total = 1, close)

## S3 method for class 'BDataLogPDFs'
clo(X, ..., total = 1, close)

## S3 method for class 'BayesSpaceBasis'
clo(X, ..., total = 1)
```

Arguments

X	the object to be closed
...	further arguments for generic use

`close`	a boolean, if FALSE the object is returned directly, if true unclosed objects will be closed.
`total`	the integral that is desired
`measure`	the reference measure for which the closing constant shall be computed. This is especially required for functions

Details

Closure rescales a (positive) function so that it integrates to ‘total‘ (one by default) with respect to the reference measure, i.e. it projects the function to a proper density and thereby picks the representative of its Bayes-space class.

Value

and object of the same type as the incoming, but closed.

Methods (by class)

- `clo(default)`: the default is [compositions::clo]
- `clo(`function`)`: method for functions
- `clo(BDataDensities)`: method for BDataDensities
- `clo(BDataPDFs)`: method for BDataPDFs
- `clo(BDataLogDensities)`: method for BDataLogDensities
- `clo(BDataLogPDFs)`: method for BDataLogPDFs
- `clo(BayesSpaceBasis)`: method for BayesSpaceBasis returns the closed logDensities

See Also

[compositions::clo] for parameters of the default, [cen]

Examples

```
clo(function(x) x^2,measure=ReferenceMeasureUniform(0,1))
```

`clo.BDataHistograms` *Closure methods for BData objects*

Description

Closure functions to enforce a distributional representation to integrate to one, if needed.

Usage

```
## S3 method for class 'BDataHistograms'
clo(X, ...)
```

Arguments

X	distribution representation, typically an object of a derived class of "BData"
...	further arguments, typically ignores

Value

A fitting object with elements now integrating to one, depending on the specific method.

Examples

```
# close frequency histograms to density histograms integrating to one
counts = rbind(c(10,20,30,40), c(25,25,25,25))
fh = BDataFrequencyHistograms(counts, breaks = seq(0,1,by=0.25),
                              basis = B2Unif01Basis5)
clo(fh)
```

---

closingConstant *Finds the closing constant(s)*

Description

A closing constant is a real value, by which the function (or its Exp in case of log functions) needs to be divided to get the representative with measure 1.

Usage

```
closingConstant(f, ...)

## S3 method for class '`function`'
closingConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BayesSpaceBasis'
closingConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataDensities'
closingConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataLogDensities'
closingConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataLogPDFs'
closingConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataPDFs'
closingConstant(f, measure = getReferenceMeasure(f), ...)

## S3 method for class 'BDataSplines'
closingConstant(f, measure = getReferenceMeasure(f), ...)
```

Arguments

`f`	a representation of a measure
`...`	further arguments for generic use
`measure`	the ReferenceMeasure, needed especially for functions

Value

a (vector) of numeric closing constants. Might contain Inf in case of infinite measures and 0 in case of functions that are constant zeros

Methods (by class)

- `closingConstant(`function`)`: method for function
- `closingConstant(BayesSpaceBasis)`: method for BayesSpaceBasis
- `closingConstant(BDataDensities)`: method for BDataDensities
- `closingConstant(BDataLogDensities)`: method for BDataLogDensities
- `closingConstant(BDataLogPDFs)`: method for BDataLogPDFs
- `closingConstant(BDataPDFs)`: method for BDataPDFs
- `closingConstant(BDataSplines)`: method for BDataSplines

Examples

```
# closing constant of f(x) = x^2 on Uniform(0,1)
closingConstant(function(x) x^2, measure = ReferenceMeasureUniform(0,1))
```

---

clr *compute the clr transform*

Description

This is a generic version of clr overwriting the one in compositions

Usage

```
clr(x, ...)

## Default S3 method:
clr(x, ...)

## S3 method for class 'BData'
clr(x, ...)
```

Arguments

x	the object to be clr transformed
...	further arguments for generic use

Details

The centred log-ratio map sends a distribution to its log-density centred to integrate to zero, i.e. to its coordinates in the L2 space underlying the Bayes space. It is an isometry from the Bayes space onto the centred subspace of that L2 space, so Bayes-space distances and inner products become the ordinary L2 ones.

Value

the centred log-ratio (clr) representation of ‘x‘. For ‘BData‘ objects this is the centred log-density representation (an object of class ‘BDataLogDensities‘ whose log-density integrates to zero); the default method returns the clr transform computed by [compositions::clr].

Methods (by class)

- `clr(default)`: the default is [compositions::clr]
- `clr(BData)`: returns a centered representation as BDataLogDensities

Examples

```
X1 = BDataPDFs(basis=B2Unif01Basis5,
   list(
    \(x)(dbeta(x,1,2)),
    \(x)(dbeta(x,1,1)),
    \(x)(dbeta(x,1.5,0.7))
   ) )
plot(X1,main="Beta Densities on Unif(0,1)")
plot(clr(X1),main="CLRs of Beta Densities on Unif(0,1)")
```

---

computeL22BayesSpacesBasisMatrix

Computes the matrix mapping an L2-space basis to a Bayes-Space-Basis.

Description

This computes a (n-1)x1 matrix mapping the basis results of the L2 basis to L^2_0 basis for a subspace of the Log-Space of a BayesSpace.

Usage

```
computeL22BayesSpacesBasisMatrix(x, ...)
```

Arguments

x	the L2-basis
...	further generic arguments potentially parametrizing the method of computation.

Value

the transformation matrix ‘A‘ mapping the L2 (B-spline) basis to the ZB-spline basis of the Bayes space. For a B-spline basis with ‘nbasis‘ functions this is an ‘(nbasis - 1) x nbasis‘ matrix, i.e. one row per ZB-spline basis function.

---

computeL22BayesSpacesBasisMatrix.basisfd

create the transform matrix A from a B-spline basis (fda::create.bspline.basis())

Description

create the transform matrix A from a B-spline basis (fda::create.bspline.basis())

Usage

```
## S3 method for class 'basisfd'
computeL22BayesSpacesBasisMatrix(x, ...)
```

Arguments

x	a ’basisfd’ object
...	further unused generic arguments

Value

matrix

Examples

```
computeL22BayesSpacesBasisMatrix(
         fda::create.bspline.basis(rangeval = c(0,1),
                                   nbasis=9,
                                   norder=4,
                                   breaks=seq(0,1,length.out=7)
                                   )
                                 )
# a cubic B-spline basis with 8 functions yields a 7 x 8 matrix
A = computeL22BayesSpacesBasisMatrix(
        fda::create.bspline.basis(c(0,1), nbasis=8, norder=4)
    )
dim(A)
```

computeLimit — *computes the limit of a function*

Description

computes the limit of a function

Usage

```
computeLimit(f, x, d = 1)
```

Arguments

f	the function whos limit shall be computed
x	the location, where the limit shall be computed
d	the direction of the limit

Value

the limit it will still return

Examples

```
computeLimit(\(x)(2*(x-pi)/(x-pi)),x=pi)
np = computeLimit(\(x)(pi*(x-pi)/(x-pi)),x=pi)
np-pi
```

CovarianceOperator — *Computes a covariance with respect to a measure. The function can be considered internal implementation details of the package, but needs to be exported due to its generic nature.*

Description

The implementation depends on the measure

Usage

```
CovarianceOperator(measure, X, Y, ...)

## S3 method for class 'ReferenceMeasure'
CovarianceOperator(
  measure,
  X,
  Y,
  ...,
  method = c("default", "sample", "MonteCarlo", "integration"),
  tol = 0.01,
  close = FALSE
)

## S3 method for class 'ReferenceMeasureDiscrete'
CovarianceOperator(measure, X, Y, ...)
```

Arguments

`measure`	typically a ReferenceMeasure
`X`	a random variable (or a list of such) on the domain of the measure represented as an R function. It returns multiple values as columns of a matrix.
`Y`	a random variable (or a list of) on the domain of the measure represented as an R function.
`...`	possibly further implementation dependent arguments for the computational method
`method`	a method to be selected. Typical methods are "sample", "MonteCarlo" and "integrate", but only sample is availabel for all reference measures. "sample" uses the prestored sample, "MonteCarlo" uses the simulated function of the reference measure, and "integrate" uses numerical integration. Typically the prestored integration sample performs well with derivatives measures derived ReferenceMeasureUniform, ReferenceMeasusureTransformed, where it proves a dense grid in the area of measure, and ReferenceMeasure discrete where it uses precise computation using a weighted sample. For Cartesian products involving more than one continues measure due to the curse of dimensionality typically only MonteCarlo is expected to work reasonably well.
`tol`	the absolute or relative tolerance (whichever is larger). accepted for methods with tolerance control.
`close`	if false the underlying measure is not assumed to be closed, and the function makes it possible to computed unclosed covariance integrals for unclosed reference measures. This is the default.

Value

the scalar products (covariances) of ‘X‘ and ‘Y‘ with respect to the reference measure. For the default ReferenceMeasure method these are obtained from [stats::cov()] applied to ‘X‘ and ‘Y‘ evaluated on the measure’s sample. The result is a numeric scalar, vector, or matrix depending on

the number of columns returned by ‘X‘ and ‘Y‘ (one entry per pair of output components); when ‘X‘ or ‘Y‘ is a list, the results are combined columnwise.

Methods (by class)

- `CovarianceOperator(ReferenceMeasure)`: method for ReferenceMeasures using a Monte-Carlo integration based on the sample provided by getSample(sample)
- `CovarianceOperator(ReferenceMeasureDiscrete)`: method for ReferenceMeasuresDiscrete explicit computation

Examples

```
CovarianceOperator(StdReferenceMeasureUnif01,
                   function(x) cbind(x,-x),
                   function(x) cbind(x,-x,2*x+1)
)
```

DiscretizedFunction *Creates a function based on a regular discretization*

Description

A simple fast tool for defining a discretized function defined continuously on [a,b] and discrete values in X, with equidistant data values y in between. The resulting function is vectorized and interpolates linearily with errors or NA outside the region. It handles infinite supports, but not infinite results on the continuous region.

Usage

```
DiscretizedFunction(y, ...)

## S3 method for class 'numeric'
DiscretizedFunction(y, ..., support = Support(...), transferCdf = NULL)

## S3 method for class '`function`'
DiscretizedFunction(
  y,
  a = 0,
  b = a,
  n = 501,
  X = c(),
  transferCdf = NULL,
  transferQf = cdf2qf(transferCdf, ...),
  ...
)
```

```
## S3 method for class 'Discretizer'
DiscretizedFunction(y, Y, ...)

## S3 method for class 'ReferenceMeasureTransformed'
DiscretizedFunction(y, f, n = 513, support = getSupport(y), ...)

## S3 method for class 'ReferenceMeasureUniform'
DiscretizedFunction(y, f, n = 513, support = getSupport(y), ...)
```

Arguments

`y`	the function to be approximated, or evaluation values, first those for X and than those equidistantly spaced xmin and xmax (in the image of transferCdf), or ReferenceMeasure on which to define the function (in this case the function is reported is given in f)
`...`	further arguments for Support (a,b,X)
`support`	the [Support] on which the function shall be defined.
`transferCdf`	a transfer function mapping the interval a,b to [0,1] (or another interval), in which the nots are place equidistantly and are interpolated linearily.
`a`	the lower limit of the domain
`b`	the upper limit of the domain
`n`	the number of discretization points on [a,b]
`X`	the discrete elements of the support (i.e. positions, where the function may jump.
`transferQf`	(qunantile function), must be the inverse of transferCdf and is used to determine the evaluation points for the function.
`Y`	replaces y, if y is a Discretizer
`f`	the function to be approximated (only if y is a ReferenceMeasure)

Value

a function interpolating linearly between the given evaluation points and working in a vectorized form.

Methods (by class)

- `DiscretizedFunction(numeric)`: method for precomputed values
- `DiscretizedFunction(`function`)`: method for functions that need to be computed first.
- `DiscretizedFunction(Discretizer)`: method for Discretizer
- `DiscretizedFunction(ReferenceMeasureTransformed)`: method for functions on Transformed Reference measures.
- `DiscretizedFunction(ReferenceMeasureUniform)`: method for functions on Transformed Reference measures.

Examples

```
x=seq(0,1,length.out=100)
f = DiscretizedFunction(sin(x),0,1)
x=seq(0,1,length.out=1000)
max(abs(f(x)-sin(x)))
```

`dist.BData` *Distance method for BData objects*

Description

Computes inter-distances between all objects of ‘x‘

Usage

```
## S3 method for class 'BData'
dist(x, ...)

## S3 method for class 'BDataLogDensities'
dist(x, ...)

## S3 method for class 'BDataSplines'
dist(x, ...)
```

Arguments

`x`	a data container of class ‘BData‘ or derived
`...`	further parameters to ‘stats::dist()‘, i.e. among other ‘method‘ allows to use Manhattan or other Minkowski norms.

Value

a distance object, as ‘stats::dist()‘. The implementations rely on transformation to BDataSplines; the spline coefficients are first mapped to orthonormal coordinates via the Gram matrix of the basis, so that the default ‘method = "euclidean"‘ yields the exact Bayes-space (clr-L2) distance also for non-orthonormal bases such as ZB-splines. Other ‘method‘s (Manhattan, Minkowski, ...) operate on these orthonormal coordinates and depend on the chosen basis.

Functions

- `dist(BDataLogDensities)`: calculations for BDataLogDensities objects
- `dist(BDataSplines)`: calculations for BDataSplines objects

Examples

```
lbdl = BDataLogDensities(basis=B2N01Basis5, closed=TRUE,
  list(
    \(x)(dnorm(x,2,1,log=TRUE)),
    \(x)(dnorm(x,1,1,log=TRUE)),
    \(x)(dnorm(x,-1,1,log=TRUE)),
    \(x)(dnorm(x,-2,1,log=TRUE))
    ))
dist(lbdl)
```

---

distanceToCenter *Distance to Center*

Description

Computes the distance of a datapoint to the center of a dataset, for the definition of central regions.

Usage

```
distanceToCenter(x, na.rm = TRUE, ...)

## Default S3 method:
distanceToCenter(
  x,
  na.rm = TRUE,
  ...,
  distType = c("meandistance", "median", "spatialdepth")
)
```

Arguments

x	the dataset
na.rm	boolean whether to remove na values in dist
...	further parameters to dist
distType	"meandistance" for mean distance to datapoints, "median" for distance to median, "spatialdepth" for one minus spatial depth.

Value

A numeric vector of distances of each observation to the centre of the dataset.

Methods (by class)

- `distanceToCenter(default)`: default method

See Also

Other caseStatistics: caseE(), caseLogRatios(), caseMeans(), caseQuantiles(), caseSds(), caseVars(), centralityQuantile(), median.BData(), spatialDepth()

Examples

```
x = rnorm(100)

plot(x,distanceToCenter(x,distType="median"),col="green")
points(x,distanceToCenter(x,distType="meandist"))
points(x,distanceToCenter(x,distType="spatialdepth"),col="red")

data(DTD)
Xpdf    = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
distanceToCenter(Xpdf)
```

Div *Division of functional objects*

Description

This division is also able to divide functions and BayesSpace objects with constants. The quotient of two functions is a function that results in the quotient of the two results. The quotient of a BayesSpace objects (in BData) with a constant, is given by the BayesSpace operations of powering with the inverse value 1/y.

Usage

```
Div(x, y)

## S3 method for class '`function`'
Div(x, y)

## S3 method for class 'list'
Div(x, y)

## S3 method for class 'numeric'
Div(x, y)

## S3 method for class 'BData'
Div(x, y)

## S3 method for class '`function`'
Div.function(x, y)

## S3 method for class '`function`'
Div.list(x, y)
```

```
## S3 method for class 'list'
Div.function(x, y)

## S3 method for class 'numeric'
Div.function(x, y)

## S3 method for class '`function`'
Div.numeric(x, y)

## S3 method for class 'numeric'
Div.numeric(x, y)

## S3 method for class 'numeric'
Div.list(x, y)

## S3 method for class 'list'
Div.numeric(x, y)

## S3 method for class 'list'
Div.list(x, y)
```

Arguments

x	a functional object
y	a functional object

Value

A BData object of the same subclass as the first argument (the Bayes-space sum / difference / scalar multiple / quotient).

Methods (by class)

- `Div(`function`)`: method for functions
- `Div(list)`: method for vectorizing over lists
- `Div(numeric)`: method for numerical values seen as constant functions
- `Div(BData)`: method for operating any BData object (argument ‘x‘) by a numeric value (argument ‘y‘); defers calculations to ‘Mul()‘

Functions

- `Div.function(`function`)`: method for deviding two functions
- `Div.list(`function`)`: method for multiplying a function with each element of a list
- `Div.function(list)`: method for multiplying a function with each element of a list
- `Div.function(numeric)`: method for dividing a function with a constant
- `Div.numeric(`function`)`: method for deviding a constant with a function

- `Div.numeric(numeric)`: method for deviding two numeric values
- `Div.list(numeric)`: method for dividing each element of a list with a constant
- `Div.numeric(list)`: method for deviding a constant with each element of a list
- `Div.list(list)`: method for vectorized division of the elements of lists

See Also

Other Add: `Add()`, `Mul()`, `Sub()`

Examples

```
Div(sin,cos)
Div(list(sin,cos),sin)
Div(cos,list(sin,cos))
Div(list(sin,cos),list(sin,cos))
Div(list(sin,cos),10)
```

`DTD` *An example dataset for distributional data analysis*

Description

This is a simulated dataset, which can be used to try different tools for distributional data analysis. The idea is that there is a set of samples 'sample' from three regions, each containing many particles. From each particle we have observed a size parameter 'size' (volume equivalent diameter), a density 'density', and a shape parameter 'shape'

Usage

```
DTD
```

Format

A data frame with 20 rows (material samples) and 9 columns. Several columns hold one *function* per row (the particle PDFs / CDFs), and `data` holds one nested data frame per row:

sample The id of the material sample

region The region the sample comes from.

d50 A measured median particle size of the sample

shapePDF Particle shape distribution in sample, provide as a probability density function. Shape is measured by a measure for sphericity.

shapeCDF Particle shape distribution in sample, provided as Cummulative distribution.

sizePDF Particle size distribution in the sample (diameter), provide as probability density function. Size is measured by equivalent diameter. The sieving diameter relevant for d50 is slightly different.

sizeCDF Particle size distribution in the sample provided as cummulative distribution function (cdf).

rhoCDF Particle mass density distribution provided as cummulative distribution function (CDF). Particle density does not have probability density, because liberated particles containing only one mineral all can only have very specific densities (e.g. Quarz 2.25 kg/l and Magnetit at 5.2 kg/l)

data data.frames containing samples of the particles in each material sample. It has four columns: sample (the id of the sample), size (equivalent diameter of the particle), shape (sphericity), and density (mass density) of the particle.

Source

Simulated data, not based on real measurements. Generated by `data-raw/DTD.R` with `set.seed(47)` (N = 20 samples drawn from three regions A/B/C). Shipped as `data/DTD.rda`; see `data-raw/DTD.R` to regenerate.

Examples

```
data("DTD")
DTD
boxplot(DTD$d50)
# Looking at the size variable
plotFuncs(DTD$sizePDF)
plotFuncs(\(x)(dlnorm(x,meanlog=-0.2,sdlog=0.5)),add=TRUE,col="red",lwd=3)
RMForSize = ReferenceMeasureTransformed("plnorm",meanlog=-0.2,sdlog=0.5)
B2ForSize  = BayesSpace(RMForSize)
BasisForSize = BayesSpaceBasis(B2ForSize,nbasis=6)

sizePDFs = BDataPDFs(DTD$sizePDF,basis=BasisForSize)
plot(sizePDFs)
sizeDensities = as.BDataDensities(sizePDFs)
plot(sizeDensities)
sizeSplines   = as.BDataSplines(sizeDensities)
#plot(sizeSplines,add=TRUE,col="blue")
DTD$data[[1]]
summary(DTD$data[[1]])
```

`ensureType` *Returns the argument, but ensures that it has a certain class.*

Description

If the x has not the given type it, the function prints information about the problem and stops with an error.

Usage

```
ensureType(x, type)
```

Arguments

x	the objects that shall be checked.
type	a string representing the type to be checked with 'is(x,type)'

Value

x

Examples

```
ensureType(3,"numeric")
ensureType(diag(2),"matrix")
```

---

expectation *Computes expectations from reference measures*

Description

Computes expectations from reference measures

Usage

```
expectation(measure, f, ..., method)

## S3 method for class 'ReferenceMeasure'
expectation(
  measure,
  f,
  ...,
  method = c("default", "sample", "MonteCarlo", "integration"),
  tol = 0.01
)

## S3 method for class 'ReferenceMeasureDiscrete'
expectation(measure, f, ..., method = c("sample"))
```

Arguments

measure	the (probability) measure to compute the expectation from
f	a real valued random variable, represented as a vectorized function, for which the expectation shall be computed.
...	further arguments to f or the method used

`method` a method to be selected. Typical methods are "sample", "MonteCarlo" and "integrate", but only sample is availabel for all reference measures. "sample" uses the prestored sample, "MonteCarlo" uses the simulated function of the reference measure, and "integrate" uses numerical integration. Typically the prestored integration sample performs well with derivatives measures derived ReferenceMeasureUniform, ReferenceMeasusureTransformed, where it proves a dense grid in the area of measure, and ReferenceMeasure discrete where it uses precise computation using a weighted sample. For Cartesian products involving more than one continues measure due to the curse of dimensionality typically only MonteCarlo is expected to work reasonably well.

`tol` the absolute or relative tolerance (whichever is larger). accepted for methods with tolerance control.

Value

the numerically computed expected value of f with respect to the measure. The method

Methods (by class)

- `expectation(ReferenceMeasure)`: general method supporting only the "sample" method.
- `expectation(ReferenceMeasureDiscrete)`: a computing the results explicitly

See Also

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `ReferenceMeasureUniform()`, `StdReferenceMeasureUnif01`, `getSample()`, `isInDomain()`

Examples

```
m <- ReferenceMeasureUniform(0,10)
expectation(m, function(x) x^2)
```

`expectation.ReferenceMeasureUniform`
integrate a function over the uniform reference measure

Description

integrate a function over the uniform reference measure

Usage

```
## S3 method for class 'ReferenceMeasureUniform'
expectation(measure, f, ..., method = c("sample", "integrate"))
```

Arguments

measure	the ReferenceMeasureUniform object
f	function to be integrated
...	further arguments to [stats::integrate()]
method	a method for computing the expectation given as a string. The default argument shows possible choices. Default chooses the default for reference measure (typically "sample"), "sample" uses the reference sample of the measure, "MonteCarlo" uses iterative random simulation until a certain tolerance is reached, "integrate" is only possible in a few measures and tries a numerical integration.

Value

the integral of f over c(x[1],x[2])

Examples

```
RMU <- ReferenceMeasureUniform(a=-5,b=5)
expectationRMU <- expectation(measure=RMU,f=cos,method="integrate")
expectationRMU
```

---

findJumps *Finds jumps of a function of a certain minimum height*

Description

This is a helper tool to infere supports from cummulative density functions.

Usage

```
findJumps(f, a, b, minJump = 1/100, eps = 1e-07, digits = 5, n = 100)
```

Arguments

f	a vectorized function, whos jumps need to be found. The function is expected to be deterministic, nondecreasing (or nonincreasing) and continuous from the right.
a	a numeric value, the lower bound of the range to look for jumps
b	a numeric value, the upper bound of the range to look for jumps
minJump	a numeric value giving the minimum jump height, that is considered a jump.
eps	a numeric value given the distance of evaluations. If the distance of two evaluations drops to epsilon, and the jump height is achieved the higher value is rounded to digits and returned. if eps=0, the search goes until machine precision.
digits	an integer providing the digits of rounding, if search has finished accordint to the epsilon criterion.
n	the numer discretization intervals considered in each iteration step.

Value

a vector of numerical values at which the function f jumps by at least minJump in an eps distance, possibly rounded to digits.

Examples

```
findJumps(function(x) pbinom(x,4,0.2),-pi,17)
findJumps(pnorm,-3,3)
p <- findJumps(function(x) ifelse(x<pi,x,x+pi),0,6,eps=0)
p==pi # TRUE
```

---

fx — *Shortcut for creating functions of x*

Description

'fx(body)' is essentially equal to 'function(x) body'. This can be used to quickly create functions, with minimal syntactic burden.

Usage

```
fx(
  .body,
  template = function(x) {
 },
  .env = parent.frame()
)
```

Arguments

`.body`	the expression that will form the body of the function, it is not evaluated.
`template`	a template for the function, providing its formal parameters.
`.env`	the environment in which the body of the function is evaluated.

Value

a function with .body as body

Examples

```
f = fx(sin(x)+cos(x))
f
f(pi)
```

`getBayesSpace` *Generic function returning the Bayes space associated to the given object.*

Description

Generic function returning the Bayes space associated to the given object.

Usage

```
getBayesSpace(X, ...)

## S3 method for class 'BayesSpace'
getBayesSpace(X, ...)

## S3 method for class 'BayesSpaceBasis'
getBayesSpace(X, ...)

## S3 method for class '`NULL`'
getBayesSpace(X, ..., Default)

## S3 method for class 'BData'
getBayesSpace(X, ...)

## S3 method for class 'ReferenceMeasure'
getBayesSpace(X, ...)

## Default S3 method:
getBayesSpace(X, ..., Default = attr(X, "BayesSpace"))

## S3 method for class 'L2Space'
getBayesSpace(X, ...)

## S3 method for class 'L2Space'
getL2Space(X, ...)
```

Arguments

`X`	the object
`...`	further named arguments used by methods e.g. Default
`Default`	the Bayes Space returned if no Bayes Space is associated

Value

The associated [BayesSpace] object (class ‘"BayesSpace"‘).

Methods (by class)

- `getBayesSpace(BayesSpace)`: method for BayesSpace returning itself
- `getBayesSpace(BayesSpaceBasis)`: method for BayesSpaceBasis
- `getBayesSpace(`NULL`)`: method for NULL, returns default
- `getBayesSpace(BData)`: method for BData
- `getBayesSpace(ReferenceMeasure)`: creates the BayesSpace for the reference measure
- `getBayesSpace(default)`: default method tries to get an attribute
- `getBayesSpace(L2Space)`: returns BayesSpace with same reference measure

Functions

- `getL2Space(L2Space)`: method for BayesSpace returning itself

See Also

Other getBayesSpace: `getL2Space()`

Examples

```
B2Unif01Basis5
unclass(B2Unif01Basis5)
getBayesSpace(B2Unif01Basis5)
```

---

getBayesSpace.ReferenceMeasureCartesianProduct

Construct the product Bayes Space for a product reference measure

---

Description

Construct the product Bayes Space for a product reference measure

Usage

```
## S3 method for class 'ReferenceMeasureCartesianProduct'
getBayesSpace(X, ...)
```

Arguments

`X`	a ReferenceMeasureCartesianProduct to construct the BayesSpace for
`...`	further arguments to the getBayesSpace of the individual spaces

Value

A BayesSpaceCartesianProduct corresponding to the cartesian Product of Bayes measures, itself also a Bayes measure.

See Also

Other Cartesian: BayesSpaceBasis.BayesSpaceCartesianProduct(), BayesSpaceBasisCartesianProduct(), BayesSpaceCartesianProduct(), ReferenceMeasureCartesianProduct(), SupportCartesianProduct(), cartesianProduct(), getCartesianFactors(), subBasisIndices()

---

getBayesSpaceBasis — *Generic function returning the BayesSpaceBasis associated to the given object.*

Description

Generic function returning the BayesSpaceBasis associated to the given object.

Usage

```
getBayesSpaceBasis(X, ...)

## S3 method for class 'BayesSpaceBasis'
getBayesSpaceBasis(X, ...)

## S3 method for class 'BData'
getBayesSpaceBasis(X, ...)

## S3 method for class 'BayesSpace'
getBayesSpaceBasis(X, ...)

## S3 method for class 'varKernel'
getBayesSpaceBasis(X, ...)

## S3 method for class 'variationKernel'
getBayesSpaceBasis(X, ...)
```

Arguments

X	the object
...	further named arguments used by methods e.g. Default

Value

The associated [BayesSpaceBasis] object (class ‘"BayesSpaceBasis"‘).

Methods (by class)

- getBayesSpaceBasis(BayesSpaceBasis): returns itself
- getBayesSpaceBasis(BData): method for Datasets

- `getBayesSpaceBasis(BayesSpace)`: constructs a default vasis from the reference measure. Use ... as further parameters to generic constructor.
- `getBayesSpaceBasis(varKernel)`: method for varKernel
- `getBayesSpaceBasis(variationKernel)`: method for variationKernel

See Also

[BayesSpaceBasis], [getBayesSpace], [BayesSpace]

Examples

```
getBayesSpaceBasis(B2Unif01)
set.seed(47)
X = BDataGeneral(list(function(x) x, function(x) x^2),basis=B2Unif01Basis5)
getBayesSpaceBasis(X)
getBayesSpace(X)
getReferenceMeasure(X)
getBayesSpaceBasis(B2Unif01Basis5)
```

`getBayesSpaceBasisDimension` *the dimension of the basis*

Description

the dimension of the basis

Usage

```
getBayesSpaceBasisDimension(basis)

## S3 method for class 'BayesSpaceBasis'
getBayesSpaceBasisDimension(basis)
```

Arguments

`basis`	a BayesSpaceBasis

Value

a integer providing the dimension of the basis

Methods (by class)

- `getBayesSpaceBasisDimension(BayesSpaceBasis)`: the method for a standard BayesSpaceBasis

Examples

```
getBayesSpaceBasisDimension(B2Unif01Basis5)
getBayesSpaceBasisDimension(B2Unif01Basis7)
```

---

getBreaks *Common utilities for BDataHistograms objects*

Description

Common utilities for BDataHistograms objects

Usage

```
getBreaks(x)

## S3 method for class 'BDataHistograms'
getBreaks(x)
```

Arguments

x	object with a class derived from "BDataHistograms"

Value

depeding on the function, this can print the content, extract the breaks attribute in a convenient fashion or do subsetting.

Methods (by class)

- `getBreaks(BDataHistograms)`: Method for BDataHistograms

See Also

Other BDataHistograms: `BDataFrequencyHistograms()`, `as.BDataDensityHistograms()`, `as.BDataDensityHistogram`
`as.BDataFrequencyHistograms.list()`, `plot.BDataHistograms()`

Examples

```
set.seed(47)
H = hist(rnorm(30))
getBreaks(H)
X = as.BDataFrequencyHistograms(list(H,H),basis=B2N01Basis5)
getBreaks(X)
X
X[1]
```

getCartesianFactors *Decomposes a cartesian product into a list of its factors.*

Description

Decomposes a cartesian product into a list of its factors.

Usage

```
getCartesianFactors(...)
```

Arguments

...	the cartesian product or products to be decomposed

Value

If x is not a cartesian product the function should return list(x). if x is a cartesian product it should provide its factors. Check [SupportCartesianProduct()] for details about this latter structure.

See Also

Other Cartesian: BayesSpaceBasis.BayesSpaceCartesianProduct(), BayesSpaceBasisCartesianProduct(), BayesSpaceCartesianProduct(), ReferenceMeasureCartesianProduct(), SupportCartesianProduct(), cartesianProduct(), getBayesSpace.ReferenceMeasureCartesianProduct(), subBasisIndices()

getDimension *Gets number of elements in a Basis*

Description

Returns the dimension of an object.

Usage

```
getDimension(x, ...)

## S3 method for class 'BayesSpaceBasis'
getDimension(x, ...)

## S3 method for class 'BDataSplines'
getDimension(x, ...)

## S3 method for class 'matrix'
getDimension(x, ...)

## S3 method for class 'numeric'
getDimension(x, ...)
```

Arguments

x	the object
...	futher unused arguments for generic use

Value

an integer giving the number of dimensions of the object

Methods (by class)

- `getDimension(BayesSpaceBasis)`: returns number of elements in the basis
- `getDimension(BDataSplines)`: returns number of elements in the basis
- `getDimension(matrix)`: method for matrix returns number of columns
- `getDimension(numeric)`: method for numeric returns number of elements

Examples

```
getDimension(1:3)
getDimension(matrix(rnorm(10),ncol=2))
getDimension(B2N01Basis5)
```

---

getDiscretizer *Computes Discretization points for nonequidistant discretization*

Description

This is a helper function to create nonequidistant discretizations

Usage

```
getDiscretizer(x, nEval = 513, ...)

## S3 method for class 'BayesSpaceBasis'
getDiscretizer(x, nEval = 513, ...)

## S3 method for class 'ReferenceMeasureDiscrete'
getDiscretizer(x, nEval, ...)

## S3 method for class 'ReferenceMeasureUniform'
getDiscretizer(x, nEval = 513, ...)

## S3 method for class 'ReferenceMeasureTransformed'
getDiscretizer(x, nEval = 513, ...)
```

Arguments

`x`	typically a ReferenceMeasure
`nEval`	the number of discretization points
`...`	further parameters with details for more complex references

Value

an object that can be use together with its evaluation on object$all to create a DiscreteFunction

Methods (by class)

- `getDiscretizer(BayesSpaceBasis)`: method for BayesSpaceBasis
- `getDiscretizer(ReferenceMeasureDiscrete)`: method for ReferenceMeasure discrete
- `getDiscretizer(ReferenceMeasureUniform)`: method for ReferenceMeasureUniform
- `getDiscretizer(ReferenceMeasureTransformed)`: method for ReferenceMeasureTransformed

Examples

```
getDiscretizer(B2Unif01Basis5)
getDiscretizer(B2N01Basis5)
```

---

getL2Space — *Generic function returning the L2Space associated to the given object.*

Description

Generic function returning the L2Space associated to the given object.

Usage

```
getL2Space(X, ...)

## S3 method for class 'BayesSpaceBasis'
getL2Space(X, ...)

## S3 method for class '`NULL`'
getL2Space(X, ..., Default)

## S3 method for class 'BData'
getL2Space(X, ...)

## S3 method for class 'ReferenceMeasure'
getL2Space(X, ...)
```

```
## Default S3 method:
getL2Space(X, ..., Default = getL2Space(attr(X, "BayesSpace")))

## S3 method for class 'BayesSpace'
getL2Space(X, ...)
```

Arguments

X	the object
...	further named arguments used by methods e.g. Default
Default	the Bayes Space returned if no L2 Space is associated

Value

The associated [L2Space] object (class ‘"L2Space"‘).

Methods (by class)

- `getL2Space(BayesSpaceBasis)`: method for BayesSpaceBasis
- `getL2Space(`NULL`)`: method for NULL, returns default
- `getL2Space(BData)`: method for BData
- `getL2Space(ReferenceMeasure)`: creates the BayesSpace for the reference measure
- `getL2Space(default)`: default method tries to get an attribute
- `getL2Space(BayesSpace)`: returns L2Space with same reference measure

See Also

Other getBayesSpace: `getBayesSpace()`

Examples

```
getL2Space(B2Unif01Basis5)
set.seed(47)
X = BDataGeneral(list(function(x) x, function(x) x^2),basis=B2Unif01Basis5)
getL2Space(X)
getL2Space(StdReferenceMeasureUnif01)
```

`getMetricTensor` *Metric tensor of a space or basis*

Description

The metric tensor is the Gram matrix ‘S‘, used in ‘|v|^2 = v’ S v‘ to compute the metric of the space when ‘v‘ is the vector of basis coefficients. ‘S‘ contains the scalar products of the basis elements, ‘S[i,j] = <psi_i, psi_j>‘.

Usage

```
getMetricTensor(x, ...)
```

Arguments

`x`	the space or basis for which the metric tensor shall be returned.
`...`	further generic parameters for the computation

Details

For an orthonormal basis ‘S‘ is the identity matrix. The default zero-integral B-spline (ZB-spline) basis is **not** orthonormal, so ‘S‘ is a banded, non-identity matrix; inner products ([scalarProduct]), norms ([norm.BData]), metric variance ([mvar][mvar.BData]) and PCA ([princomp.BData]) all rely on it. You can check whether a basis is orthonormal by comparing the result to ‘diag(nrow(S))‘.

Value

A symmetric positive-definite matrix ‘S‘ (the Gram matrix / metric tensor), with ‘S[i,j]‘ the inner product of basis elements ‘i‘ and ‘j‘.

See Also

[BayesSpaceBasis], [scalarProduct], [princomp.BData]

Examples

```
getMetricTensor(B2Unif01Basis5)   # not the identity: ZB-splines are not orthonormal
```

`getPDF` *returns the probability density function*

Description

The probability density function, also known as the Lebesgue density of the measure is returned. In case of datasets a dataset of such functions is returned. (To be encoded as BDataPDF)

Usage

```
getPDF(X, ...)

## S3 method for class 'ReferenceMeasureUniform'
getPDF(X, ...)

## S3 method for class 'ReferenceMeasureTransformed'
getPDF(X, ...)

## S3 method for class 'BData'
getPDF(X, ...)
```

Arguments

`X`	the distribution hows Lebesgue density shall be returned
`...`	potential futher arguments for the computational method

Value

a function (or a list of functions if a dataset of distributions is given) that can compute the probably density function.

Methods (by class)

- `getPDF(ReferenceMeasureUniform)`: method for ReferenceMeasureUniform
- `getPDF(ReferenceMeasureTransformed)`: method for ReferenceMeasureUniform
- `getPDF(BData)`: method for BData objects, returns BDataPDFs (TODO)

Examples

```
x = seq(-1,2,by=0.1)
f1 = getPDF(StdReferenceMeasureUnif01)
plot(x,f1(x))
f2 = getPDF(StdReferenceMeasureN01)
plot(x,f2(x),type="l")
f3 = getPDF(BDataLogDensities(list(sin),basis=B2Unif01Basis5))
plot(x,f3(x),type="l")
getPDF(BDataLogDensities(list(sin,cos,\(x)(1)),basis=B2Unif01Basis5))
```

`getReferenceMeasure` *Getter for the ReferenceMeasure of an object*

Description

BayesSpaces and L2Spaces are defined on underlying reference measures. This function returns this measure.

Usage

```
getReferenceMeasure(x)

## S3 method for class 'BayesSpace'
getReferenceMeasure(x)

## S3 method for class 'L2Space'
getReferenceMeasure(x)

## S3 method for class 'ReferenceMeasure'
getReferenceMeasure(x)

## Default S3 method:
getReferenceMeasure(x)
```

Arguments

`x`	An object that is somehow related to a reference measure, like e.g. a ReferenceMeasure, a BayesSpace, a BayesSpaceBasis, or a BData object.

Value

The reference-measure object (class ‘"ReferenceMeasure"‘) underlying ‘x‘.

Methods (by class)

- `getReferenceMeasure(BayesSpace)`: method for [BayesSpace]
- `getReferenceMeasure(L2Space)`: method for [L2Space]
- `getReferenceMeasure(ReferenceMeasure)`: method for ReferenceMeasure, returns the measusre itself
- `getReferenceMeasure(default)`: default method tries to get a BayesSpace and the corresponding reference measure from that.

See Also

[ReferenceMeasure], [BayesSpace], [L2Space]

Examples

```
StdReferenceMeasureUnif01
getReferenceMeasure(B2Unif01)
getReferenceMeasure(B2Unif01Basis5)
getReferenceMeasure(L2Space(StdReferenceMeasureUnif01))
getReferenceMeasure(BDataSamples(replicate(2,runif(10)),basis=B2Unif01Basis5))
```

`getSample` *Get a sample from the reference measure*

Description

This provides a sample of data form the support of the reference measure, e.g. to test functions or to be used as integration points. The sample can optionally come as a weighted sample to ensure integration. In order to achieve good performance and reproducability the sample is typically fixed and not random, except in cases where no good algorithm for a deterministic grid could be implemented.

This function is here for fast and reproducable Monte Carlo integration, with respect to a measure. Unlike a [simulate] it is allowed to return the same sample in different calls and to provide default size rather than a requested one.

Usage

```
getSample(x, ...)

## S3 method for class 'ReferenceMeasure'
getSample(x, ...)

## S3 method for class 'ReferenceMeasureCartesianProduct'
getSample(x, ...)

## S3 method for class 'BayesSpace'
getSample(x, ...)

## S3 method for class 'L2Space'
getSample(x, ...)

getSample(x, ...)

## S3 method for class 'ReferenceMeasureDiscrete'
getSample(x, ...)

hasWeights(x)

## Default S3 method:
hasWeights(x, ...)
```

```
getWeights(x, ...)
```

Arguments

x	the ReferenceMeasure or distribution
...	further parameters to simulate, which might or might not be called.

Value

a sample of the distribution in the form that would be expected from a dataset, like e.g. a vector or a data.matrix.

Methods (by class)

- `getSample(ReferenceMeasure)`: method for referencemeasure, please overwrite
- `getSample(ReferenceMeasureCartesianProduct)`: method for ReferenceMeasureCartesianProduct
- `getSample(BayesSpace)`: method for BayesSpace
- `getSample(L2Space)`: method for L2Space
- `getSample(ReferenceMeasureDiscrete)`: method for discrete measures, returning a weighted sample

Functions

- `hasWeights()`: generic method checking whether its weighted sample
- `hasWeights(default)`: default method checking for a weights attribute.
- `getWeights()`: generic method for retrieving weights of a sample

See Also

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `ReferenceMeasureUniform()`, `StdReferenceMeasureUnif01`, `expectation()`, `isInDomain()`

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `ReferenceMeasureUniform()`, `StdReferenceMeasureUnif01`, `expectation()`, `isInDomain()`

Examples

```
getSample(B2Unif01)

hist(getSample(StdReferenceMeasureUnif01))
mean(cos(getSample(StdReferenceMeasureN01)))
```

`getSplineCoefs` *Coefficients of a single BayesSpace element.*

Description

The coefficients are computed with respect to the given basis. Only a single density data element my be provide, but in many different froms including single case BData* objects, density functions, samples, frequency and density histograms. A density histogram is a histogram structure without counts. Samples need to provide at least minSampleSize elements, to avoid simply misinterpreting inadmissible object types as samples.

Usage

```
getSplineCoefs(x, ..., basis)

## S3 method for class 'BDataSplines'
getSplineCoefs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class '`function`'
getSplineCoefs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'hist'
getSplineCoefs(x, ..., basis = getBayesSpaceBasis(x))

## S3 method for class 'BData'
getSplineCoefs(x, ..., basis = getBayesSpaceBasis(x))

## Default S3 method:
getSplineCoefs(x, ..., minSampleSize = 10, basis = getBayesSpaceBasis(x))
```

Arguments

`x`	a single object representing a distribution. A function will be interpreted as density unless.
`...`	generic arguments supporting the converter. Either minSampleSize or relevant argumens of as.BDataSpline.
`basis`	the BayesSpaceBasis to be used.
`minSampleSize`	the minimum length admissible for samples

Value

a numerical vector of coefficients in the given basis

Methods (by class)

- `getSplineCoefs(BDataSplines)`: the method for BDataSpline objects.
- `getSplineCoefs(` `function` `)`: method for density functions. Convert log densities to densities by applying 'Exp()'.
- `getSplineCoefs(hist)`: method for histograms
- `getSplineCoefs(BData)`: method for distributions represented as a single element in a BData structure.
- `getSplineCoefs(default)`: interpreting it as a sample of data. It must have at least 10 observations.

Examples

```
getSplineCoefs(function(x) x^2,basis=B2Unif01Basis5)
getSplineCoefs(rbeta(1000,2,2),basis=B2Unif01Basis5)
```

---

hasSameBayesSpace *Checks whether two objects have the same BayesSpace*

Description

The function fails if no BayesSpace is associated to any of the objects.

Usage

```
hasSameBayesSpace(x, y)
```

Arguments

x	some object with a BayesSpace
y	some object with a BayesSpace

Value

a boolean

Examples

```
hasSameBayesSpace(B2Unif01Basis5,B2Unif01Basis7) # TRUE
hasSameBayesSpace(B2Unif01Basis5,B2N01Basis5) # FALSE
```

hasSameBayesSpaceBasis

Checks whether the associated BayesSpaceBasis of to objects is the same

Description

Checks whether the associated BayesSpaceBasis of to objects is the same

Usage

```
hasSameBayesSpaceBasis(x, y)
```

Arguments

x	an object with an associated BayesSpaceBasis
y	an object with an associated BayesSpaceBasis

Value

a boolean

Examples

```
x = BDataSamples(list(runif(100)), basis=B2Unif01Basis5)
hasSameBayesSpaceBasis(B2Unif01Basis5,x) # TRUE
hasSameBayesSpaceBasis(B2Unif01Basis5,B2Unif01Basis7) # FALSE
hasSameBayesSpace(B2Unif01Basis5,B2N01Basis5) # FALSE
```

idt.BData

Isometric default representations for BData objects

Description

The isometric default transform for BData objects is return as BDataSplines..

Usage

```
## S3 method for class 'BData'
idt(x, ...)

## S3 method for class 'BData'
idtInv(x, orig = attr(x, "orig"), ...)

## S3 method for class 'BDataSplines'
idtInv(x, orig = attr(x, "orig"), ...)
```

Arguments

x	object to be transformed
...	extra parameters for generic functionality
orig	object specifying the desired target representation for ‘x‘

Value

a BSpline representation of the Distributional data

Functions

- idtInv(BData): invert representation of an object ‘x‘ according to the representation of ‘orig‘
- idtInv(BDataSplines): returns directly to splines

See Also

Other BayesSpaceConverters: as.BDataDensities(), as.BDataDensityHistograms(), as.BDataFrequencyHistograms as.BDataHistograms(), as.BDataLogDensities(), as.BDataLogPDFs(), as.BDataPDFs(), as.BDataSamples(), as.BDataSplines(), toHistogram()

Examples

```
# BData densities
bdld = BDataLogDensities(list(\(x)(x^2-x^3), \(x)(sin(x*pi))), basis=B2Unif01Basis20)
plot(bdld, col=c(2,4))
coefs = as.BDataSplines(bdld)
mn = colMeans(coefs)
recasted = idtInv(mn, orig=bdld)
class(recasted)
f = as.function(recasted[[1]])
curve(f, add=TRUE)

# BSpline coefficients
coefs=matrix(1:15,nrow=3)
bssplines=BDataSplines(coefs,B2Unif01Basis5)
cf = idt(bssplines)
mn = colMeans(cf)
idtInv(mn, orig=bssplines)
```

---

isClosed — *Returns whether its know that the object is closed or centered*

Description

Returns whether its know that the object is closed or centered

Usage

```
isClosed(x)

## Default S3 method:
isClosed(x)

## S3 method for class 'BData'
isClosed(x)

isCentered(x)

## Default S3 method:
isCentered(x)

## S3 method for class 'BData'
isCentered(x)

## S3 method for class 'BDataSplines'
isCentered(x)

## S3 method for class 'BayesSpaceBasis'
isCentered(x)
```

Arguments

x	the object to be checked

Value

a boolean, if true the system believes that the object is already closed

Methods (by class)

- `isClosed(default)`: default methods always returns FALSE
- `isClosed(BData)`: BData returns true if it was closed or declared closed

Functions

- `isCentered()`: Checks for centering
- `isCentered(default)`: default methods always returns FALSE
- `isCentered(BData)`: BData returns true if it was closed or declared closed
- `isCentered(BDataSplines)`: BDataSplines returns true if the basis is centered
- `isCentered(BayesSpaceBasis)`: BayeSpaceBasis requests whether the basis is considered centered

Examples

```
isClosed(dnorm) # FALSE, because we have not way of knowing
isClosed(BDataDensities(list(dnorm),basis=B2N01Basis5,closed=TRUE)) # TRUE by assertation
isClosed(BDataDensities(list(Mul(2,dnorm)),basis=B2N01Basis5,close=TRUE))  # TRUE by making it
isCentered(dnorm) # FALSE, because we have not way of knowing
isCentered(BDataDensities(list(dnorm),basis=B2N01Basis5,closed=TRUE)) # FALSE
isCentered(cen(BDataDensities(list(Mul(2,dnorm)),basis=B2N01Basis5)))  # TRUE by making it
```

---

isInDomain *Checks whether values are in the domain of the ReferenceMeasure*

Description

The function acts vectorized, i.e. in case of numbers or other atomic objects it expects a vector of the given type. In case of values that can not be stored in a vector it expects a list of the elements.

Usage

```
isInDomain(x, referenceMeasure)

## S3 method for class 'BayesSpace'
isInDomain(x, referenceMeasure)

## S3 method for class 'L2Space'
isInDomain(x, referenceMeasure)

## S3 method for class 'BayesSpaceBasis'
isInDomain(x, referenceMeasure)

## S3 method for class 'ReferenceMeasureTransformed'
isInDomain(x, referenceMeasure)

## S3 method for class 'ReferenceMeasureDiscrete'
isInDomain(x, referenceMeasure)

## S3 method for class 'ReferenceMeasureCartesianProduct'
isInDomain(x, referenceMeasure)
```

Arguments

`x`	values to be checked to be in the reference measure
`referenceMeasure`	the referenceMeasure to be checked against. It can be provided indirectly.

Value

a logical vector of the same length as ‘x‘, ‘TRUE‘ where the corresponding value lies in the domain (support) of the reference measure and ‘FALSE‘ otherwise.

Methods (by class)

- `isInDomain(BayesSpace)`: method for BayesSpaces
- `isInDomain(L2Space)`: method for L2Spaces
- `isInDomain(BayesSpaceBasis)`: method for BayesSpaces
- `isInDomain(ReferenceMeasureTransformed)`: method for ReferenceMeasureTransformed. The domain is always an open interval, where 0<cdf(x)<1.
- `isInDomain(ReferenceMeasureDiscrete)`: method for discrete Reference Measures
- `isInDomain(ReferenceMeasureCartesianProduct)`: method for product measures

See Also

isInDomain.ReferenceMeasureUniform

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `ReferenceMeasureUniform()`, `StdReferenceMeasureUnif01`, `expectation()`, `getSample()`

Examples

```
isInDomain(1:10,B2Unif01)
```

---

isInDomain.ReferenceMeasureUniform

assess whether a vector of points belong to the domain of the uniform reference measure

---

Description

unlike typical measures the domain is closed, i.e. the extreme values are considered part of the support.

Usage

```
## S3 method for class 'ReferenceMeasureUniform'
isInDomain(x, referenceMeasure)
```

Arguments

`x`	a dataset of points to be checked
`referenceMeasure`	the ReferenceMeasureUniform object, providing the domain

Value

a logical vector of TRUE/FALSE with a length length(x)

Examples

```
isInDomain(c(1:10),ReferenceMeasureUniform(0,1))
```

`isValidBayesSpaceBasis`

Checks that an object is a BayesSpaceBasis

Description

Checks that an object is a BayesSpaceBasis

Usage

```
isValidBayesSpaceBasis(X)
```

Arguments

X	the object to be checked

Value

boolean

Examples

```
isValidBayesSpaceBasis(B2Unif01Basis7)
```

`isValidBayesSpaceElement`

Checks whether an object is an element (or dataset of elements) of a BayesSpace

---

Description

May different R objects can be interpreted as elements of a BayesSpace including density functions, histograms, and BData objects.

Usage

```
isValidBayesSpaceElement(X, basis = NULL)

## S3 method for class '`function`'
isValidBayesSpaceElement(X, basis = NULL)

## S3 method for class 'histogram'
isValidBayesSpaceElement(X, basis = NULL)

## S3 method for class 'BData'
isValidBayesSpaceElement(X, basis = NULL)

## Default S3 method:
isValidBayesSpaceElement(X, basis = NULL)
```

Arguments

`X`	the element to be checked
`basis`	an optional BayesSpaceBasis in which it should be an element. If 'basis' is given a TRUE shall only be returned if can be seen as an element of that BayesSpace. That however is not fully implemented yet.

Value

boolean, FALSE if it is definitely not an element of a BayesSpace

Methods (by class)

- `isValidBayesSpaceElement(`function`)`: functions are valid BayesSpaceElements
- `isValidBayesSpaceElement(histogram)`: histograms are valid BayesSpaceElements
- `isValidBayesSpaceElement(BData)`: BData objects contain valid BayesSpaceElements if they are for the same BayesSpace
- `isValidBayesSpaceElement(default)`: method for samples, tries to check whether the values are in the domain.

Examples

```
isValidBayesSpaceElement(sin)
isValidBayesSpaceElement(hist(runif(100), plot=FALSE))
```

kdeDensity *Computes an evaluable kernel density estimate on a dataset*

Description

The distance is measured in the metric $d(x, y) = \sqrt{(x-y)^t M (x-y)}$ given by the 'Metric' tensor M. The kernel density is estimated as

Usage

```
kdeDensity(X, ..., bandwidth, Metric, kernel, K, Var, sdFactor)

## S3 method for class 'matrix'
kdeDensity(
  X,
  ...,
  bandwidth = NULL,
  Metric = if (is.null(bandwidth)) invertVariance(Var * sdFactor^2) else
    diag(1/bandwidth^2, nrow = ncol(X)),
  K = 1/sqrt(2 * pi)^(NCOL(X) - 1) * sqrt(det(Metric)),
  kernel = dnorm,
  Var = var(X),
  sdFactor = (0.2)^(1/NCOL(X))
)

## S3 method for class 'data.frame'
kdeDensity(
  X,
  ...,
  bandwidth = NULL,
  Metric = if (is.null(bandwidth)) invertVariance(Var * sdFactor^2) else
    diag(1/bandwidth^2, nrow = ncol(X)),
  K = 1/sqrt(2 * pi)^(NCOL(X) - 1) * sqrt(det(Metric)),
  kernel = dnorm,
  Var = var(X),
  sdFactor = (0.2)^(1/NCOL(X))
)

## S3 method for class 'list'
kdeDensity(
  X,
  ...,
  bandwidth = NULL,
  Metric = if (is.null(bandwidth)) 1/(Var * sdFactor^2) else 1/bandwidth^2,
  K = sqrt(Metric),
  kernel = dnorm,
  Var = var(X),
```

```
  sdFactor = 0.2,
  minus = "-"
)

## S3 method for class 'numeric'
kdeDensity(
  X,
  ...,
  support = Support(a = -Inf, b = Inf, lim = range(X)),
  bandwidth = NULL,
  Metric = if (is.null(bandwidth)) 1/(Var * sdFactor^2) else 1/bandwidth^2,
  K = sqrt(Metric),
  kernel = dnorm,
  Var = var(X),
  sdFactor = 0.2,
  minus = "-"
)
```

Arguments

X	the sample to compute the kernel density estimated from
...	futher generic parameter
bandwidth	the bandwidth to be used, if null the metric tensor is computed in a different way.
Metric	the metric tensor used to scale the distances before passed to the kernel.
kernel	a kernel function evaluated for a scaled distance. In case of univariate data the kernel is also evaluated with just the scaled difference (i.e. signed distances) and can thus be asymetric.
K	the factor to scale the kernel density to get an integral of 1. The factor can be ignored in base of BData objects.
Var	the variance (matrix) of the dataset to base a Metric calculation on
sdFactor	the factor to multiply the directional standard deviation width to the the directional bandwidth.
minus	the function to be used to compute the distance of elements
support	the support on which to compute the kernel density estimate. Finite supports lead to a mirroring at the support limits. I future also an independent estimate of the density at discrete points is planned.

Details

$f(x) = \frac{1}{N}\sum_{i=1}^{n} scale * kernel(d(x, X_i))$

The computation is done explicity in all calls to the resulting function.

Value

a function that can be evaluated with a dataset similar to X

Methods (by class)

- `kdeDensity(matrix)`: method for multivariate datasets provided as matrix
- `kdeDensity(data.frame)`: method for multivariate datasets provided as data.frame
- `kdeDensity(list)`: method for datasets provided as list. The minus operator must be defined on the elements.
- `kdeDensity(numeric)`: method for datasets provided as a numeric vector.

Examples

```
X = rnorm(10)
x=seq(-4,4,by=0.1)
f = kdeDensity(X,sdFactor=0.5)
plot(x,f(x),type="b")

# data generation and kde function construction
xx = cbind(rnorm(100, sd=2.5), rnorm(100, sd=0.5))
kdxy = kdeDensity(xx)
# check that the integral is approx 1:
x0 = seq(from=-2.5, to=2.5, length.out=25)
xys = expand.grid(x0*2.5, x0*0.5)
zs =  kdxy(xys)
(dx0 = x0[2]-x0[1])
(A = sum(zs)*dx0^2*2.5*0.5) # bad approx but within expectations given low precision
# compare with kde from stats::density for one marginal:
dim(zs) = c(25,25)
dx = density(xx[,1], kernel="gaussian")
dx |> plot()
lines(x0*2.5, apply(zs,1,sum)*dx0*0.5/A, col=2)
rug(xx[,1])
xx = rnorm(1000, sd=2.5) # simulate some data
# generate and plot the kde function
kdx = kdeDensity(xx)
plot(kdx, from=-5, to=5)
# ensure that the integral is approx 1
xs = seq(from=-5, to=5, by=0.01)
ys = kdx(xs)
sum(ys)*0.01             #  approx 1.0
# graphically compare with kde from stats::density
dx = density(xx, kernel="gaussian")
with(dx, lines(x, y, col=2))
plot(dx)
curve(kdx, from=-4, to=4, col=2, add=TRUE)
```

L2Space — *Creates the L2-space for a Reference Measure*

Description

An L2-space is a Hilbert space of with respect to the reference measure square integrable functions. They are vector addition and multiplication with a scalar is done on the results. The scalar product is defined integrating the product of two functions over the reference measure.

Usage

```
L2Space(x = StdReferenceMeasureUnif01, ...)

## S3 method for class 'ReferenceMeasure'
L2Space(x, ...)

## S3 method for class 'BayesSpace'
L2Space(x, ...)
```

Arguments

x	the ReferenceMeasure of the L2 space
...	further arguments for generic use

Value

An object of class ‘"L2Space"‘: a list holding the reference measure.

Methods (by class)

- `L2Space(ReferenceMeasure)`: Method for ReferenceMeasure
- `L2Space(BayesSpace)`: Method for BayesSpaces.

See Also

[ReferenceMeasure]

Other BayesSpace: `BayesSpace()`

Examples

```
space = L2Space(ReferenceMeasureUniform(0,10))
space
bs = BayesSpace(ReferenceMeasureUniform(0,10))
space2 = L2Space(bs)
space2
```

listDataElementAccess *Accesses list elements with an expression*

Description

This is the generic behind ' to modify the behaviour of '

Usage

```
listDataElementAccess(X, .body, .env)
```

Arguments

X	the list on which to operate
.body	an expression, which is either a single name, in which case a the correponding '$'-operator is applied to the list, or a full expression, in which '.' takes the role of the selected element.
.env	the environment the .body expression shall be evaluated in

Value

a list from the results of body evaluated in .env, with . standing in for the elements of X, or if .body is a name with .$name evaluted.

Examples

```
L = list(list(name = "a", val = 1), list(name = "b", val = 2))
# a bare name applies the corresponding `$` accessor to each element
listDataElementAccess(L, quote(name), environment())
# a full expression uses `.` for each list element
listDataElementAccess(L, quote(.$val * 10), environment())
```

ListDataEnv *Makes a list includable in a data.frame*

Description

data.frame objects treat lists as variables and not as cases. This functions presents the list as data to a data.frame. For better printing the elements of the list will be represented as a name.

Usage

```
ListDataEnv

ListData(x)

ListDataRecover(hash)

storingDigester(x, env = environment(x))

## S3 method for class 'ListData'
format(x, ...)

## S3 method for class 'BData'
format(x, ...)

## S3 method for class 'ListData'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'BData'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	a list like to be converted to something includable in a data.frame.
`hash`	the displayed name of the object as character string
`env`	the environment keeping the object in the hash, if none provided, .GlobalEnv is used.
`...`	for generic use
`row.names`	the row names for the dataset
`optional`	unused, see [as.data.frame] for details

Format

An object of class `environment` of length 1.

Details

The first letter of the name is the first letter of the class, then comes a '_' and then a alphanumerical digest, of the objects. As long as the object is in reach, it can be inspected by this name.

Afterwards an NA of class "ForgottenObject" is returned.

Value

the object x with an additional "ListData" class attribute

Methods (by generic)

- `format(ListData)`: formats as the digest
- `as.data.frame(ListData)`: converts to a data.frame

Functions

- `ListDataEnv`: the environment in which the digests can be found
- `ListDataRecover()`: recovers object from the digest
- `storingDigester()`: returns a recoverable digest of the object
- `format(BData)`: formats as the digest
- `as.data.frame(BData)`: converts to a data.frame

Examples

```
FX = ListData(
   list(function(x) x,
        function(x) x^2,
        function(x) x^3))
FX
D = data.frame(X=FX,Y=FX)
D
N = format(FX)[[1]]
N
ListDataRecover(N)
attach(ListDataEnv)
get(N)
N2 = storingDigester(function(x) sin(x))
N2
ListDataRecover(N2)
detach(ListDataEnv)
```

Ln — *Functional mapping a function to its exp version*

Description

Exp maps f(x) to Exp(f(x)). Ln is its inverse operation.

Usage

```
Ln(f)

## S3 method for class 'numeric'
Ln(f)

## S3 method for class '`function`'
```

```
Ln(f)

## S3 method for class 'list'
Ln(f)

Exp(f)

## S3 method for class 'numeric'
Exp(f)

## S3 method for class '`function`'
Exp(f)

## S3 method for class 'list'
Exp(f)
```

Arguments

f	the function (or a constant)

Value

same type as f but exponentiated

Methods (by class)

- `Exp(numeric)`: numeric method just evaluates exp
- `Exp(`function`)`: Function method maps the function
- `Exp(list)`: list method iterates in a vectorized manner over the list

Functions

- `Ln()`: Inverse of Exp
- `Ln(numeric)`: method for numerics just applies the natural logarithm
- `Ln(`function`)`: Inverse functional mapping
- `Ln(list)`: list method operates in a vectorized way over the list

Examples

```
Exp(1)
f = function(x) x^2
expf = Exp(f)
expf
lnf = Ln(f)
Exp(lnf)
Ln(expf)
```

makeFunctionOuter — *Makes a vectorized 2-ary function into one that returns a matrix*

Description

A vectorized function of two arguments is made returning the result for each pair structured as a matrix, or if it already returns an n-ary array, as an n+1-ary array with the last dimension splitted

Usage

```
makeFunctionOuter(f)
```

Arguments

f	the 2-ary working on vectors, lists or rows of matrices or datasets

Value

a function like f but returning the result for each pair of input structured with one more dimension.

Examples

```
makeFunctionOuter(\(x,y) exp(-abs(x-y)))(1:4,1:4)
```

makeLimitStable — *Makes a real function returning limits at undefined points*

Description

The function identifies Punctures in the definition of a function, by getting a non finite result (Inf,-Inf,Nan) and than asks computeLimit to compute the corresponding directional limit (for finite values given by d). The value can still be infinite.

Usage

```
makeLimitStable(f, x0 = 0.01)
```

Arguments

f	a function, to be make stable against problems with limits
x0	the direction from which limits shall be taken.

Value

a numerically stabilised version of ‘f‘: a function that evaluates ‘f‘ but replaces any non-finite result (Inf, -Inf, NaN) by the corresponding directional limit obtained from [computeLimit] (which may itself still be infinite).

mapToRealLineWithSupport

Gets coordinates on an axis for possibly non real data

Description

Gets coordinates on an axis for possibly non real data

Usage

```
mapToRealLineWithSupport(X, support, d = 1)
```

Arguments

X	the data
support	a [Support] object giving the continuous, discrete or mixed support of the data.
d	the dimension of the space, that may be used

Value

the value(s) of ‘X‘ mapped onto the ‘support‘, i.e. possibly the same values of ‘X‘ (i.e. if the support is a subset of the real line) or the indices corresponding to the atoms in the support equal to the values of ‘X‘ (if the support is a discrete set of points or of categories).

maxFunc

Compute Maximum and Minimum Functions

Description

Compute Maximum and Minimum Functions

Usage

```
maxFunc(..., na.rm = FALSE, bound = NULL, Funcs = c(...))

minFunc(..., na.rm = FALSE, bound = NULL, Funcs = c(...))
```

Arguments

...	arbitrary list of vectorized functions or vectorized functions, behaving returning numerical values of the same size as a result to the same arguments
na.rm	boolean, should na results be discared?
bound	a limiting function. All values exceeding (or smaller than for minFunc) are ignored in the maximum.
Funcs	the list of all functions that should be used in the maximum or minimum

Value

a function taking like the given functions, and returning the maximum (or minium) of the values not exceeding the boundary.

Functions

- `minFunc()`: computes the minimum

Examples

```
F = maxFunc(\(x) dnorm(x),\(x) dnorm(x,mean=2),bound=\(x) rep(0.3,length(x)))
x=seq(-2,5,by=0.01)
plot(x,F(x))
F = minFunc(\(x) dnorm(x),\(x) dnorm(x,mean=2),bound=\(x) rep(0.1,length(x)))
plot(x,F(x))
```

`mean.BDataSplines` *Computes the distribution mean for BDataSplines.*

Description

This function simply averages the spline coefficients and returns a correponding spline.

Usage

```
## S3 method for class 'BDataSplines'
mean(x, ..., method = c("classical", "mcd", "mve"))

## S3 method for class 'BDataDensities'
mean(x, ..., method = c("functions", "classical", "mcd", "mve"))

## S3 method for class 'BDataPDFs'
mean(x, ..., method = c("functions", "classical", "mcd", "mve"))

## S3 method for class 'BDataLogPDFs'
mean(x, ..., method = c("functions", "classical", "mcd", "mve"))

## S3 method for class 'BDataLogDensities'
mean(x, ..., method = c("functions", "classical", "mcd", "mve"))
```

Arguments

`x`	the BDataSplines object over which to average
`...`	unused arguments required for generic use
`method`	the method to be chosen (as in [cov.rob]), functions does operates according to BayesSpace formulas directly on the densities, classical uses the spline representation, mcd and mve apply these robust methods on the spline coefficients.

Value

a BDataSplines with a single observation representing the mean

Functions

- `mean(BDataDensities)`: method for full densities
- `mean(BDataPDFs)`: method for full PDFs
- `mean(BDataLogPDFs)`: method for full logPDFs
- `mean(BDataLogDensities)`: method for full logDensities

Examples

```
data(DTD)
Xpdf    = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
Xspline = as.BDataSplines(Xpdf)
plot(Xspline,col=2:12)
meanXspline = mean(Xspline)
plot(meanXspline,col=1,lwd=3,add=TRUE)

plot(Xspline,as="PDF")
plot(meanXspline,col="red",lwd=3,add=TRUE,as="PDF")

Xdensities = as.BDataDensities(Xpdf)

plot(Xdensities)
meanXdensities = clo(mean(Xdensities))
plot(meanXdensities,col="green",lwd=7,add=TRUE)
plot(meanXspline,col="red",lwd=3,add=TRUE,as="density")

plot(Xpdf)
meanXpdf = mean(Xpdf)
plot(clo(meanXpdf),col="green",lwd=7,add=TRUE)
plot(meanXspline,col="red",lwd=3,add=TRUE,as="PDF",close=TRUE)
plot(as.BDataPDFs(meanXdensities),col="blue",lwd=3,add=TRUE,close=TRUE)
plot(as.BDataPDFs(mean(as.BDataLogPDFs(Xpdf))),col="orange",lwd=3,add=TRUE,close=TRUE)
plot(as.BDataPDFs(mean(as.BDataLogDensities(Xpdf))),col="magenta",lwd=3,add=TRUE,close=TRUE)
```

`median.BData` *A distributional median*

Description

The distributional median is the distribution (from a dataset), that minimizes the sum over the Bayes Space distances in the dataset. The sample median selects the distribution from the dataset that minimizes this distance (or maximizes the spatial depth).

Usage

```
## S3 method for class 'BData'
median(x, na.rm = TRUE, ..., distType = c("meandistance", "spatialdepth"))
```

Arguments

x	a BData object
na.rm	if true missing distances will be set to the maximum distance
...	further parameters to dist
distType	"meandistance" for mean distance to datapoints, "spatialdepth" for the spatial depth.

Value

An object of the same type as x, but with only the median, defined as the value with the smallest distance to all others (distType "mean"), or the largest spatial depth (distType "spatialdepth").

See Also

Other caseStatistics: caseE(), caseLogRatios(), caseMeans(), caseQuantiles(), caseSds(), caseVars(), centralityQuantile(), distanceToCenter(), spatialDepth()

Examples

```
data(DTD)
Xpdf    = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
median(Xpdf)
```

Mul *Multiplication of functional objects*

Description

This multiplication is also able to multiply functions and BayesSpace objects with constants. The product of two functions is a function that results in the product of the two results. Product of a BayesSpace objects (in BData), is given by the BayesSpace operations, which is given powering the density with respect to the reference measure, with the given scalar.

Usage

```
Mul(x, y)

## S3 method for class '`function`'
Mul(x, y)

## S3 method for class 'list'
```

```
Mul(x, y)

## S3 method for class 'numeric'
Mul(x, y)

## S3 method for class '`function`'
Mul.function(x, y)

## S3 method for class '`function`'
Mul.list(x, y)

## S3 method for class 'list'
Mul.function(x, y)

## S3 method for class 'numeric'
Mul.function(x, y)

## S3 method for class '`function`'
Mul.numeric(x, y)

## S3 method for class 'numeric'
Mul.numeric(x, y)

## S3 method for class 'BData'
Mul.numeric(x, y)

## S3 method for class 'numeric'
Mul.list(x, y)

## S3 method for class 'list'
Mul.numeric(x, y)

## S3 method for class 'list'
Mul.list(x, y)

## S3 method for class 'BDataDensities'
Mul(x, y)

## S3 method for class 'BDataLogDensities'
Mul(x, y)

## S3 method for class 'BDataPDFs'
Mul(x, y)

## S3 method for class 'BDataLogPDFs'
Mul(x, y)

## S3 method for class 'BDataSplines'
```

```
Mul(x, y)
```

Arguments

x	a functional object
y	a functional object

Value

A BData object of the same subclass as the first argument (the Bayes-space sum / difference / scalar multiple / quotient).

Methods (by class)

- `Mul(`function`)`: method for multiplying functions
- `Mul(list)`: method for vectorized operation over lists
- `Mul(numeric)`: method for numerical values treated as constant functions
- `Mul(BDataDensities)`: BayesSpace scaling multiplication
- `Mul(BDataLogDensities)`: BayesSpace scaling multiplication
- `Mul(BDataPDFs)`: BayesSpace scaling multiplication
- `Mul(BDataLogPDFs)`: BayesSpace scaling multiplication for log-PDFs
- `Mul(BDataSplines)`: BayesSpace scaling multiplication

Functions

- `Mul.function(`function`)`: method for operating two functions
- `Mul.list(`function`)`: method for multiplying a function with each element of a list
- `Mul.function(list)`: method for multiplying a function with each element of a list
- `Mul.function(numeric)`: method for multipying a function with a numeric value
- `Mul.numeric(`function`)`: method for multiplying numeric constant with a function
- `Mul.numeric(numeric)`: method for multiplying two constant values
- `Mul.numeric(BData)`: method for multiplying a numeric object by a BData object
- `Mul.list(numeric)`: method for multiplying each element of a list with a numeric value
- `Mul.numeric(list)`: method for multiplying a numeric value with each element of a list
- `Mul.list(list)`: method for vectorized multiplication of the elements of a list.

See Also

Other Add: `Add()`, `Div()`, `Sub()`

Examples

```
Mul(sin,cos)
Mul(list(sin,cos),sin)
Mul(cos,list(sin,cos))
Mul(list(sin,cos),list(sin,cos))
Mul(list(sin,cos),10)
```

`mvar.BData` *Compute the metric variance*

Description

The metric variance is the expected squared distance from the mean in the metric of the enclosing space. By construction the metric variance computed from a spline representation should be smaller than metrix variances computed on functional data, because the spline captures only a part of the variance.

Usage

```
## S3 method for class 'BData'
mvar(x, ..., meanx = mean(x), method = "classical")

## S3 method for class 'BDataSplines'
mvar(x, ..., method = c("classical", "mcd", "mve"), meanx = mean(x, method))

## S3 method for class 'BData'
msd(x, ...)
```

Arguments

`x`	some BData dataset.
`...`	further parameters for the methods to compute it (expectation)
`meanx`	an external mean estimate
`method`	the method of variance estimation

Value

a positive real number representing the metric variance

Functions

- `mvar(BDataSplines)`: a specialized method for BDataSpline, which does not require integration.
- `msd(BData)`: compute the metric standard deviation, proportional to the square root of the metric variance

Examples

```
data(DTD)
Xpdf    = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
Xsplines= as.BDataSplines(Xpdf)
mvar(Xsplines)
 mvar(Xsplines,method="mcd")
mvar(as.BDataDensities(Xpdf))
```

```
mvar(as.BDataLogDensities(Xpdf))
mvar(as.BDataLogPDFs(Xpdf))
mvar(Xpdf)
```

---

nCases *Accessing cases regardless how a dataset is stored.*

Description

Accessing cases regardless how a dataset is stored.

Usage

```
nCases(X)

## Default S3 method:
nCases(X)

selectCases(X, I)

## Default S3 method:
selectCases(X, I)
```

Arguments

X	the dataset, may be a vector a matrix, a dataframe, or if implmenented something more advanced.
I	the Index vector indexing the cases

Value

returns the subset of cases.

Methods (by class)

- `nCases(default)`: default method working for most know types of datasets, by using the first array dimension (if applicable) or the length as number of cases.

Functions

- `selectCases()`: selects the cases given by the index I
- `selectCases(default)`: selects the cases given by the index I

Examples

```
nCases(1:5)
nCases
```

norm *Computes the Bayes-Space norm of BData objects*

Description

Computes the Bayes-Space norm of BData objects

Usage

```
norm(x, ...)

## S3 method for class 'BData'
norm(x, ..., measure = getReferenceMeasure(x))

## S3 method for class 'BDataSplines'
norm(x, ...)
```

Arguments

x	a BData object
...	further arguments to expectation.
measure	the reference measure to on which to compute the L2 norm of the clr

Value

A numeric vector of the L2 (Bayes-space) norms, one per observation.

Functions

- `norm()`: generic functionality computing Euclidean norms of vectors (default method) and redirecting to ‘base::norm()‘ for matrices.
- `norm(BDataSplines)`: fast exact norm for BDataSplines via the Gram matrix of the basis: the squared norm of each observation is the quadratic form of its coefficient vector with the metric tensor S.

Examples

```
lbdl = BDataLogDensities(basis=B2N01Basis5, closed=TRUE,
  list(
    \(x)(dnorm(x,2,1,log=TRUE)),
    \(x)(dnorm(x,1,1,log=TRUE)),
    \(x)(dnorm(x,-1,1,log=TRUE)),
    \(x)(dnorm(x,-2,1,log=TRUE))
    ))
norm(lbdl)
```

`numberOfCases` *A generic function computing the number of cases.*

Description

Different representations of datasets provide its elements in a very different way. This function provides a uniform interface to figure get the number of cases in a dataset.

Usage

```
numberOfCases(x)

## S3 method for class 'data.frame'
numberOfCases(x)

## Default S3 method:
numberOfCases(x)
```

Arguments

`x`	a dataset, of which the number of cases shall be counted. It can be a numeric vector, a matrix, a data.frame, a compositional dataset, or a BData object.

Value

an integer

Methods (by class)

- `numberOfCases(data.frame)`: method for dataframes
- `numberOfCases(default)`: method for lists and numeric, BDataDensities,...

Examples

```
data(iris)
numberOfCases(iris$Sepal.Length)
numberOfCases(iris)
numberOfCases(lapply(1:numberOfCases(iris),function(i) iris[i,]))
```

pdf2Support *Guesses the support from a pdf*

Description

The function is designed to allow a quick start into distributional data analysis and plotting.

Usage

```
pdf2Support(
  pdf,
  starts = exp(seq(-20, 20, by = 0.2)),
  minDensityFactor = dnorm(qnorm(alpha/2))/dnorm(0),
  alpha = 0.01
)
```

Arguments

pdf	a probability density function of a real random variable
starts	a vector of values, where the algorithm should look for a nonzero pdf. If the pdf is nowhere positive the computation will fail.
minDensityFactor	a factor of the maximum density used to compute an effective range. should be a numeric in (0,0.5).
alpha	a way to specify minDensityFactor by providing an alpha level for the normal distribution.

Value

a Support object with the guessed support

See Also

Other Support: Support(), cdf2Support(), plotFuncs(), sample2Support()

Examples

```
pdf2Support(dnorm)
pdf2Support(dunif)
```

plot.BayesSpaceBasis *plot the 'BayesSpaceBasis'*

Description

plot the 'BayesSpaceBasis'

Usage

```
## S3 method for class 'BayesSpaceBasis'
plot(
  x,
  ...,
  as = c("is", "logPDF", "PDF", "CDF", "density", "logDensity", "clr", "coefs")
)
```

Arguments

x	'BayesSpaceBasis' object
...	further arguments to plot.BDataSplines
as	tells in which version to plot the basis

Value

typically some representation of the data plotted

Examples

```
print(B2Unif01Basis5)
```

plot.BDataHistograms *Plot histograms as segments*

Description

Plot a histograms dataset as a set of "stairs" using matplot.

Usage

```
## S3 method for class 'BDataHistograms'
plot(
  x,
  probability = is(x, "BDataDensityHistograms"),
  xlab = "x",
  ylab = ifelse(probability, "density", "frequency"),
  ...
)
```

Arguments

x	a "BDataHistograms" object, of any class
probability	logical, should the histogram represent density (forced if ‘x‘ is of class "BDataDensityHistograms") or frequency?
xlab	label for the x-axis
ylab	label for the y-axis
...	further arguments to [graphics::matplot()]

Value

nothing, called for its side effect to produce a plot. Each histogram is represented only as its outline.

See Also

Other BDataHistograms: BDataFrequencyHistograms(), as.BDataDensityHistograms(), as.BDataDensityHistogram
as.BDataFrequencyHistograms.list(), getBreaks()

Examples

```
x = lapply(rep(1000,4), runif)
bdfhx = as.BDataFrequencyHistograms(x, breaks="Sturges",basis=B2Unif01Basis5)
plot(bdfhx)
bdfhx2 = as.BDataFrequencyHistograms(x, breaks=sqrt(0:10)/sqrt(10),basis=B2Unif01Basis5)
plot(bdfhx2, probability=TRUE)
# plot(bdfhx2) # this generates a warning
```

---

plot.BDataSamples *Plotting BDataSample Objects*

Description

Plotting BDataSample Objects

Usage

```
## S3 method for class 'BDataSamples'
plot(x, ..., pch = 20, method = "stack", as = c("points"), add = FALSE)
```

Arguments

x	the data
...	further generic graphic parameters
pch	the plotting character
method	the method argument of the stripchart method.
as	the from of plotting
add	a boolean, if FALSE a new plot is generated.

Value

‘NULL‘ invisibly, called for the side effect of producing a plot.

Examples

```
s <- BDataSamples(list(rnorm(200), rnorm(200,1)), basis=B2N01Basis5)
plot(s)
```

---

plot.BDataSplines *Plot a set of spline functions (using visualisation tools from fda)*

Description

Plot a set of spline functions (using visualisation tools from fda)

Usage

```
## S3 method for class 'BDataSplines'
plot(
  x,
  ...,
  as = c("coef", "PDF", "CDF", "density", "logPDF", "logDensity", "clr"),
  close = TRUE
)
```

Arguments

x	’BDataSplines’ object
...	further arguments to plot, plotFunc, plotCoefs
as	a string defining how the spline should be plotted. The default "coef" is a parallel coordinate plot of the coefficients. Other options include "PDF", "CDF", "density", "logPDF", "logDensity", and "clr" (centered log-ratio, showing the additive representation in Bayes space).
close	if true, and as is not plotting the spline, but a closable BData, this is used as an argument to [clo].

Value

nothing or an invisible version of the object actually plotted

See Also

Other BDataSplines: BDataSplines()

Examples

```
X = BDataSplines(matrix(rnorm(30),ncol=5),basis=B2Unif01Basis5)
plot(X, as = "coef")
plot(X, as = "PDF")
plot(X, as = "clr")  # centered log-ratio representation
```

---

plot.princomp.BData *Plot Method for PCA of Distributional Data*

Description

A versatile plotting function for visualizing principal component analysis results on distributional data. Provides multiple plot types for comprehensive exploration of the PCA results, including the unique ability to visualize loadings as functional perturbations of the mean distribution.

Usage

```
## S3 method for class 'princomp.BData'
plot(
  x,
  type = c("screeplot", "variance", "scores", "eigenfunctions", "loadings", "biplot",
    "covariance", "reconstruction", "boxplot"),
  choices = 1:2,
  scale = 1,
  pc.biplot = FALSE,
  main = NULL,
  col = NULL,
  lwd = 1,
  lty = 1,
  legend = TRUE,
  labels = FALSE,
  ...,
  npcs = min(10, length(x$sdev)),
  as = c("PDF", "density", "logPDF", "logDensity", "clr", "coef"),
  scale.sdev = 1
)
```

Arguments

x	An object of class ‘princomp.BData‘ as returned by princomp.BData.
type	Character string specifying the type of plot to produce:

‘"screeplot"‘ Bar chart showing the percentage of variance explained by each principal component. Useful for deciding how many components to retain.

‘"variance"‘ Line plot showing cumulative variance explained as more components are included. Horizontal reference lines at 80%, 90%, and 95% help identify cutoff points.

‘"scores"‘ Scatter plot of observations in the space of two principal components. Useful for identifying clusters, outliers, or patterns in the data.

‘"eigenfunctions"‘ Key plot for interpretation. Shows how the distribution shape changes along each principal component direction. The mean distribution is shown along with perturbations in the positive and negative PC directions. This is the standard SFPCA terminology (Hron et al. 2016).

‘"loadings"‘ Alias for ‘"eigenfunctions"‘ (kept for backwards compatibility).

‘"biplot"‘ Combines scores and eigenfunctions in a single plot. Calls `biplot.princomp.BData`.

‘"covariance"‘ Heatmap showing the functional covariance surface. This visualizes which parts of the domain co-vary: high values indicate that when one part of the distribution is above average, the other part tends to be as well. Useful for understanding the correlation structure across the support of the distributions.

‘"reconstruction"‘ Shows the reconstructed distributions using only the first ‘npcs‘ principal components. Useful for assessing how well a reduced number of components approximates the original data.

‘"boxplot"‘ Boxplots showing the distribution of scores for each principal component. Useful for visualizing the spread of observations along each PC direction and identifying potential outliers.

`choices` Integer vector of length 2 specifying which principal components to plot (for ‘type="scores"‘ and ‘type="biplot"‘). Default is ‘c(1, 2)‘.

`scale` Scaling factor for loadings in biplot (passed to `biplot.princomp.BData`). Default is 1.

`pc.biplot` Logical. If ‘TRUE‘, use Gabriel’s biplot scaling. Default is ‘FALSE‘.

`main` Character string for plot title. If ‘NULL‘, an appropriate default is generated based on the plot type.

`col` Colors for plotting. For loadings plot: first color is for the mean, subsequent colors for each PC. For scores plot: color for points/labels.

`lwd` Line width for functional plots (loadings). Default is 1.

`lty` Line type (currently not directly used; solid for positive direction, dashed for negative).

`legend` Logical. If ‘TRUE‘ (default), add a legend to the loadings plot.

`labels` For ‘type="scores"‘: Controls labeling of observations:

‘FALSE‘ (Default) Show only points, no labels.

‘TRUE‘ Show observation numbers (1, 2, 3, ...) instead of points.

character vector Use custom labels for each observation.

This helps identify which distribution each point represents.

`...` Additional graphical parameters passed to underlying plot functions.

`npcs` Number of principal components to display in screeplot, variance plot, or loadings plot. Default is ‘min(10, length(x$sdev))‘.

`as` For loadings plot: how to display the distributions. One of:

‘"PDF"‘ Probability density function (default). Shows distributions as proper densities that integrate to 1.

‘"density"‘ Density with respect to the reference measure.

‘"logPDF"‘ Log of the probability density function.

‘"logDensity"‘ Log density with respect to reference measure.

‘"clr"‘ Centered log-ratio transformation. Shows the additive structure in Bayes space. The clr is centered around zero, making it easy to see positive and negative deviations from the geometric mean. This is the natural representation for Bayes space PCA.

‘"coef"‘ Raw spline coefficients (parallel coordinates plot). Useful for seeing which coefficients drive each PC.

`scale.sdev` Numeric scaling factor for the loadings plot. Controls how far from the mean the perturbed distributions are shown:

‘1‘ Show distributions at ±1 standard deviation from mean (default).

‘2‘ Show distributions at ±2 standard deviations (more extreme).

‘NA‘ Show unit loadings without scaling by standard deviation.

Details

Understanding the Eigenfunctions Plot

The ‘type="eigenfunctions"‘ plot is the most important visualization for interpreting SFPCA results on distributional data. It answers the question: "How does the distribution shape change as we move along each principal component?"

The plot shows:

- **Black thick line**: The mean distribution (center of the data).
- **Colored solid lines**: Distributions at +‘scale.sdev‘ standard deviations from the mean along each PC direction (positive perturbation).
- **Colored dashed lines**: Distributions at -‘scale.sdev‘ standard deviations from the mean along each PC direction (negative perturbation).

For example, if PC1 captures variation in the "location" of distributions, the PC1+ line will be shifted right and PC1- will be shifted left compared to the mean. If PC1 captures variation in "spread", PC1+ will be wider and PC1- will be narrower.

Choosing the Number of Components

Use the screeplot and variance plot to decide how many components are meaningful:

- **Screeplot**: Look for an "elbow" where the variance drops off. Components before the elbow are typically retained.
- **Variance plot**: Choose enough components to explain a target percentage (e.g., 80% or 90%) of total variance.

Value

Invisibly returns the data used for plotting (varies by plot type).

Author(s)

Matthias Templ

See Also

princomp.BData for performing the PCA.

biplot.princomp.BData for detailed control over biplots.

screeplot.princomp.BData for the standalone screeplot function.

Other PCA: biplot.princomp.BData(), plot.princomp.BData.grouped(), plot.princomp.BData.multivariate(), predict.princomp.BData(), princomp.BData(), screeplot.princomp.BData(), summary.princomp.BData()

Examples

```
## Setup: Create PCA result for examples
meanDist <- BDataPDFs(list(dunif), basis = B2Unif01Basis7)
gaussProc <- B2GaussianDistribution(meanDist, function(x,y) exp(-4*abs(x-y)^1.7))
pcaResult <- princomp(as.BDataSplines(simulate(gaussProc, 30, seed = 42)))

## Screeplot: Decide how many components to keep
plot(pcaResult, type = "screeplot")
# Interpretation: The bars show the percentage of total variance explained by
# each PC. Here, PC1 and PC2 dominate, suggesting that the main structure in
# the data can be captured by just two components. The "elbow" after PC2 or
# PC3 indicates where additional components add little explanatory power.

## Cumulative variance: What percentage is explained?
plot(pcaResult, type = "variance")
# Interpretation: This shows how total explained variance accumulates as more
# components are included. The dashed reference lines help identify how many
# components are needed to reach common thresholds (80%, 90%, 95%). Here,
# approximately 3 components explain over 90% of the variance.

## Score plot: Explore structure in the data
plot(pcaResult, type = "scores")
# Interpretation: Each point is one distribution projected onto PC1 and PC2.
# The spread along PC1 (x-axis) shows variation in the first mode; spread
# along PC2 (y-axis) shows the second mode. Clusters indicate groups of
# similar distributions; outliers are distributions with unusual shapes.

## Score plot with labels: Identify individual distributions
plot(pcaResult, type = "scores", labels = TRUE)
# Using labels=TRUE replaces points with observation numbers (1, 2, 3, ...),
# allowing you to identify which specific distribution each position
# represents. This is useful for tracking outliers or interesting cases
# back to the original data.

## Eigenfunctions plot: THE KEY INTERPRETIVE PLOT
plot(pcaResult, type = "eigenfunctions", npcs = 2)
# Interpretation: The black curve is the mean (average) distribution. For
# each PC, solid lines show how distributions look when they score HIGH on
# that component, dashed lines show LOW scores. Compare the colored curves
# to the black mean to see what "high PC1" or "low PC2" means in terms of
# actual distribution shape (e.g., shifted location, changed spread, altered
# skewness, or different modality).
```

```
## More extreme perturbations (2 standard deviations)
plot(pcaResult, type = "eigenfunctions", npcs = 2, scale.sdev = 2)
# Using scale.sdev=2 shows more extreme deviations from the mean, making
# the differences between PC directions more visible.

## Show loadings as log-PDFs (useful for seeing tail behavior)
plot(pcaResult, type = "loadings", npcs = 2, as = "logPDF")
# The log-PDF representation emphasizes differences in the tails of the
# distributions, which can be important for detecting outlier behavior.

## Show loadings in CLR (centered log-ratio) representation
plot(pcaResult, type = "loadings", npcs = 2, as = "clr")
# The CLR representation shows the additive structure in Bayes space.
# Values are centered around zero, making it easy to see which parts of the
# distribution are above or below the geometric mean. This is the natural
# representation for PCA in Bayes space.

## Covariance surface: See which parts of the domain co-vary
plot(pcaResult, type = "covariance")
# Interpretation: This heatmap shows functional covariance Cov(f(s), f(t))
# across the domain. High values (red) indicate that when one part of the
# distribution is above average, the other tends to be as well. The diagonal
# shows variance at each point. Off-diagonal patterns reveal correlation
# structure (e.g., if left and right tails move together or oppositely).

## Reconstruction: How well do k PCs approximate the data?
plot(pcaResult, type = "reconstruction", npcs = 2)
# Shows all distributions reconstructed using only the first 2 PCs.
# Compare to the original data to assess information loss.
# If reconstructions look similar to originals, 2 PCs are sufficient.

plot(pcaResult, type = "reconstruction", npcs = 1)
# Using only 1 PC shows more loss of detail - useful for understanding
# what information each additional PC captures.

## Boxplot: Distribution of scores across PCs
plot(pcaResult, type = "boxplot", npcs = 4)
# Shows the spread of observations along each PC direction.
# Useful for identifying outliers (points beyond whiskers) and comparing
# the amount of variation captured by each component. The percentage
# labels show variance explained by each PC.
```

`plot.princomp.BData.grouped`

Plot Method for Grouped PCA of Distributional Data

Description

Creates multi-panel comparison plots for grouped PCA results. Each group is shown in a separate panel, allowing visual comparison of PCA results across different subgroups of the data.

Usage

```
## S3 method for class 'princomp.BData.grouped'
plot(
  x,
 type = c("screeplot", "variance", "scores", "eigenfunctions", "loadings", "covariance",
    "reconstruction", "boxplot"),
  mfrow = NULL,
  main = NULL,
  ...
)
```

Arguments

x	An object of class ‘princomp.BData.grouped‘ as returned by princomp.BData with a ‘group‘ argument.
type	Character string specifying the type of plot. Same options as plot.princomp.BData: ‘"screeplot"‘, ‘"variance"‘, ‘"scores"‘, ‘"loadings"‘, ‘"covariance"‘, or ‘"reconstruction"‘.
mfrow	Optional vector of c(nrow, ncol) for panel layout. If ‘NULL‘, panels are arranged automatically in a single row.
main	Character string for overall title. If ‘NULL‘, group names are used as individual panel titles.
...	Additional arguments passed to plot.princomp.BData.

Details

This function sets up a multi-panel plotting layout and calls plot.princomp.BData for each group. The layout is automatically determined based on the number of groups, but can be overridden using ‘mfrow‘.

Common use cases:

- Compare loadings across groups to see if different subgroups have different modes of variation.
- Compare screeplots to assess if dimensionality differs by group.
- Compare score plots to see internal structure within each group.

Value

Invisibly returns the grouped PCA object.

Author(s)

Matthias Templ

See Also

princomp.BData for performing grouped PCA. plot.princomp.BData for single-group plot options.

Other PCA: biplot.princomp.BData(), plot.princomp.BData(), plot.princomp.BData.multivariate(), predict.princomp.BData(), princomp.BData(), screeplot.princomp.BData(), summary.princomp.BData()

Examples

```
# Create grouped PCA using age distribution data
data("age")

# Set up basis
rmAge <- ReferenceMeasureUniform(0, 100)
B2Age <- BayesSpace(rmAge)
basisAge <- BayesSpaceBasis(B2Age, nbasis = 7)

# Create histograms and convert to splines
breaks <- c(0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 100)
ageHist <- BDataDensityHistograms(age$distributions, breaks = breaks, basis = basisAge)
ageSplines <- as.BDataSplines(as.BDataDensities(ageHist))

# Perform grouped PCA by sex
pcaGrouped <- princomp(ageSplines, group = age$data$sex)
pcaGrouped

# Compare screeplots by group
plot(pcaGrouped, type = "screeplot")

# Compare loadings by group
plot(pcaGrouped, type = "loadings", npcs = 2)

# Custom layout (2 rows)
plot(pcaGrouped, type = "scores", mfrow = c(2, 1))
```

---

plot.princomp.BData.multivariate

Plot Method for Multivariate SFPCA of Distributional Data

Description

Creates visualization plots for multivariate SFPCA (2D-SFPCA) results. Eigenfunctions span multiple groups and are shown with one panel per group.

Usage

```
## S3 method for class 'princomp.BData.multivariate'
plot(
```

```
  x,
  type = c("screeplot", "variance", "scores", "eigenfunctions", "loadings"),
  npcs = 2,
  pcs = c(1, 2),
  as = "PDF",
  ...
)
```

Arguments

x	An object of class ‘princomp.BData.multivariate‘ as returned by princomp.BData with ‘group‘ and ‘multivariate = TRUE‘.
type	Character string specifying the type of plot: **"screeplot"** Variance explained by each component **"scores"** Scatter plot of PC scores **"eigenfunctions"** Eigenfunctions shown per group (default) **"loadings"** Alias for eigenfunctions
npcs	Number of principal components to display. Default is 2.
pcs	Which PCs to use for scores plot. Default is c(1, 2).
as	Representation for eigenfunction plots: "PDF", "clr", "density", etc.
...	Additional graphical arguments.

Details

For multivariate SFPCA, eigenfunctions are joint across all groups. The eigenfunctions plot shows how each group's component of the eigenfunction affects the distribution shape.

Value

Invisibly returns the multivariate PCA object.

Author(s)

Matthias Templ

See Also

princomp.BData for performing multivariate SFPCA.

Other PCA: biplot.princomp.BData(), plot.princomp.BData(), plot.princomp.BData.grouped(), predict.princomp.BData(), princomp.BData(), screeplot.princomp.BData(), summary.princomp.BData()

Examples

```
data("age")
basisAge <- BayesSpaceBasis(BayesSpace(ReferenceMeasureUniform(0, 100)), nbasis = 7)
breaks <- c(0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 100)
ageHist <- BDataDensityHistograms(age$distributions, breaks = breaks, basis = basisAge)
ageSplines <- as.BDataSplines(as.BDataDensities(ageHist))
```

```
pcaMulti <- princomp(ageSplines, group = age$data$sex, multivariate = TRUE)
plot(pcaMulti, type = "eigenfunctions", npcs = 2)
```

---

plot.varKernel *Plotting varKernel objects*

Description

Plots a varKernel object as an image or contour plot.

Usage

```
## S3 method for class 'varKernel'
plot(
  x,
  ...,
  support = getSupport(x),
  SD = supportDiscretization(support, nEval),
  C = x(SD$all, SD$all),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  cex = par("lwd"),
  type = c("both", "image", "contour"),
  nEval = 513,
  add = FALSE,
  xlim = getSupport1DPlottingLimits(support),
  ylim = rev(xlim),
  zlim = range(breaks),
  breaks = pretty(c(C), ncol),
  ncol = 10,
  cols = grDevices::hcl.colors(length(breaks) - 1, "YlOrRd", rev = TRUE),
  dWidth = (xlim[2] - xlim[1])/200,
  xlab = "x",
  ylab = xlab,
  zlab = "c",
  notColour = "#FFFFFF"
)

## S3 method for class 'variationKernel'
plot(
  x,
  ...,
  support = getSupport(x),
  SD = supportDiscretization(support, nEval),
```

```
  V = x(SD$all),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  cex = par("lwd"),
  type = c("both", "image", "contour"),
  nEval = 513,
  add = FALSE,
  xlim = getSupport1DPlottingLimits(support),
  ylim = rev(xlim),
  zlim = range(breaks),
  breaks = pretty(V, ncol),
  ncol = 10,
  cols = grDevices::hcl.colors(length(breaks) - 1, "YlOrRd", rev = TRUE),
  dWidth = (xlim[2] - xlim[1])/200,
  xlab = "x",
  ylab = "x",
  zlab = "c",
  notColour = "#FFFFFF"
)
```

Arguments

`x`	an function of two arguments to be plotted. The function is expected to be in an outer vectorized way, i.e. that it returns a matrix, if both arguments are vectors.
`...`	further arguments for plot
`support`	the support of the first argument of the function
`SD`	a support discretization object used to discretized the function
`C`	the variance kernel evaluated at SD$all
`col`	a (vector of) plotting colors recycled over functions
`pch`	a (vector of) plotting chars recycled over functions used for the discrete support
`lty`	a (vector of) line types recycled over the functions used for the lines of the continuous support
`lwd`	line width for the countour plot
`cex`	see par: size of plotting symbols
`type`	either "both", "image", "contour" defining whether it should be represented by an image or a contour or both.
`nEval`	the number of discretization points for the continuous support
`add`	a boolean, if TRUE functions are added to an existing plot, if FALSE a new plot is generated.
`xlim`	the xlim of a new plot
`ylim`	the ylim of a new plot, if NULL it will be computed from the range of the functions.

zlim	the limits for plotting the covariance values
breaks	the values for which to change color and where to draw the countour lines.
ncol	the number of colours, it is one less then the number of breaks
cols	the colours to represent the values between the breaks
dWidth	discrete Width, the width to draw the discrete covariance values, which would theoretically take only an infinitely think line.
xlab	the label of the x-axis
ylab	the label of the y-axis y-axis
zlab	the label of the covariance values, not used yet
notColour	the colour to plot outside the range of values
V	the variation kernel evaluated at SD$all

Value

'NULL' invisibly, called for the side effect of producing an image and/or contour plot of the kernel.

Functions

- `plot(variationKernel)`: method for variationKernel

Examples

```
C = varKernel(function(x,y) exp(-outer(x,y,"-")^2),basis=B2N01Basis5)
plot(C)
V = variationKernel(function(x) 1-exp(-outer(x,x,"-")^2),basis=B2N01Basis5)
plot(V)
```

---

plotCoefs *Plot multiple coefficients in a parallel coordinate plot*

Description

The function currently uses the same axes,

Usage

```
plotCoefs(
  x,
  type = "b",
  ...,
  commonAxis = TRUE,
  sequence = 1:ncol(x),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
```

```
  lwd = par("lwd"),
  cex = par("cex"),
  nEval = NULL,
  add = FALSE,
  xlim = c(0.5, length(sequence) + 0.5),
  ylim = range(c(x)),
  xlab = "j",
  axes = rep(list(c(0, 1)), ncol(x)),
  ylab = expression(c["." * j]),
  xaxt = "s"
)
```

Arguments

x	a matrix of coefficients
type	character plotting type: "p" points, "l" lines, "b" both, "n" nothing
...	further arguments for plot
commonAxis	logical, if TRUE all coefficients are plotted on the same axis. if FALSE individual axes are plotted. FALSE is not yet implemented.
sequence	gives the sequence of the coefficients to be plotted.
col	a (vector of) plotting colors recycled over functions
pch	a (vector of) plotting chars recycled over functions and used for the points
lty	a (vector of) line types recycled over the functions used for the lines of the continuous support
lwd	a (vector of) line widths recycled over the function used for the lines of the continuous support
cex	a (vector of) character extends recycled over the data rows and used for points
nEval	the number of discretization points for the continuous support
add	a boolean, if TRUE functions are added to an existing plot, if FALSE a new plot is generated.
xlim	the xlim of a new plot
ylim	the ylim of a new plot, if NULL it will be computed from the range of the functions.
xlab	the label of the x-axis
axes	a list of scalings for the individual parallel axes (not yet implemented)
ylab	the label of the y-axis
xaxt	a character specifying the type of xaxis, please see par or plot for a description. Use "n" to remove it and "s" to plot.

Value

The coefficient matrix ‘x‘ (invisibly), called for the side effect of producing the parallel coordinate plot.

Examples

```
plotCoefs(matrix(rnorm(20),nrow=5))
```

---

plotFuncs *Helper function to plot multiple functions*

Description

This function uses a loop over lines and points to plot multiple functions with different parameters into one plot.

Usage

```
plotFuncs(
  x,
  ...,
  support = getSupport(x),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  nEval = 1500,
  add = FALSE,
  xlim = getSupport1DPlottingLimits(support),
  ylim = NULL,
  xs = xL[!xL %in% X],
  xL = if (xlim[1] != xlim[2]) seq(xlim[1], xlim[2], length.out = nEval),
  X = getSupportAtoms(support),
  xlab = "x",
  ylab = "f(x)",
  close = FALSE
)

## S3 method for class 'BDataDensities'
plot(
  x,
  ...,
  support = getSupport(x),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  nEval = 1500,
  add = FALSE,
```

```
  xlim = getSupport1DPlottingLimits(support),
  ylim = NULL,
  xs = xL[!xL %in% X],
  xL = if (xlim[1] != xlim[2]) seq(xlim[1], xlim[2], length.out = nEval),
  X = getSupportAtoms(support),
  xlab = "x",
  ylab = expression(d * X/d * P),
  close = FALSE
)

## S3 method for class 'BDataLogDensities'
plot(
  x,
  ...,
  support = getSupport(x),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  nEval = 1500,
  add = FALSE,
  xlim = getSupport1DPlottingLimits(support),
  ylim = NULL,
  xs = xL[!xL %in% X],
  xL = if (xlim[1] != xlim[2]) seq(xlim[1], xlim[2], length.out = nEval),
  X = getSupportAtoms(support),
  xlab = "x",
  ylab = expression(ln(dX/d * P)),
  close = FALSE
)

## S3 method for class 'BDataPDFs'
plot(
  x,
  ...,
  support = getSupport(x),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  nEval = 1500,
  add = FALSE,
  xlim = getSupport1DPlottingLimits(support),
  ylim = NULL,
  xs = xL[!xL %in% X],
  xL = if (xlim[1] != xlim[2]) seq(xlim[1], xlim[2], length.out = nEval),
  X = getSupportAtoms(support),
  xlab = "x",
```

```
  ylab = expression(pdf(x) == d * X/d * lambda),
  close = FALSE
)

## S3 method for class 'BDataLogPDFs'
plot(
  x,
  ...,
  support = getSupport(x),
  col = par("col"),
  pch = par("pch"),
  lty = par("lty"),
  lwd = par("lwd"),
  nEval = 1500,
  add = FALSE,
  xlim = getSupport1DPlottingLimits(support),
  ylim = NULL,
  xs = xL[!xL %in% X],
  xL = if (xlim[1] != xlim[2]) seq(xlim[1], xlim[2], length.out = nEval),
  X = getSupportAtoms(support),
  xlab = "x",
  ylab = "f(x)",
  close = FALSE
)
```

Arguments

x	a list of functions to be plotted, naming to be compatible with plot. Alternatively it can be a single function or a list with a single function which gives a matrix of results, where the columns correspond to the arguments and the rows to the different results.
...	further arguments for plot
support	a Support object telling what to plot
col	a (vector of) plotting colors recycled over functions
pch	a (vector of) plotting chars recycled over functions used for the discrete support
lty	a (vector of) line types recycled over the functions used for the lines of the continuous support
lwd	a (vector of) line widths recycled over the function used for the lines of the continuous support
nEval	the number of discretization points for the continuous support
add	a boolean, if TRUE functions are added to an existing plot, if FALSE a new plot is generated.
xlim	the xlim of a new plot
ylim	the ylim of a new plot, if NULL it will be computed from the range of the functions.
xs	the discretization points for the continuous support

xL	candidates for the discretization points for the continuous part, unallowed points will be removed.
X	the points for the discrete part of the support
xlab	the label of the x-axis
ylab	the label of the y-axis
close	if TRUE the data is plotted in closed from (i.e. representing a measure of integral 1), if a numeric value, it is closed to this numeric value. (not yet implemented: if 0 the centered version is plotted), if FALSE, the functions are plotted as is. see [clo].

Value

invisible the results of the computation

- x - the evaluation points for continuous evaluation
- y - the evaluated values for the continuous evaluation, each function gets a column
- X - the evaluation points for the discrete support
- Y - the evaluated values for the discrete support, each function gets a column

Functions

- `plot(BDataDensities)`: plot method for BDataDensities
- `plot(BDataLogDensities)`: plot method for BDataLogDensities
- `plot(BDataPDFs)`: plot method for BDataPDFs
- `plot(BDataLogPDFs)`: plot method for BDataLogPDFs

See Also

Other Support: `Support()`, `cdf2Support()`, `pdf2Support()`, `sample2Support()`

Examples

```
plotFuncs(list(fx(sin(10*x)),fx(cos(10*x))),support=Support(0,pi,0:3),pch=2:3,col=2:3)
X = BDataPDFs(list(fx(dnorm(x,0,1)),fx(dnorm(x,0.3,0.5)),fx(dnorm(x,0,1.5))),
              basis=B2N01Basis5)
plot(X,col=2:4)
plot(as.BDataLogPDFs(X),col=2:4)
plot(as.BDataDensities(X),col=2:4)
plot(as.BDataLogDensities(X),col=2:4)
```

`predict.princomp.BData`

Predict Scores for New Distributional Data

Description

Projects new distributional data onto the principal component space defined by a fitted PCA model. This allows new distributions to be positioned relative to the original dataset without re-running the entire PCA.

Usage

```
## S3 method for class 'princomp.BData'
predict(object, newdata, ...)
```

Arguments

`object`	A ‘princomp.BData‘ object containing the fitted PCA model.
`newdata`	A BData object containing the new distributions to project. Can be any BData subclass; it will be converted to splines using the same basis as the original PCA.
`...`	Additional arguments (currently unused).

Details

The prediction process:

1. Converts ‘newdata‘ to spline coefficients using the same basis as the original PCA.
2. **For non-orthonormal bases:** Applies the same Gram matrix transformation that was used during fitting (see `princomp.BData` details).
3. Centers the (transformed) coefficients using the center from the original PCA.
4. Projects the centered coefficients onto the loading vectors to obtain scores.

The resulting scores can be compared directly to the scores from the original analysis. New observations with scores similar to original observations have similar distribution shapes.

Note: For non-orthonormal bases, the function automatically applies the same transformation that was used during fitting. This ensures consistency between training and prediction.

Value

A matrix of principal component scores with observations in rows and components in columns. Each row contains the coordinates of one new distribution in the PC space.

Author(s)

Matthias Templ

See Also

princomp.BData for fitting the PCA model.

plot.princomp.BData with ‘type="scores"‘ for visualizing scores.

Other PCA: biplot.princomp.BData(), plot.princomp.BData(), plot.princomp.BData.grouped(), plot.princomp.BData.multivariate(), princomp.BData(), screeplot.princomp.BData(), summary.princomp.BData()

Examples

```
## Fit PCA on training data
meanDist <- BDataPDFs(list(dunif), basis = B2Unif01Basis7)
gaussProc <- B2GaussianDistribution(meanDist, function(x,y) exp(-4*abs(x-y)^1.7))
trainData <- as.BDataSplines(simulate(gaussProc, 30, seed = 42))
pcaModel <- princomp(trainData)

## Generate new data and predict scores
newData <- as.BDataSplines(simulate(gaussProc, 5, seed = 999))
newScores <- predict(pcaModel, newData)
print(newScores)

## Compare: new observations vs training data
trainScores <- pcaModel$scores
plot(trainScores[,1], trainScores[,2], pch = 19, col = "gray50",
     xlab = "PC1", ylab = "PC2", main = "Training (gray) vs New (red)")
points(newScores[,1], newScores[,2], pch = 17, col = "red", cex = 1.5)
legend("topright", c("Training", "New"), pch = c(19, 17),
       col = c("gray50", "red"))
```

---

princomp.BData *Principal Component Analysis for Distributional Data*

Description

Performs Simplicial Functional Principal Component Analysis (SFPCA) on distributional data where each observation is an entire probability distribution. This function extends the classical ‘princomp‘ to work with BData objects, enabling dimensionality reduction and exploratory analysis of datasets of distributions.

Usage

```
## S3 method for class 'BData'
princomp(
  x,
  cor = FALSE,
  scores = TRUE,
  covmat = NULL,
```

```
  center = NULL,
  group = NULL,
  multivariate = FALSE,
  ...
)

## S3 method for class 'princomp.BData'
print(x, ...)
```

Arguments

x	A princomp.BData object to print.
cor	Logical. If ‘TRUE‘, the analysis uses the correlation matrix instead of the covariance matrix. This standardizes each coefficient to unit variance before analysis, which can be useful when coefficients have very different scales. Default is ‘FALSE‘.
scores	Logical. If ‘TRUE‘ (default), the principal component scores are computed for each observation. Scores represent the coordinates of each distribution in the reduced PC space.
covmat	An optional pre-computed covariance matrix. If provided, this matrix is used instead of computing the covariance from ‘x‘. Useful for robust or regularized covariance estimates. **Note:** When provided with a non-orthonormal basis, this should be the covariance in the transformed (orthonormalized) space.
center	An optional centering vector. If ‘NULL‘ (default), the column means of the coefficient matrix are used. Providing a custom center allows for centering around a specific reference distribution.
group	An optional factor or vector specifying group membership for each observation. If provided, PCA is performed separately for each group level (unless ‘multivariate = TRUE‘), and the result is an object of class ‘"princomp.BData.grouped"‘ containing a list of PCA results. This is useful for comparing principal components across different subgroups (e.g., males vs females, treatment vs control).
multivariate	Logical. Only relevant when ‘group‘ is provided. If ‘FALSE‘ (default), PCA is performed separately for each group. If ‘TRUE‘, performs **multivariate SFPCA** (2D-SFPCA per Hron et al. 2016) where clr-representations are stacked across groups and a single joint PCA is performed. This captures the covariation between groups and yields common eigenfunctions spanning all groups. Requires that observations are **paired** across groups (same number of observations per group, same observation index = same subject).
...	Additional arguments (currently unused).

Details

Conceptual Background

In distributional data analysis, each observation is not a single number but an entire probability distribution. Examples include:

- Particle size distributions from different samples

- Income distributions across different regions
- Age distributions of populations over time
- Spectral distributions in chemistry or physics

PCA for such data aims to find the main "modes of variation" - the principal ways in which distributions in the dataset differ from each other.

Method

The function works by:

1. Converting all input distributions to a common spline basis representation using `as.BDataSplines`. This yields a matrix where each row is an observation and each column is a spline basis coefficient.
2. **Handling non-orthonormal basis:** If the basis has a non-identity Gram matrix S, the coefficients are transformed via $C' = C \cdot S^{1/2}$ to orthonormalize the representation before PCA.
3. Computing the covariance (or correlation) matrix of these (transformed) coefficients.
4. Performing eigendecomposition to find the principal directions.
5. Projecting the centered data onto these directions to obtain scores.
6. **Back-transforming loadings:** If transformation was applied, loadings are back-transformed via $L = S^{-1/2} \cdot L'$ so they remain in the original coefficient space.

Non-Orthonormal Basis Handling

Important: Many spline bases, including ZB-splines (zero-integral B-splines), are **not orthonormal**. For such bases, standard PCA on raw coefficients gives incorrect results because the Euclidean inner product in coefficient space does not correspond to the L2 inner product in function space.

This function automatically detects non-orthonormal bases by checking the Gram matrix S stored in the `BayesSpaceBasis`. If $S \neq I$, the function applies the correction described in Hron et al. (2016, Eq. 6):

$$N^{-1}C'CMb = \rho b$$

where M is the Gram matrix. This is implemented via Cholesky transformation:

1. Compute $S^{1/2}$ via Cholesky decomposition: $S = LL^T$
2. Transform coefficients: $\tilde{C} = C \cdot L$
3. Perform standard PCA on $\tilde{C}$
4. Back-transform loadings: $V = L^{-1} \cdot \tilde{V}$

The scores remain unchanged since they are computed with the orthonormalized representation. The loadings are transformed back to the original coefficient space for interpretability.

For users: If you want to use an orthonormal basis, you can either:

- Use an explicitly orthonormal spline basis
- Provide the correct Gram matrix S when constructing the `BayesSpaceBasis` (the function will handle the correction)

The Gram matrix S defines the inner product in coefficient space: $\langle c_1, c_2 \rangle_S = c_1^T S c_2$. If S = I (identity), the basis is orthonormal and no correction is needed.

This approach is analogous to `princomp.acomp` from the compositions package, which performs PCA on compositional data using log-ratio coordinates.

Interpretation

- **Eigenvalues/sdev**: The variance explained by each component. Large values indicate important modes of variation.
- **Eigenfunctions**: Directions in coefficient space representing functional perturbations. When added to/subtracted from the mean, they show how distributions change along each PC.
- **Scores**: Coordinates of each observation in the PC space. Similar scores indicate similar distribution shapes.

Use `plot.princomp.BData` with ‘type="eigenfunctions"‘ to visualize how the distribution shape changes along each principal component direction.

Important Considerations

- The number of meaningful components is limited by both the number of observations and the basis dimension.
- If your data shows that PC1 explains nearly 100% of variance, the distributions may vary primarily in one characteristic (e.g., only in scale or only in location). This is not a bug but a property of your data.
- For meaningful PCA, ensure your dataset has genuine multivariate variation. Use `B2GaussianDistribution` with an appropriate covariance structure for simulation studies.

Value

An object of class ‘c("princomp.BData", "princomp")‘ containing:

sdev Numeric vector of standard deviations of the principal components, i.e., the square roots of the eigenvalues of the covariance matrix. These quantify the amount of variation captured by each component.

loadings Matrix of eigenfunctions in coefficient form (class ‘"loadings"‘). Each column is an eigenvector representing a principal component direction in the space of spline coefficients. Rows correspond to basis coefficients. **Note:** For non-orthonormal bases, these are back-transformed to the original coefficient space. Kept for compatibility with ‘stats::princomp‘.

eigenfunctions Alias for ‘loadings‘. This is the proper SFPCA terminology (Hron et al. 2016).

center Numeric vector containing the mean of each spline coefficient, used for centering the data before PCA. This represents the "average distribution" in coefficient space.

scale Numeric vector of scalings applied to each coefficient. All ones unless ‘cor=TRUE‘, in which case these are the standard deviations used for standardization.

n.obs Integer giving the number of observations (distributions) in the dataset.

scores Matrix of principal component scores (if ‘scores=TRUE‘). Each row corresponds to an observation, each column to a principal component. The score indicates how far each distribution deviates from the mean along that PC direction.

call The matched function call.

Center The mean distribution as a BDataSplines object. This can be plotted or converted to other representations (PDF, CDF, etc.).

Loadings The eigenfunctions as a BDataSplines object. Each row represents one principal component as a functional perturbation. Kept for compatibility.

Eigenfunctions Alias for ‘Loadings‘. This is the proper SFPCA terminology.

basis The BayesSpaceBasis used for the spline representation.

OrigClass Character string recording the original class of the input data, useful for back-conversion.

orthonormal Logical indicating whether the basis was orthonormal (‘TRUE‘) or required Gram matrix correction (‘FALSE‘).

GramMatrix The Gram matrix S from the basis. Identity matrix if the basis is orthonormal.

Methods (by generic)

- `print(princomp.BData)`: Print method displaying key PCA results

Author(s)

Matthias Templ

References

Hron, K., Menafoglio, A., Templ, M., Hruzova, K., & Filzmoser, P. (2016). Simplicial principal component analysis for density functions in Bayes spaces. *Computational Statistics & Data Analysis*, 94, 330-350.

See Also

`plot.princomp.BData` for visualization options including screeplots, score plots, biplots, and functional loadings plots.

`biplot.princomp.BData` for creating biplots showing both observations and loadings.

`predict.princomp.BData` for projecting new distributions onto the principal components.

`summary.princomp.BData` for a summary of variance explained.

`as.BDataSplines` for the conversion to spline representation.

`princomp.acomp` for the analogous method for compositional data in the compositions package.

Other PCA: `biplot.princomp.BData()`, `plot.princomp.BData()`, `plot.princomp.BData.grouped()`, `plot.princomp.BData.multivariate()`, `predict.princomp.BData()`, `screeplot.princomp.BData()`, `summary.princomp.BData()`

Examples

```
## Example 1: Basic PCA on simulated distributional data
## -----------------------------------------------------
## We use B2GaussianDistribution to generate distributions with
## genuine multivariate variation (not just scaling differences).
```

```
# Define a Gaussian process on Bayes space with exponential covariance
meanDist <- BDataPDFs(list(dunif), basis = B2Unif01Basis7)
covFunction <- function(x, y) exp(-4 * abs(x - y)^1.7)
gaussProc <- B2GaussianDistribution(meanDist, covFunction)

# Simulate 30 distributions from this process
simData <- simulate(gaussProc, nsim = 30, seed = 42)
splineData <- as.BDataSplines(simData)

# Perform PCA
pcaResult <- princomp(splineData)

# View results
pcaResult
summary(pcaResult)

## Example 2: Visualizing and interpreting the PCA results
## -------------------------------------------------------

# Screeplot: How much variance does each PC explain?
plot(pcaResult, type = "screeplot")
# Interpretation: The first two components (PC1 and PC2) capture most of the
# variance (around 70%), with a clear "elbow" after PC3. This suggests that
# the main patterns in the distributional data can be described by 2-3 modes
# of variation.

# Score plot: How do observations cluster in PC space?
plot(pcaResult, type = "scores")
# Interpretation: Each point represents one distribution. Distributions that
# are close together in this plot have similar shapes. Look for clusters
# (groups of similar distributions), outliers (isolated points), or gradients
# (continuous variation along a direction).

# To identify individual distributions, use labels:
plot(pcaResult, type = "scores", labels = TRUE)
# Now each distribution is labeled with its observation number, making it
# easy to trace outliers or interesting cases back to the original data.

# Loadings plot: How do distributions change along each PC?
# This is THE KEY PLOT for interpretation!
plot(pcaResult, type = "loadings", npcs = 2)
# Interpretation: The black line shows the mean distribution. For each PC,
# the solid colored line shows what a distribution looks like when it has a
# HIGH score on that PC (positive direction), while the dashed line shows a
# LOW score (negative direction). For example, if PC1+ is shifted right and
# PC1- is shifted left compared to the mean, then PC1 captures location
# variation. If PC1+ is wider and PC1- is narrower, then PC1 captures spread.

# Biplot: Combine scores and loadings in one plot
biplot(pcaResult)
# Interpretation: Numbers show observations, red arrows show how spline
# coefficients contribute to the PCs. Observations in similar positions have
# similar distributions. The arrows are harder to interpret directly since
```

```
# they represent abstract basis coefficients - use the loadings plot above
# for functional interpretation.

## Example 3: Predicting scores for new data
## ------------------------------------------
# Generate new distributions from the same process
newData <- as.BDataSplines(simulate(gaussProc, nsim = 5, seed = 123))

# Project onto the existing PC space
newScores <- predict(pcaResult, newData)
print(newScores)

## Example 4: PCA on real age distribution data
## ---------------------------------------------
# Load age distribution data from Upper Austrian communes
data("age")

# Create a basis for age distributions (support 0-100 years)
rmAge <- ReferenceMeasureUniform(0, 100)
B2Age <- BayesSpace(rmAge)
basisAge <- BayesSpaceBasis(B2Age, nbasis = 7)

# Convert proportions to BDataDensityHistograms
# The data are proportions in 5-year age bins (last bin is 10 years: 90-100)
breaks <- c(0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 100)
ageHist <- BDataDensityHistograms(age$distributions, breaks = breaks, basis = basisAge)

# Convert: histograms -> densities -> splines
ageDensities <- as.BDataDensities(ageHist)
ageSplines <- as.BDataSplines(ageDensities)

# Perform PCA
pcaAge <- princomp(ageSplines)
summary(pcaAge)

# Screeplot: Most variance is in PC1 and PC2
plot(pcaAge, type = "screeplot")

# Score plot colored by sex
sexColors <- ifelse(age$data$sex == "male", "blue", "red")
plot(pcaAge, type = "scores", col = sexColors)
legend("topright", c("Male", "Female"), col = c("blue", "red"), pch = 19)
# Interpretation: Males and females separate along PC1, indicating that the
# primary mode of variation captures sex differences in age structure
# (e.g., females tend to live longer, creating different elderly proportions).

# Loadings plot: What do the PCs represent?
plot(pcaAge, type = "loadings", npcs = 2)
# Interpretation: PC1 captures the balance between younger and older ages.
# PC2 may capture differences in the working-age population bulge.

## Example 5: Grouped PCA - compare males vs females
## -------------------------------------------------
```

```
# Perform PCA separately for each sex group
pcaGrouped <- princomp(ageSplines, group = age$data$sex)
pcaGrouped

# Compare screeplots: Do males and females have similar dimensionality?
plot(pcaGrouped, type = "screeplot")

# Compare loadings: Are the modes of variation the same?
plot(pcaGrouped, type = "loadings", npcs = 2)
# Interpretation: If the loadings look similar across groups, the same
# types of variation exist in both. If they differ, each group has
# distinct patterns of age distribution variation.

# Compare score distributions with boxplots
plot(pcaGrouped, type = "boxplot", npcs = 3)

## Example 6: Multivariate SFPCA (2D-SFPCA) - joint analysis across groups
## ---------------------------------------------------------------------------
# Unlike grouped PCA which analyzes groups separately, multivariate SFPCA
# stacks the representations and finds COMMON principal directions across groups.
# This requires paired data: same observation index = same subject across groups.

pcaMulti <- princomp(ageSplines, group = age$data$sex, multivariate = TRUE)
pcaMulti

# Screeplot: Total variance across both male and female distributions
plot(pcaMulti, type = "screeplot")

# Score plot: Each point is a commune, scored on joint PCs
plot(pcaMulti, type = "scores")

# Eigenfunctions: Shows how each group component of the joint eigenfunction
# perturbs the mean. Left column = male, right column = female.
plot(pcaMulti, type = "eigenfunctions", npcs = 2)
# Interpretation: If PC1 shows similar patterns for both groups, they vary
# together. If patterns differ, the PC captures group-specific variation
# or interactions between groups.
```

`print.BayesSpace` *Pretty printing of BayesSpaces*

Description

Pretty printing of BayesSpaces

Usage

```
## S3 method for class 'BayesSpace'
print(x, ...)
```

Arguments

x	the Bayes space
...	further unused parameters for generic use

Value

‘x‘, invisibly (called for the printed side effect).

Examples

```
print(B2Unif01)
```

---

print.BDataLogDensities

Print a BDataLogDensities object

Description

Print a BDataLogDensities object

Usage

```
## S3 method for class 'BDataLogDensities'
print(x, ...)
```

Arguments

x	a [BDataLogDensities] object
...	passed to the next print method.

Value

‘x‘, invisibly.

print.BDataLogPDFs *Print a BDataLogPDFs object*

Description

Print a BDataLogPDFs object

Usage

```
## S3 method for class 'BDataLogPDFs'
print(x, ...)
```

Arguments

x	a [BDataLogPDFs] object
...	passed to the next print method.

Value

‘x‘, invisibly.

print.BDataSplines *Print a BDataSplines object*

Description

Print a BDataSplines object

Usage

```
## S3 method for class 'BDataSplines'
print(x, ...)
```

Arguments

x	a [BDataSplines] object
...	further arguments, currently unused.

Value

‘x‘, invisibly (called for the printed side effect).

Examples

```
X <- BDataSplines(t(1:5), B2Unif01Basis5)
print(X)
```

print.L2Space — *prints "L2Space with reference measure" and than the reference measure.*

Description

prints "L2Space with reference measure" and than the reference measure.

Usage

```
## S3 method for class 'L2Space'
print(x, ...)
```

Arguments

x	the L2 space
...	further unused parameters for generic use

Value

‘x‘, invisibly (called for the printed side effect).

Examples

```
print(getL2Space(B2Unif01))
```

print.princomp.BData.grouped — *Print a grouped distributional PCA*

Description

Print a grouped distributional PCA

Usage

```
## S3 method for class 'princomp.BData.grouped'
print(x, ...)
```

Arguments

x	a ‘princomp.BData.grouped‘ object (from [princomp.BData] with a ‘group‘).
...	further arguments, currently unused.

Value

‘x‘, invisibly. Prints, per group, the number of observations and the share of variance explained by the first principal component.

See Also

[princomp.BData]

Examples

```
data("age")
basisAge <- BayesSpaceBasis(BayesSpace(ReferenceMeasureUniform(0, 100)), nbasis = 7)
breaks <- c(0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 100)
ageHist <- BDataDensityHistograms(age$distributions, breaks = breaks, basis = basisAge)
ageSplines <- as.BDataSplines(as.BDataDensities(ageHist))
pcaGrouped <- princomp(ageSplines, group = age$data$sex)
print(pcaGrouped)
```

---

print.princomp.BData.multivariate

Print a multivariate (2D-)SFPCA on distributional data

Description

Print a multivariate (2D-)SFPCA on distributional data

Usage

```
## S3 method for class 'princomp.BData.multivariate'
print(x, ...)
```

Arguments

x	a ‘princomp.BData.multivariate‘ object (from [princomp.BData] with ‘multivariate = TRUE‘).
...	further arguments, currently unused.

Value

‘x‘, invisibly. Prints the groups, observations and dimensions of the joint analysis together with the variance explained.

See Also

[princomp.BData]

Examples

```
data("age")
basisAge <- BayesSpaceBasis(BayesSpace(ReferenceMeasureUniform(0, 100)), nbasis = 7)
breaks <- c(0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 100)
ageHist <- BDataDensityHistograms(age$distributions, breaks = breaks, basis = basisAge)
ageSplines <- as.BDataSplines(as.BDataDensities(ageHist))
pcaMulti <- princomp(ageSplines, group = age$data$sex, multivariate = TRUE)
print(pcaMulti)
```

---

print.ReferenceMeasure

Printing ReferenceMeasures

Description

The measure is printed by the call that created it.

Usage

```
## S3 method for class 'ReferenceMeasure'
print(x, ...)
```

Arguments

x	the reference measure to be printed
...	further generic arguments, which will be ignored.

Value

an invisible version of the reference measure

Examples

```
print(StdReferenceMeasureUnif01)
print(StdReferenceMeasureN01)
```

ReferenceMeasure *Reference measures: the geometry of a Bayes space*

Description

A reference measure is the probability distribution that defines a [BayesSpace]: it plays the role of the origin, and all modelled distributions are expressed relative to it. Choosing it fixes the support and the geometry of the analysis.

Details

Choosing a reference measure Pick a measure whose support matches your data, and ideally whose mass is distributed similarly:

[ReferenceMeasureUniform] uniform on an interval ‘(a, b)‘; the simplest choice for data on a bounded interval (predefined: ‘B2Unif01‘).

[ReferenceMeasureTransformed] a transformed support, e.g. a Gaussian reference via its CDF (predefined: ‘B2N01‘); useful for data on the real line or with a known reference shape.

[ReferenceMeasureDiscrete] a finite set of categories with given probabilities.

A poor choice (e.g. a uniform reference for strongly skewed data) still works but gives a less informative geometry. Inspect a measure with [getReferenceMeasure] and the resulting metric with [getMetricTensor].

See Also

[BayesSpace], [BayesSpaceBasis], [BData]

ReferenceMeasureCartesianProduct *Compute a cartesian Product of measures*

Description

Compute a cartesian Product of measures

Usage

```
ReferenceMeasureCartesianProduct(..., nSamples = 1e+05)
```

Arguments

`...`	the measures, possibly named
`nSamples`	the (maximum) number of samples to be used to represent the measure in the fixed sample for Monte Carlo integrations.

Value

A ReferenceMeasureCartesianProduct, which is essentially a list of the measures, in an object of that class, that acts like a ReferenceMeasure for the Cartesian Product of the supports of the individual measures.

The product measure corresponds gives a any product set the measure of the that corresponds to the product of the individual measures. In case of a probablity distribution, that would be the measure representing an independent distribution, with the initial measures as marginal distributions. This concept is used to establish a reference measure for multivariate BayesSpaces which can also contain non independent measures.

See Also

Other Cartesian: BayesSpaceBasis.BayesSpaceCartesianProduct(), BayesSpaceBasisCartesianProduct(), BayesSpaceCartesianProduct(), SupportCartesianProduct(), cartesianProduct(), getBayesSpace.ReferenceMea getCartesianFactors(), subBasisIndices()

Examples

```
RMCP3 = ReferenceMeasureCartesianProduct(A=StdReferenceMeasureN01,
                                         B=StdReferenceMeasureUnif01,
                                C=ReferenceMeasureDiscrete(c("A","B","C"),c(0.2,0.3,0.5))
                                         )
```

ReferenceMeasureDiscrete

Discrete Reference Measure for BayesSpaces

Description

A discrete reference measure is a measure on a discrete support.

Usage

```
ReferenceMeasureDiscrete(
  X,
  p = rep(1/length(X), length(X)),
  sampler = NULL,
  support = Support(X = X)
)
```

Arguments

X	either a complete vector or list of all possible values or a subset that can be used as a sample. In this case the support shall be given.
p	is either vector of probability values, or a function mapping a value to its probability. Currently only uniform distributions are supported

sampler	a function of a number of random realizations, that samples ffrom this measure.
support	the support to be used for the ReferenceMeasure

Value

An object of type ReferenceMeasureDiscrete encapsulating this measure

See Also

Other ReferenceMeasures: B2N01, ReferenceMeasureTransformed(), ReferenceMeasureUniform(), StdReferenceMeasureUnif01, expectation(), getSample(), isInDomain()

Examples

```
X <- c("A","B","C")
( rmd <- ReferenceMeasureDiscrete(X) )
```

ReferenceMeasureTransformed

A general continuous reference measure on the real line

Description

The reference measure is created by transforming a ReferenceMeasure Uniform to a given distributions based using a distribution function and a quantile function.

Usage

```
ReferenceMeasureTransformed(
  cdf = c("pnorm", "pbeta", "pcauchy", "pchisq", "pexp", "pf", "pgamma", "plnorm",
     "plogis", "pt", "punif", "pweibull", "pwilcox"),
   qf = cdf2qf(cdf, ...),
   ...,
   pdf = cdf2density(cdf, ...),
   reference = StdReferenceMeasureUnif01,
   .args = list(...),
   .eps = 1e-07
)

## S3 method for class 'ReferenceMeasureTransformed'
getSample(x, ...)
```

Arguments

cdf	the cumulative distribution function of the reference measure. For the standard distributions listed as default argument, it can be provided as a string. In this case the corresponding quantile and density functions are selected automatically. The distribution parameters should be provided in ...
qf	the corresponding quantile function, mapping a value of the reference measure to its quantiles.
...	further arguments for cdf, qf, and lebesgue_density
pdf	a function representing the Lebesgue Density (aka as probability density function) of the measure, for conversion to and from pdfs.
reference	the underlying untransformed reference measure to be used
.args	the list of further arguments to cdf and qf
.eps	the precision to check the roundtrip between
x	a ReferenceMeasureTransformed object

Value

a reference measure with representing the reference measure described by the distribution and quantile function.

Methods (by generic)

- `getSample(ReferenceMeasureTransformed)`: returns transformed sample from the underlying reference

See Also

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureUniform()`, `StdReferenceMeasureUnif01`, `expectation()`, `getSample()`, `isInDomain()`

Examples

```
rmt = ReferenceMeasureTransformed(pnorm,qnorm,mean=2,sd=2)
isInDomain(0:17,rmt)
expectation(rmt,\(x) x^2)
hist(getSample(rmt))
```

ReferenceMeasureUniform

create the ReferenceMeasureUniform class

Description

create the ReferenceMeasureUniform class

Usage

```
ReferenceMeasureUniform(
  a = 0,
  b = 1,
  sample = makeUnifGrid(2^12, a, b),
  seed = NULL
)
```

Arguments

a	the minimum of the domain interval
b	the maximum of the domain interval
sample	the sample to be used for internal computations (use with care)
seed	optional integer. If given, the Monte Carlo sample is drawn under this seed and the state of the random number generator is restored afterwards (the same convention as `simulate()`), so that the constructed object is reproducible independently of the global RNG state and of any code executed before or after the call. The default sample however is currently deterministic.

Value

a 'ReferenceMeasureUniform class'

See Also

[BData]

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `StdReferenceMeasureUnif01`, `expectation()`, `getSample()`, `isInDomain()`

Examples

```
min <- -5
max <- 5
RMU <- ReferenceMeasureUniform(min,max)
RMU
## reproducible construction, RNG state left untouched:
RMU1 <- ReferenceMeasureUniform(0, 1, seed = 20260702)
RMU2 <- ReferenceMeasureUniform(0, 1, seed = 20260702)
identical(RMU1$sample, RMU2$sample)
```

`sample2Support` *Guesses the support from a sample*

Description

A support is a range plus possible points with positive probability. The function detects them by bindings.

Usage

```
sample2Support(X, digits = 1, minBindings = max(length(X)/50, 3))
```

Arguments

`X`	a numerical vector with sample data
`digits`	the number of digits infered boundaries should be rounded to.
`minBindings`	the minimum number of equal values required to consider them as a value with positive measure.

Value

a Support object

See Also

Other Support: `Support()`, `cdf2Support()`, `pdf2Support()`, `plotFuncs()`

Examples

```
sample2Support(rnorm(400))
sample2Support(runif(1000))
sample2Support(rlnorm(400))
sample2Support(rpois(400,2))
sample2Support(c(rlnorm(200),rep(0,30)))
```

`scalarProduct` *Computes the scalar product of (datasets of) vectors from various spaces*

Description

The scalar product might depend on the space in which the operation takes place.

Usage

```
scalarProduct(x, y, space = NULL, ...)

## S3 method for class 'numeric'
scalarProduct(x, y = x, space, ...)

## S3 method for class 'BDataSplines'
scalarProduct(x, y, space = getBayesSpace(x), ...)

## S3 method for class 'BDataDensities'
scalarProduct(x, y = x, space = getBayesSpace(x), ...)

## S3 method for class 'BDataLogDensities'
scalarProduct(x, y = x, space = getBayesSpace(x), ...)
```

Arguments

`x`	an element of the space (or a dataset of such elements)
`y`	an element of the space (or a dataset of such elements)
`space`	the space in which the scalar product shall be computed
`...`	further generic arguments, like e.g. method for the compution

Value

A numeric scalar, vector, or matrix of inner products, depending on the inputs.

Methods (by class)

- `scalarProduct(numeric)`: of vectors (or rows of a matrix) potentially over the metric tensor of the space.
- `scalarProduct(BDataSplines)`: in BayesSpaces for BDataSplines objects
- `scalarProduct(BDataDensities)`: in BayesSpaces for BDataDensities objects
- `scalarProduct(BDataLogDensities)`: in BayesSpaces for BDataLogDensities objects

Examples

```
X <- BDataSplines(matrix(c(1,2,3,4,5, 2,1,0,1,2), nrow=2, byrow=TRUE),
                  B2Unif01Basis5)
scalarProduct(X, X)
```

`scalarProductOfLogDensities`
Computes the scalar product of log densities in a BayesSpace

Description

This uses the Covariance Operator capability of the ReferenceMeasure, x and y are converted to functions.

Usage

```
scalarProductOfLogDensities(x, y = x, space)
```

Arguments

x	a log density function (centering is not required)
y	a log density function (centering is not required)
space	a BayesSpace (or anything that has a ReferenceMeasure)

Value

the scalar product of the (centred) log-densities ‘x‘ and ‘y‘ with respect to the reference measure of ‘space‘, computed via the covariance operator of that measure. A single numeric value for scalar functions, or a matrix of pairwise scalar products when ‘x‘ and ‘y‘ are vector-valued (e.g. basis functions).

Examples

```
scalarProductOfLogDensities(function(x) rep(1,length(x)),space=B2Unif01)
```

scale.BData — *Scale and/or center a BData set*

Description

Apply scaling and/or centering to a dataset of BData objects. Diverts all calculations to ‘mean.BData‘-instances as well as to ‘msd.BData()‘ and the arithmetic operations.

Usage

```
## S3 method for class 'BData'
scale(x, center = TRUE, scale = TRUE, ...)
```

Arguments

x	data set to transform
center	either logical (default: TRUE) or the mean BData object to subtract from the elements of ‘x‘
scale	either logical (default: TRUE) or the scalar to divide the (centered?) elements of ‘x‘ with
...	further generic arguments

Value

a data set of the same kind and size of ‘x‘

Examples

```
bdl = BDataDensities(list(\(x)(dnorm(x,1,1)),\(x)(dnorm(x,-1,1)) ),basis=B2N01Basis5,closed=TRUE)
bdlsc = scale(bdl)
msd(bdlsc) # should now be 1
mean(bdlsc)[[1]](-5:5) # should be 1 everywhere, i.e. the neutral element
```

screeplot.princomp.BData — *Screeplot for PCA of Distributional Data*

Description

Creates a screeplot showing the variance (or eigenvalue) associated with each principal component. This is a standard tool for deciding how many principal components to retain based on the "elbow" criterion.

Usage

```
## S3 method for class 'princomp.BData'
screeplot(
  x,
  npcs = min(10, length(x$sdev)),
  type = c("barplot", "lines"),
  main = "Screeplot",
  ...
)
```

Arguments

`x`	A ‘princomp.BData‘ object.
`npcs`	Number of principal components to include in the plot. Default is ‘min(10, length(x$sdev))‘.
`type`	Character string specifying the plot type: **‘"barplot"‘** Bar chart of variances (default). **‘"lines"‘** Line plot connecting variance values.
`main`	Plot title. Default is ‘"Screeplot"‘.
`...`	Additional graphical parameters.

Details

The screeplot displays the eigenvalues (variances) of the principal components in decreasing order. The name comes from the plot’s resemblance to a scree slope at the base of a mountain.

How to Use

Look for an "elbow" in the plot where the variance drops off sharply. Components before the elbow are typically considered important and retained, while those after the elbow contribute little to the overall structure.

Alternative criteria for choosing components include:

- Retain components explaining a target percentage of variance (see ‘plot(x, type="variance")‘).
- Kaiser criterion: retain components with variance > 1 (when using correlation matrix, ‘cor=TRUE‘).

Value

Invisibly returns the variance values for each component.

Author(s)

Matthias Templ

See Also

princomp.BData for the PCA analysis.

plot.princomp.BData for additional plot types including cumulative variance plots.

Other PCA: biplot.princomp.BData(), plot.princomp.BData(), plot.princomp.BData.grouped(), plot.princomp.BData.multivariate(), predict.princomp.BData(), princomp.BData(), summary.princomp.BData(

Examples

```
meanDist <- BDataPDFs(list(dunif), basis = B2Unif01Basis7)
gaussProc <- B2GaussianDistribution(meanDist, function(x,y) exp(-4*abs(x-y)^1.7))
pcaResult <- princomp(as.BDataSplines(simulate(gaussProc, 25, seed = 42)))

# Standard barplot screeplot
screeplot(pcaResult)

# Line plot version
screeplot(pcaResult, type = "lines")
```

---

selectIntegrationMethod

Identifies the integration method to be choosen for a measure

---

Description

Identifies the integration method to be choosen for a measure

Usage

```
selectIntegrationMethod(
  measure,
  method = c("default", "sample", "MonteCarlo", "integrate")
)

## S3 method for class 'ReferenceMeasure'
selectIntegrationMethod(measure, method)

## S3 method for class 'CartesianProductMeasure'
selectIntegrationMethod(measure, method)
```

Arguments

measure	the ReferenceMeasure for which the method should be selected
method	a user provide method hint

Value

a string identifiying the method

Methods (by class)

- `selectIntegrationMethod(ReferenceMeasure)`: standard method using "sample"
- `selectIntegrationMethod(CartesianProductMeasure)`: select "MonteCarlo" if the getSample is an exhausted random default sample. Otherwise use "sample"

Examples

```
selectIntegrationMethod(StdReferenceMeasureUnif01,"default")
```

---

simulate.B2DistHyper *Simulates a B2DistHyper distribution*

Description

Simulates a B2DistHyper distribution

Usage

```
## S3 method for class 'B2DistHyper'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  density = object$density,
  basis = object$basis,
  paramSimulator = object$paramSimulator,
  simulator = object$simulator,
  hyperparameters = object$hyperparameters,
  ...
)
```

Arguments

`object`	an B2DistHyper object as returned from B2DistHyper
`nsim`	the number of simulations
`seed`	a seed eventually to be set before the simulation (use with care!)
`density`	A function that can be called with a vectorized x of data values of the BayesSpace and parameters, to calculate a density.
`basis`	A BayesSpaceBasis (or an object that can be converted to one) of BayesSpace containing the values of the distribution (of distributions).
`paramSimulator`	A function to be called without arguments to simulate one set of parameters.
`simulator`	A function that can be called with an integer n and the parameters to create n random simulations.
`hyperparameters`	the list of parametrs to be given to the the parameterSimulator
`...`	futher unused parameters, required to fullfill the generic definition of simulate

Value

A [BDataDensities] following the given distribution.

See Also

[stats::simulate]

Examples

```
D = B2DistHyper(dbeta,B2Unif01Basis5,function() rexp(2),simulator=rbeta)
simulate(D,4,seed=47)
```

---

simulate.BData — *Simulate from the distribution described by the BData objects*

Description

Simulate from the distribution described by the BData objects

Usage

```
## S3 method for class 'BData'
simulate(
  object,
  nsim = 1L,
  seed = NULL,
  ...,
  returnBDataSamples = numberOfCases(object) > 1
)
```

Arguments

`object`	a dataset of distributions from a BayesSpace
`nsim`	number of simulations
`seed`	the seed to be used. If seed is given the resulting object has a seed attribute, with the seed given (use with care).
`...`	futher generic parameters e.g. for the method of simulation used.
`returnBDataSamples`	a boolean, if returnBDataSamples==FALSE 'object' may only contain 1 distribution and the simulation of that distribution is returned directly. If TRUE the simulation is done for each of them and the results is returned as BDataSamples.

Value

(Planned, once implemented:) A BDataSample object with each entry nsim simulations and one simulation for each distribution in the BData object. Currently the function always throws an error.

Note

Not yet implemented. Calling this method currently stops with an error. It is registered so that ‘simulate()‘ on a ‘BData‘ object gives an informative message rather than a method-dispatch failure. The documented parameters and return value describe the planned interface.

---

simulate.ReferenceMeasureCartesianProduct

Simulates from a ReferenceMeasureCartesianProduct

Description

Simulates from a ReferenceMeasureCartesianProduct

Usage

```
## S3 method for class 'ReferenceMeasureCartesianProduct'
simulate(object, nsim = 1L, seed = NULL, ...)
```

Arguments

`object`	the ReferenceMeasureCartesianProduct object
`nsim`	the number of simulations
`seed`	optional seed, if set it is used and reported as attribute in the simulation (use with care).
`...`	further unused generic arguments

Value

a dataframe with nsim rows with simulations of the product measure.

See Also

[stats::simulate]

Examples

```
set.seed(4)
CP = cartesianProduct(StdReferenceMeasureUnif01,StdReferenceMeasureN01)
simulate(CP,2,seed=17)
simulate(CP,2)
simulate(CP,2,seed=17)
simulate(CP,2)
```

simulate.ReferenceMeasureDiscrete

Generates simulations from the Discrete Reference Measure seen as distribution.

Description

The simulation is done by the r function sample

Usage

```
## S3 method for class 'ReferenceMeasureDiscrete'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	the ReferenceMeasureDiscrete object
nsim	the number of simulations
seed	optional seed, if set it is used and reported as attribute in the simulation (use with care).
...	further unused generic arguments

Value

a vector (or list) of length nsim with of random variables with the given distribution

Examples

```
set.seed(4)
aDiscreteMeasure = ReferenceMeasureDiscrete(c(0.2,0.3,0.4),X=c("A","B","C"))
simulate(aDiscreteMeasure,10,seed=2)
simulate(aDiscreteMeasure,10)
simulate(aDiscreteMeasure,10,seed=2)
simulate(aDiscreteMeasure,10)
```

simulate.ReferenceMeasureTransformed

Generates simulations from the Reference Measure seen as distribution.

Description

The simulations are done by applying the transformation to simulations on the underlying reference measure

Usage

```
## S3 method for class 'ReferenceMeasureTransformed'
simulate(object, nsim = 1, seed = NULL, ...)
```

Arguments

object	the ReferenceMeasureUniform object
nsim	the number of simulations
seed	optional seed, if set it is used and reported as attribute in the simulation (use with care).
...	further unused generic arguments

Value

a vector of length nsim with of random variables with the given distribution

Examples

```
set.seed(4)
simulate(StdReferenceMeasureN01,10,seed=2)
simulate(StdReferenceMeasureN01,10)
simulate(StdReferenceMeasureN01,10,seed=2)
simulate(StdReferenceMeasureN01,10)
```

---

simulate.ReferenceMeasureUniform

simulation method for the uniform referencemeasure

Description

simulation method for the uniform referencemeasure

Usage

```
## S3 method for class 'ReferenceMeasureUniform'
simulate(object, nsim = 1L, seed = NULL, ...)
```

Arguments

object	the ReferenceMeasureUniform object
nsim	the number of simulations
seed	optional seed, if set it is used and reported as attribute in the simulation (use with care).
...	further unused generic arguments

Value

a vector of length nsim with of uniformly distributed random variables.

See Also

[stats::simulate]

Examples

```
set.seed(4)
simulate(StdReferenceMeasureUnif01,10,seed=2)
simulate(StdReferenceMeasureUnif01,10)
simulate(StdReferenceMeasureUnif01,10,seed=2)
simulate(StdReferenceMeasureUnif01,10)
```

---

spatialDepth *Spatial Depth in dataset*

Description

The depth in a dataset as defined by 1 the length of the mean of the unit vectors pointing to all (other) datapoints. For a point with depth 0 the direction to all other points is parallel. For a point with depth 1, the point corresponds exactly to the median of the dataset.

Usage

```
spatialDepth(x, data = x, ..., norm)

## S3 method for class 'BData'
spatialDepth(x, data = x, ..., norm = DistributionalDDA::norm)

## S3 method for class 'numeric'
spatialDepth(x, data = x, ..., norm = abs)

## S3 method for class 'matrix'
spatialDepth(x, data = x, ..., norm = function(x) sqrt(rowMeans(x^2)))
```

Arguments

`x`	the values for which the depth shall be computed
`data`	the dataset against which the spatial depth of ‘x‘ is computed (the reference cloud of points). Defaults to ‘x‘ itself.
`...`	further arguments to norm determining computation or type of norm
`norm`	the norm to be used

Value

the spatial depth of the values in x

Methods (by class)

- `spatialDepth(BData)`: method for BData objects
- `spatialDepth(numeric)`: method for numerical data
- `spatialDepth(matrix)`: method for multivariate data

See Also

Other caseStatistics: `caseE()`, `caseLogRatios()`, `caseMeans()`, `caseQuantiles()`, `caseSds()`, `caseVars()`, `centralityQuantile()`, `distanceToCenter()`, `median.BData()`

Examples

```
data(DTD)
Xpdf    = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
 spatialDepth(Xpdf)
```

StdReferenceMeasureUnif01

The standard B2(Unif(0,1)) Bayes space.

Description

B2Unif01 is the Bayes space B2(Unif(0,1)), and B2Unif01BasisN its B-Spline basis with N elements of order 2

Usage

```
StdReferenceMeasureUnif01

StdReferenceMeasureN01

B2Unif01

B2Unif01Basis5

B2Unif01Basis7

B2Unif01Basis20
```

Format

An object of class `ReferenceMeasureUniform` (inherits from `ReferenceMeasure`) of length 3.

An object of class `ReferenceMeasureTransformed` (inherits from `ReferenceMeasure`) of length 5.

An object of class `BayesSpace` of length 1.

An object of class `BayesSpaceBasis` of length 5.

An object of class `BayesSpaceBasis` of length 5.

An object of class `BayesSpaceBasis` of length 5.

Functions

- `StdReferenceMeasureUnif01`: the underlying standard reference measure
- `StdReferenceMeasureN01`: the underlying standard reference measure
- `B2Unif01Basis5`: a basis for B2Unif01 with 5 elements
- `B2Unif01Basis7`: a basis for B2Unif01 with 7 elements
- `B2Unif01Basis20`: a basis for B2Unif01 with 20 elements

See Also

[BayesSpaceBasis]

Other ReferenceMeasures: `B2N01`, `ReferenceMeasureDiscrete()`, `ReferenceMeasureTransformed()`, `ReferenceMeasureUniform()`, `expectation()`, `getSample()`, `isInDomain()`

Examples

```
getReferenceMeasure(B2Unif01)
getMetricTensor(B2Unif01Basis5)
```

`Sub` *Substract functional objects*

Description

This subtraction is also able to add functions and BayesSpace objects. The sum of two functions is a function that results in the difference of the two results. The difference of two BayesSpace objects (in BData), is given by the BayesSpace operations, which is given by the quotient of the densities with respect to the reference measure.

Usage

```
Sub(x, y)

## S3 method for class '`function`'
Sub(x, y)

## S3 method for class 'list'
Sub(x, y)

## S3 method for class 'numeric'
Sub(x, y)

## S3 method for class '`function`'
Sub.function(x, y)

## S3 method for class '`function`'
Sub.list(x, y)

## S3 method for class 'list'
Sub.function(x, y)

## S3 method for class 'numeric'
Sub.function(x, y)

## S3 method for class '`function`'
Sub.numeric(x, y)

## S3 method for class 'numeric'
Sub.numeric(x, y)

## S3 method for class 'numeric'
Sub.list(x, y)

## S3 method for class 'list'
Sub.numeric(x, y)

## S3 method for class 'list'
Sub.list(x, y)

## S3 method for class 'BDataHistograms'
Sub(x, y)

## S3 method for class 'BDataHistograms'
Sub.BDataHistograms(x, y)

## S3 method for class 'BDataDensities'
Sub(x, y)
```

```
## S3 method for class 'BDataSplines'
Sub(x, y)

## S3 method for class 'BDataLogDensities'
Sub(x, y)

## S3 method for class 'BDataPDFs'
Sub(x, y)

## S3 method for class 'BDataLogPDFs'
Sub(x, y)
```

Arguments

x	a functional object
y	a functional object

Value

A BData object of the same subclass as the first argument (the Bayes-space sum / difference / scalar multiple / quotient).

Methods (by class)

- `Sub(`function`)`: method for functions
- `Sub(list)`: method for lists, iterating over the list
- `Sub(numeric)`: method for numerics, treating them as constant functions
- `Sub(BDataHistograms)`: method for BDataHistograms (operating in BayesSpaces)
- `Sub(BDataDensities)`: BayesSpace Difference for BDataDensities
- `Sub(BDataSplines)`: BayesSpace Difference for BDataSplines
- `Sub(BDataLogDensities)`: BayesSpace Difference for BDataLogDensities
- `Sub(BDataPDFs)`: BayesSpace Difference for BDataPDFs
- `Sub(BDataLogPDFs)`: BayesSpaceAddition for BDataLogPDFs

Functions

- `Sub.function(`function`)`: method for two functions, returning a function giving the difference of the two as a result
- `Sub.list(`function`)`: method for a function and list, iterating over the list
- `Sub.function(list)`: method for a function and list, iterating over the list
- `Sub.function(numeric)`: method substracting a constant value from a function
- `Sub.numeric(`function`)`: method substracting a function from a constant
- `Sub.numeric(numeric)`: method substracting two values
- `Sub.list(numeric)`: method for substracting a value from each element of a list, returning a list.

- `Sub.numeric(list)`: method each element of a list from a constant
- `Sub.list(list)`: method for vectorized application of sub on the elements of lists
- `Sub.BDataHistograms(BDataHistograms)`: method for two BDataHistograms

See Also

Other Add: `Add()`, `Div()`, `Mul()`

Examples

```
Sub(sin,cos)
Sub(list(sin,cos),sin)
Sub(cos,list(sin,cos))
Sub(list(sin,cos),list(sin,cos))
Sub(list(sin,cos),10)
```

---

subBasisIndices *Computes the indices of a certain marginal basis*

Description

Computes the indices of a certain marginal basis

Usage

```
subBasisIndices(
  basis,
  indices,
  basisDims = sapply(attr(basis, ".LIST"), function(x) nrow(x$S))
)
```

Arguments

`basis`	the full cartesian BayesSpaceBasis
`indices`	giving which the marginal subspaces of interest, given as booelan vector or vector of names of positive or negative integers .
`basisDims`	the dimensions of the basises for the various subspaces

Value

an integer vector with the indices of the interaction described by the indices.

See Also

Other Cartesian: `BayesSpaceBasis.BayesSpaceCartesianProduct()`, `BayesSpaceBasisCartesianProduct()`, `BayesSpaceCartesianProduct()`, `ReferenceMeasureCartesianProduct()`, `SupportCartesianProduct()`, `cartesianProduct()`, `getBayesSpace.ReferenceMeasureCartesianProduct()`, `getCartesianFactors()`

summary.princomp.BData

Summary of PCA for Distributional Data

Description

Provides a summary of the principal component analysis results, showing the importance of each component in terms of standard deviation, proportion of variance explained, and cumulative proportion.

Usage

```
## S3 method for class 'princomp.BData'
summary(object, ...)
```

Arguments

`object`	A ‘princomp.BData‘ object.
`...`	Additional arguments (currently unused).

Value

An object of class ‘summary.princomp.BData‘ containing:

sdev Standard deviations of the principal components.

importance Matrix with three rows: standard deviation, proportion of variance, and cumulative proportion for each component.

Author(s)

Matthias Templ

See Also

`princomp.BData`

Other PCA: `biplot.princomp.BData()`, `plot.princomp.BData()`, `plot.princomp.BData.grouped()`, `plot.princomp.BData.multivariate()`, `predict.princomp.BData()`, `princomp.BData()`, `screeplot.princomp.BDat`

Examples

```
meanDist <- BDataPDFs(list(dunif), basis = B2Unif01Basis7)
gaussProc <- B2GaussianDistribution(meanDist, function(x,y) exp(-4*abs(x-y)^1.7))
pcaResult <- princomp(as.BDataSplines(simulate(gaussProc, 25, seed = 42)))
summary(pcaResult)
```

Support *The support of the measures*

Description

For this package the support tells what is needed to know for plotting

Usage

```
Support(a, b, X = c(), lim)

getSupport1DPlottingLimits(x)

getSupportAtoms(x)

getSupport(x, ...)

## S3 method for class 'Support'
getSupport(x, ...)

## S3 method for class 'BayesSpace'
getSupport(x, ...)

## S3 method for class 'BayesSpaceBasis'
getSupport(x, ...)

## S3 method for class 'BData'
getSupport(x, ...)

## S3 method for class 'ReferenceMeasureUniform'
getSupport(x, ...)

## S3 method for class 'ReferenceMeasureDiscrete'
getSupport(x, ...)

## S3 method for class 'ReferenceMeasureTransformed'
getSupport(x, ..., alpha = 0.01)

## S3 method for class 'list'
getSupport(x, ...)

## S3 method for class '`function`'
getSupport(
  x,
  ...,
  extrems = c(-1e+20, 1e+20),
  starts = c(exp(seq(-20, 20, by = 0.2)), -exp(seq(-20, 20, by = 0.2)))
```

```
)

## S3 method for class 'numeric'
getSupport(x, ...)

## S3 method for class 'ListData'
getSupport(x, ...)

## S3 method for class 'ReferenceMeasureCartesianProduct'
getSupport(x, ...)

## S3 method for class 'varKernel'
getSupport(x, ...)

## S3 method for class 'variationKernel'
getSupport(x, ...)
```

Arguments

a	lower limit of the continuous support interval (a,b), may be -Inf
b	upper limit of the continuous support interval (a,b), may be Inf, a==b (which should be a value in x) means no continuous support)
X	the discrete values of the support, possibly an empty vector.
lim	the effective xlim to use for plots or continuous evaluations on the support.
x	the object whos support shall be returned
...	further unused arguments for generic use
alpha	a small real number given the portion of the reference support to be ignored to avoid infinite supports.
extrems	a vector of two numerical values used to compute values near -Inf and Inf, to look whether the function is a density (approach zero at both sides) or a cdf (go from 0 to 1)
starts	starting values to find a point where the function is not zero.

Value

An object of class "Support" with the given parameters

Methods (by generic)

- `getSupport(Support)`: method for Support returns itself

Functions

- `getSupport1DPlottingLimits()`: method for getting plotting limits of supports
- `getSupportAtoms()`: method for getting values with mass
- `getSupport()`: get the support associated to an object

- `getSupport(BayesSpace)`: method for BayesSpaces
- `getSupport(BayesSpaceBasis)`: method for BayesSpaces
- `getSupport(BData)`: method for BData
- `getSupport(ReferenceMeasureUniform)`: method for ReferenceMeasureUniform
- `getSupport(ReferenceMeasureDiscrete)`: method for ReferenceMeasureDiscrete
- `getSupport(ReferenceMeasureTransformed)`: method for ReferenceMeasureTransformed
- `getSupport(list)`: combines the supports of all Elements in the list
- `getSupport(` `function` `)`: tries to identify pdfs and cdfs and useds pdf2Support or cdf2Support, a simple 0,1 otherwise.
- `getSupport(numeric)`: guesses the support through sample2Support
- `getSupport(ListData)`: just calls NextMethod to hand over to the other classes
- `getSupport(ReferenceMeasureCartesianProduct)`: returns the product support
- `getSupport(varKernel)`: getter method for varKernel
- `getSupport(variationKernel)`: getter method for variationKernel

See Also

[ReferenceMeasureCartesianProduct], [SupportCartesianProduct]

Other Support: `cdf2Support()`, `pdf2Support()`, `plotFuncs()`, `sample2Support()`

Examples

```
Support(0,1)
Support(X=1:4)
getSupport(StdReferenceMeasureUnif01)
getSupport(StdReferenceMeasureN01)
```

SupportCartesianProduct

Creates a cartesian product of supports

Description

Creates a cartesian product of supports

Usage

```
SupportCartesianProduct(...)
```

Arguments

...	Give the supports, either directly or as a simple list. If they correspond to named columns in a dataframe, name them

Value

A Support of type SupportCartesianProduct that provides the Cartesian product of the supports

Supports are essentially encoded sets or spaces of possible values. The cartesian product of such spaces is the space containing the tuples of with one entry from each space. In R it will be encoded by the columns of a data.frame.

See Also

[Support], [data.frame]

Other Cartesian: BayesSpaceBasis.BayesSpaceCartesianProduct(), BayesSpaceBasisCartesianProduct(), BayesSpaceCartesianProduct(), ReferenceMeasureCartesianProduct(), cartesianProduct(), getBayesSpace.ReferenceMeasureCartesianProduct(), getCartesianFactors(), subBasisIndices()

Examples

```
s1 = Support(-Inf,Inf,lim=c(-2,2))
s2 = Support(0,2)
s3 = Support(X=c(1,2,3))
s4 = Support(X=c("A","B","C","D"))
s  = SupportCartesianProduct(s1,s2,s3,s4)
```

---

supportDiscretization *Computes the discretization of a mixed support*

Description

This provides discretization points for functions with a individual jumps for the purpose of graphical display, i.e. with equispaced points in lim

Usage

```
supportDiscretization(support, ...)

## S3 method for class 'RealSupport'
supportDiscretization(support, nEval = 513, ...)

## S3 method for class 'DiscreteSupport'
supportDiscretization(support, ...)
```

Arguments

support	the [Support] to be discretized (or an object which has a support)
...	further generic arguments
nEval	the number of gridpoints to discretize the continuous part (defaults to 513)

Value

a list with the elements

`x`	the continuous discretization values
`X`	the discrete values
`Xplot`	the location to plot the discrete values
`Xlab`	labels to plot the discrete values
`all`	X and x combined
`iX`	the indices to access X in all
`ix`	the indices to access x in all

Methods (by class)

- `supportDiscretization(RealSupport)`: method for real supports
- `supportDiscretization(DiscreteSupport)`: method for real supports

Examples

```
supportDiscretization(Support(1,3,X=seq(1,2,by=0.1)))
```

---

toHistogram *Convert something to a histogram*

Description

The function tries to convert anything it gets to a histogram

Usage

```
toHistogram(X, breaks, ..., nsim = NULL, seed = NULL)

## S3 method for class 'numeric'
toHistogram(X, breaks, ..., nsim = NULL, seed = NULL)

## S3 method for class 'histogram'
toHistogram(X, breaks = NULL, ..., nsim = NULL, seed = NULL)

## S3 method for class 'BDataHistograms'
toHistogram(X, breaks, ..., nsim = NULL, seed = NULL)

## S3 method for class 'BDataUnivariateSamples'
toHistogram(X, breaks, ..., nsim = NULL, seed = NULL)

## S3 method for class 'BData'
toHistogram(X, breaks, ..., nsim = NULL, seed = NULL)
```

Arguments

X	the single data representing the histogram.
breaks	the breaks of the histogram. Most methods only work if given explicitly.
...	further arguments to the [graphics::hist] if called
nsim	number of datapoints to be simulated. If null, no simulation will take place.
seed	if not null the given seed will be used in case of a simulation and will be added as an attribute to the result (use with care).

Value

a histogram object, either with or without a counts element, depending on whether count data is availabel (e.g. through simulation.)

Methods (by class)

- `toHistogram(numeric)`: method computing the histogram from a sample
- `toHistogram(histogram)`: checks that the breaks fit and returns the histogram if they do.
- `toHistogram(BDataHistograms)`: checks that the breaks of the single histogram fit and returns the histogram if they do.
- `toHistogram(BDataUnivariateSamples)`: computes the histogram with the given breaks from the dataset.
- `toHistogram(BData)`: (not yet implemented) if nsim given, simulates a dataset from the distribution and computes a histogram from it, otherwise ...

See Also

Other BayesSpaceConverters: `as.BDataDensities()`, `as.BDataDensityHistograms()`, `as.BDataFrequencyHistograms` `as.BDataHistograms()`, `as.BDataLogDensities()`, `as.BDataLogPDFs()`, `as.BDataPDFs()`, `as.BDataSamples()`, `as.BDataSplines()`, `idt.BData()`

Examples

```
myBreaks = seq(0,1,by=0.2)
toHistogram(runif(100),breaks=myBreaks)
toHistogram(hist(runif(100),plot=FALSE,breaks=myBreaks),breaks=myBreaks)
toHistogram(BDataSamples(list(runif(100)),basis=B2Unif01Basis5),
            breaks=myBreaks)
```

variationKernel *Compute the variation kernel*

Description

Variance in Bayes spaces can be expressed in various forms. The variation kernel is essentially the variance of the difference of the log-densities or clr-transforms at evaluated at different values

Usage

```
variationKernel(X, ...)

## S3 method for class 'BDataSplines'
variationKernel(X, ...)

## S3 method for class 'BData'
variationKernel(X, ...)

## S3 method for class '`function`'
variationKernel(X, basis = getBayesSpaceBasis(X), ...)

## S3 method for class 'variationKernel'
variationKernel(X, basis = getBayesSpaceBasis(X), ...)
```

Arguments

X	a dataset of BData objects
...	further parameters detailing computation
basis	the BayesSpaceBasis for which this covariance kernel is valid

Details

v(x) = (var(clr(X)(x_i) - clr(X)(y_j)))_ij

It relates to the covariance like the variogram of the residual mean corrected field corresponds to the covariance. The advantage of the variation kernel is its subcompositional coherence, i.e. it is independent of support and reference measure.

Value

a function of two arguments with a basis attribute representing the BayesSpaceBasis, and variationKernel class.

Methods (by class)

- `variationKernel(BDataSplines)`: method for BDataSplines uses the covariance of the spline coefficients to compute

- `variationKernel(BData)`: default method for BData computes implicitly using clr transforms.
- `variationKernel(`function`)`: constructor
- `variationKernel(variationKernel)`: method for varKernel, keeps the kernel, possibly overwrites the basis.

See Also

plot.variationKernel, varKernel

Examples

```
data(DTD)
X = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
V = variationKernel(X)
V
plot(V)

K = BDataSplines(matrix(rnorm(10*5),ncol=5),basis=B2Unif01Basis5)
V2 = variationKernel(K)
V2
plot(V2)
```

`varKernel` *Compute the kernel of the Variance Kovariance operator*

Description

Variance in Bayes spaces can be expressed in various forms. The variance kernel is essentially the covariance function of the clr-transform of a random variable:

Usage

```
varKernel(X, ...)

## S3 method for class 'BDataSplines'
varKernel(X, ...)

## S3 method for class 'BData'
varKernel(X, ...)

## S3 method for class '`function`'
varKernel(X, basis = getBayesSpaceBasis(X), ...)

## S3 method for class 'varKernel'
varKernel(X, basis = getBayesSpaceBasis(X), ...)
```

Arguments

`X`	a dataset of BData objects
`...`	further parameters detailing computation
`basis`	the BayesSpaceBasis for which this covariance kernel is valid

Details

c(x,y) = cov(clr(X)(x),clr(X)(y))

Value

a function of two arguments with a basis attribute representing the BayesSpaceBasis, and varKernel class.

Methods (by class)

- `varKernel(BDataSplines)`: method for BDataSplines uses the covariance of the spline coefficients to compute
- `varKernel(BData)`: default method for BData computes implicitly using clr transforms.
- `varKernel(` `function` `)`: constructor
- `varKernel(varKernel)`: method for varKernel, keeps the kernel, possibly overwrites the basis.

See Also

plot.varKernel, variationKernel

Examples

```
data(DTD)
X = BDataPDFs(DTD$shapePDF,basis=B2Unif01Basis5)
C = varKernel(X)
C
plot(C)

K = BDataSplines(matrix(rnorm(10*5),ncol=5),basis=B2Unif01Basis5)
var(K)
C2 = varKernel(K)
C2
plot(C2)
```

weightedStats *Computes mean and variance for weighted reference measures*

Description

The function computes the expectation and variance of 'X' for the probability distribution given by the density f with respect to the measure. The method supports lists even if the individual measure does not.

Usage

```
weightedStats(
  measure,
  X,
  f,
  method = c("default", "sample", "MonteCarlo", "integration"),
  tol = 0.01
)

## S3 method for class 'ReferenceMeasure'
weightedStats(
  measure,
  X,
  f,
  method = c("default", "sample", "MonteCarlo", "integration"),
  ...
)

## S3 method for class 'ReferenceMeasureDiscrete'
weightedStats(measure, X, f, method, tol)
```

Arguments

measure	should be a reference measure. Other classes can implement the method.
X	a function (or list of functions) providing the statistics
f	a function providing the density with respect to X
method	a method to be selected. Typical methods are "sample", "MonteCarlo" and "integrate", but only sample is availabel for all reference measures. "sample" uses the prestored sample, "MonteCarlo" uses the simulated function of the reference measure, and "integrate" uses numerical integration. Typically the prestored integration sample performs well with derivatives measures derived ReferenceMeasureUniform, ReferenceMeasusureTransformed, where it proves a dense grid in the area of measure, and ReferenceMeasure discrete where it uses precise computation using a weighted sample. For Cartesian products involving more than one continues measure due to the curse of dimensionality typically only MonteCarlo is expected to work reasonably well.

`tol`	the absolute or relative tolerance (whichever is larger). accepted for methods with tolerance control.
`...`	further arguments to integration methods e.g. tol.

Details

The default method for reference measures used [getSample] and Monte Carlo integration with respect to this sample to perform the computation. More advanced methods e.g. using numerical integration might be implemented for certain subclasses.

Value

a list with the elements E for the computed expectation and V for the computed variance.

Methods (by class)

- `weightedStats(ReferenceMeasure)`: method for ReferenceMeasure
- `weightedStats(ReferenceMeasureDiscrete)`: method for ReferenceMeasureDiscrete exact computation based on X and w of the measure.

See Also

[getSample]

Examples

```
weightedStats(ReferenceMeasureUniform(0,1),function(x) x, function(x) x^2)
weightedStats(ReferenceMeasureUniform(0,1),as.function(B2Unif01Basis5), function(x) x^2)
```

Index

∗ **.DensitFromHist**
.DensityFromHist, 5
.LogDensityFromHist, 6
∗ **Add**
Add, 10
Div, 87
Mul, 130
Sub, 189
∗ **BDataBayesSpaceSplines**
ReferenceMeasureUniform, 174
∗ **BDataDensities**
as.BDataDensities, 18
∗ **BDataHistograms**
as.BDataDensityHistograms, 20
as.BDataDensityHistograms.list, 21
as.BDataFrequencyHistograms.list, 23
BDataFrequencyHistograms, 52
getBreaks, 99
plot.BDataHistograms, 138
∗ **BDataLogDensities**
as.BDataLogDensities, 25
∗ **BDataSamples**
BDataSamples, 56
∗ **BDataSplines**
BDataSplines, 57
plot.BDataSplines, 140
∗ **BData**
BDataDensities, 49
BDataDensityHistograms, 51
BDataLogDensities, 54
∗ **BayesSpaceBasis**
plot.BayesSpaceBasis, 138
∗ **BayesSpaceConverters**
as.BDataDensities, 18
as.BDataDensityHistograms, 20
as.BDataFrequencyHistograms, 22
as.BDataHistograms, 24
as.BDataLogDensities, 25
as.BDataLogPDFs, 27
as.BDataPDFs, 29
as.BDataSamples, 31
as.BDataSplines, 32
idt.BData, 111
toHistogram, 198
∗ **BayesSpace**
BayesSpace, 39
L2Space, 120
∗ **Cartesian**
BayesSpaceBasis.BayesSpaceCartesianProduct, 41
BayesSpaceBasisCartesianProduct, 47
BayesSpaceCartesianProduct, 47
cartesianProduct, 63
getBayesSpace.ReferenceMeasureCartesianProduct, 96
getCartesianFactors, 100
ReferenceMeasureCartesianProduct, 171
subBasisIndices, 192
SupportCartesianProduct, 196
∗ **Exp**
Ln, 124
∗ **PCA**
biplot.princomp.BData, 58
plot.princomp.BData, 141
plot.princomp.BData.grouped, 145
plot.princomp.BData.multivariate, 147
predict.princomp.BData, 157
princomp.BData, 158
screeplot.princomp.BData, 179
summary.princomp.BData, 193
∗ **ReferenceMeasures**
B2N01, 37
expectation, 91
getSample, 107

isInDomain, 114
ReferenceMeasureDiscrete, 172
ReferenceMeasureTransformed, 173
ReferenceMeasureUniform, 174
StdReferenceMeasureUnif01, 188
∗ **Support**
cdf2Support, 70
pdf2Support, 137
plotFuncs, 153
sample2Support, 176
Support, 194
∗ **caseStatistics**
caseE, 64
caseLogRatios, 65
caseMeans, 66
caseQuantiles, 66
caseSds, 67
caseVars, 68
centralityQuantile, 73
distanceToCenter, 86
median.BData, 129
spatialDepth, 187
∗ **datasets**
age, 15
B2N01, 37
DTD, 89
ListDataEnv, 122
StdReferenceMeasureUnif01, 188
∗ **getBayesSpace**
getBayesSpace, 95
getL2Space, 102
*.BData *(+.BData)*, 9
+.BData, 9
-.BData *(+.BData)*, 9
.DensityFromHist, 5, *7*
.LogDensityFromHist, 6, *6*
/.BData *(+.BData)*, 9
[.BDataHistograms, 7
[.BDataSamples *(BDataSamples)*, 56
[.BDataSplines *(BDataSplines)*, 57
[.BDataUnivariateSamples *(BDataSamples)*, 56
%..% *(%.%)*, 8
%.%, 8

Add, 10, *89*, *132*, *192*
addFrequency, 13
age, 15
as.BayesSpaceBasis, 17
as.BDataDensities, 18, *21*, *23*, *25*, *27*, *29*, *31*, *32*, *34*, *112*, *199*
as.BDataDensityHistograms, 20, *20*, *22*, *23*, *25*, *27*, *29*, *31*, *32*, *34*, *53*, *99*, *112*, *139*, *199*
as.BDataDensityHistograms.list, 21, *21*, *23*, *53*, *99*, *139*
as.BDataFrequencyHistograms, *20*, *21*, 22, *25*, *27*, *29*, *31*, *32*, *34*, *112*, *199*
as.BDataFrequencyHistograms.list, *21*, *22*, 23, *53*, *99*, *139*
as.BDataHistograms, *20*, *21*, *23*, 24, *27*, *29*, *31*, *32*, *34*, *112*, *199*
as.BDataLogDensities, *20*, *21*, *23*, 25, *25*, *29*, *31*, *32*, *34*, *112*, *199*
as.BDataLogPDFs, *20*, *21*, *23*, *25*, 27, *27*, *31*, *32*, *34*, *112*, *199*
as.BDataPDFs, *20*, *21*, *23*, *25*, *27*, 29, *29*, *32*, *34*, *112*, *199*
as.BDataSamples, *20*, *21*, *23*, *25*, *27*, *29*, 31, *31*, *34*, *112*, *199*
as.BDataSplines, *20*, *21*, *23*, *25*, *27*, *29*, *31*, 32, *32*, *112*, *160*, *162*, *199*
as.data.frame.BData *(ListDataEnv)*, 122
as.data.frame.ListData *(ListDataEnv)*, 122
as.function.BayesSpaceBasis *(basisEval)*, 38

B2DistHyper, 34
B2GaussianDistribution, 35, *161*
B2N01, 37, *92*, *108*, *115*, *173–175*, *189*
B2N01Basis20 *(B2N01)*, 37
B2N01Basis5 *(B2N01)*, 37
B2N01Basis7 *(B2N01)*, 37
B2Unif01 *(StdReferenceMeasureUnif01)*, 188
B2Unif01Basis20 *(StdReferenceMeasureUnif01)*, 188
B2Unif01Basis5 *(StdReferenceMeasureUnif01)*, 188
B2Unif01Basis7 *(StdReferenceMeasureUnif01)*, 188
basisEval, 38
BayesSpace, 39, *121*
BayesSpaceBasis, 39, *160*

BayesSpaceBasis.BayesSpaceCartesianProduct, 41, *47, 48*, *64*, *97*, *100*, *172*, *192*, *197*
BayesSpaceBasis.function, 42
BayesSpaceBasis.ReferenceMeasureCartesianProduct *(*BayesSpaceBasis.BayesSpaceCartesianProduct*)*, 41
BayesSpaceBasis.ReferenceMeasureDiscrete, 44
BayesSpaceBasis.ReferenceMeasureTransformed, 45
BayesSpaceBasis.ReferenceMeasureUniform, 46
BayesSpaceBasisCartesianProduct, *42*, 47, *48*, *64*, *97*, *100*, *172*, *192*, *197*
BayesSpaceCartesianProduct, *42*, 47, *47*, *64*, *97*, *100*, *172*, *192*, *197*
BData, 48
BDataDensities, 49, *52*, *56*
BDataDensities.BDataLogDensities *(*BDataLogDensities*)*, 54
BDataDensityHistograms, 51, *51*, *56*
BDataFrequencyHistograms, *21–23*, 52, *99*, *139*
BDataGeneral, 53
BDataLogDensities, *51, 52*, 54
BDataLogDensities.list *(*BDataDensities*)*, 49
BDataLogPDFs *(*BDataLogDensities*)*, 54
BDataLogPDFs.list *(*BDataDensities*)*, 49
BDataPDFs *(*BDataDensities*)*, 49
BDataPDFs.BDataLogPDFs *(*BDataLogDensities*)*, 54
BDataSamples, 56
BDataSplines, 57, *140*
biplot, *61*
biplot.princomp.BData, 58, *142*, *144*, *147, 148*, *158*, *162*, *181*, *193*
boxplot.BData, 61
boxplot.BDataSamples *(*boxplot.BData*)*, 61
boxplot.BDataSplines *(*boxplot.BData*)*, 61

cartesianProduct, *42*, *47, 48*, 63, *97*, *100*, *172*, *192*, *197*
caseE, 64, *65–69*, *75*, *87*, *130*, *188*
caseLogRatios, *64*, 65, *66–69*, *75*, *87*, *130*, *188*
caseMeans, *64, 65*, 66, *67–69*, *75*, *87*, *130*, *188*
caseQuantiles, *64, 65*, 66, *66*, *68, 69*, *75*, *87*, *130*, *188*
caseSds, *64–66*, 67, *67*, *69*, *75*, *87*, *130*, *188*
caseVars, *64–67*, 68, *68*, *75*, *87*, *130*, *188*
cdf2density, 69
cdf2Support, 70, *137*, *156*, *176*, *196*
cov, 71
centeringConstant, 72
centralityQuantile, *64–69*, 73, *87*, *130*, *188*
centralityRank *(*centralityQuantile*)*, 73
clo, 75
clo.BDataHistograms, 76
closingConstant, 77
clr, 78
computeL22BayesSpacesBasisMatrix, 79
computeL22BayesSpacesBasisMatrix.basisfd, 80
computeLimit, 81
CovarianceOperator, 81

DiscretizedFunction, 83
dist.BData, 85
dist.BDataLogDensities *(*dist.BData*)*, 85
dist.BDataSplines *(*dist.BData*)*, 85
distanceToCenter, *64–69*, *75*, 86, *130*, *188*
Div, *13*, 87, *132*, *192*
DTD, 89

ensureType, 90
Exp *(*Ln*)*, 124
expectation, *37*, 91, *108*, *115*, *173–175*, *189*
expectation.ReferenceMeasureUniform, 92

findJumps, 93
format.BData *(*ListDataEnv*)*, 122
format.ListData *(*ListDataEnv*)*, 122
fx, 94

getBayesSpace, 95, *103*
getBayesSpace.ReferenceMeasureCartesianProduct, *42*, *47, 48*, *64*, 96, *100*, *172*, *192*, *197*
getBayesSpaceBasis, 97
getBayesSpaceBasisDimension, 98
getBreaks, *21–23*, *53*, 99, *139*
getCartesianFactors, *42*, *47, 48*, *64*, *97*, 100, *172*, *192*, *197*
getDimension, 100
getDiscretizer, 101
getL2Space, *96*, 102

getL2Space.L2Space *(getBayesSpace)*, 95
getMetricTensor, 104
getPDF, 105
getReferenceMeasure, 106
getSample, *37*, *92*, 107, *115*, *173–175*, *189*
getSample.ReferenceMeasureTransformed *(ReferenceMeasureTransformed)*, 173
getSplineCoefs, 109
getSupport *(Support)*, 194
getSupport1DPlottingLimits *(Support)*, 194
getSupportAtoms *(Support)*, 194
getWeights *(getSample)*, 107

hasSameBayesSpace, 110
hasSameBayesSpaceBasis, 111
hasWeights *(getSample)*, 107

idt.BData, *20, 21*, *23*, *25*, *27*, *29*, *31, 32*, *34*, 111, *199*
idtInv.BData *(idt.BData)*, 111
idtInv.BDataSplines *(idt.BData)*, 111
isCentered *(isClosed)*, 112
isClosed, 112
isInDomain, *37*, *92*, *108*, 114, *173–175*, *189*
isInDomain.ReferenceMeasureUniform, 115
isValidBayesSpaceBasis, 116
isValidBayesSpaceElement, 116

kdeDensity, 118

L2Space, *39*, 120
L2SpaceCartesianProduct *(BayesSpaceCartesianProduct)*, 47
ListData *(ListDataEnv)*, 122
listDataElementAccess, 122
ListDataEnv, 122
ListDataRecover *(ListDataEnv)*, 122
Ln, 124

makeFunctionOuter, 126
makeLimitStable, 126
mapToRealLineWithSupport, 127
maxFunc, 127
mean.BDataDensities *(mean.BDataSplines)*, 128
mean.BDataLogDensities *(mean.BDataSplines)*, 128
mean.BDataLogPDFs *(mean.BDataSplines)*, 128
mean.BDataPDFs *(mean.BDataSplines)*, 128
mean.BDataSplines, 128
median.BData, *64–69*, *75*, *87*, 129, *188*
minFunc *(maxFunc)*, 127
msd.BData *(mvar.BData)*, 133
Mul, *13*, *89*, 130, *192*
mvar.BData, 133
mvar.BDataSplines *(mvar.BData)*, 133

nCases, 134
norm, 135
numberOfCases, 136

pdf2Support, *70*, 137, *156*, *176*, *196*
plot.BayesSpaceBasis, 138
plot.BDataDensities *(plotFuncs)*, 153
plot.BDataHistograms, *21–23*, *53*, *99*, 138
plot.BDataLogDensities *(plotFuncs)*, 153
plot.BDataLogPDFs *(plotFuncs)*, 153
plot.BDataPDFs *(plotFuncs)*, 153
plot.BDataSamples, 139
plot.BDataSplines, *58*, 140
plot.princomp.BData, *60, 61*, 141, *146–148*, *158*, *161, 162*, *181*, *193*
plot.princomp.BData.grouped, *61*, *144*, 145, *148*, *158*, *162*, *181*, *193*
plot.princomp.BData.multivariate, *61*, *144*, 147, *147*, *158*, *162*, *181*, *193*
plot.variationKernel *(plot.varKernel)*, 149
plot.varKernel, 149
plotCoefs, 151
plotFuncs, *70*, *137*, 153, *176*, *196*
predict.princomp.BData, *61*, *144*, *147, 148*, 157, *162*, *181*, *193*
princomp.acomp, *161, 162*
princomp.BData, *59*, *61*, *141*, *144*, *146–148*, *157*, 158, *158*, *181*, *193*
print.BayesSpace, 165
print.BDataLogDensities, 166
print.BDataLogPDFs, 167
print.BDataSplines, 167
print.L2Space, 168
print.princomp.BData *(princomp.BData)*, 158

print.princomp.BData.grouped, 168
print.princomp.BData.multivariate, 169
print.ReferenceMeasure, 170

ReferenceMeasure, 171
ReferenceMeasureCartesianProduct, *42*, *47, 48*, *64*, *97*, *100*, 171, *192*, *197*
ReferenceMeasureDiscrete, *37*, *92*, *108*, *115*, 172, *174, 175*, *189*
ReferenceMeasureTransformed, *37*, *92*, *108*, *115*, 173, *173*, *175*, *189*
ReferenceMeasureUniform, *37*, *92*, *108*, *115*, *173*, 174, *174*, *189*

sample2Support, *70*, *137*, *156*, 176, *196*
scalarProduct, 177
scalarProductOfLogDensities, 178
scale.BData, 179
screeplot.princomp.BData, *61*, *144*, *147, 148*, *158*, *162*, 179, *193*
selectCases *(*nCases*)*, 134
selectIntegrationMethod, 181
simulate.B2DistHyper, 182
simulate.B2GaussianDistribution *(*B2GaussianDistribution*)*, 35
simulate.BData, 183
simulate.ReferenceMeasureCartesianProduct, 184
simulate.ReferenceMeasureDiscrete, 185
simulate.ReferenceMeasureTransformed, 185
simulate.ReferenceMeasureUniform, 186
spatialDepth, *64–69*, *75*, *87*, *130*, 187
StdReferenceMeasureN01 *(*StdReferenceMeasureUnif01*)*, 188
StdReferenceMeasureUnif01, *37*, *92*, *108*, *115*, *173–175*, 188
storingDigester *(*ListDataEnv*)*, 122
Sub, *13*, *89*, *132*, 189
subBasisIndices, *42*, *47, 48*, *64*, *97*, *100*, *172*, 192, *197*
summary.BDataSamples *(*BDataSamples*)*, 56
summary.BDataUnivariateSamples *(*BDataSamples*)*, 56
summary.princomp.BData, *61*, *144*, *147, 148*, *158*, *162*, *181*, 193
Support, *70*, *137*, *156*, *176*, 194
SupportCartesianProduct, *42*, *47, 48*, *64*, *97*, *100*, *172*, *192*, 196
supportDiscretization, 197

toHistogram, *20, 21*, *23*, *25*, *27*, *29*, *31, 32*, *34*, *112*, 198

variationKernel, 200
varKernel, 201

weightedStats, 203