Package ‘distrEx’

March 29, 2019

Version 2.8.0
Date 2019-03-29
Title Extensions of Package ‘distr’
Description Extends package ‘distr’ by functionals, distances, and conditional distributions.
Depends R(>= 3.4), methods, distr(>= 2.8.0)
Imports startupmsg, utils, stats
Suggests tcltk
ByteCompile yes
License LGPL-3
Encoding latin1

URL http://distr.r-forge.r-project.org/

R topics documented:

distrEx-package ..................................................... 2
AbscontCondDistribution-class .................................. 7
AsymTotalVarDist .................................................. 8
Condition-class .................................................... 12
ContaminationSize ............................................... 12
**Description**

*distrEx* provides some extensions of package *distr*:

- expectations in the form
  - $E(X)$ for the expectation of a distribution object $X$
  - $E(X, f)$ for the expectation of $f(X)$ where $X$ is some distribution object and $f$ some function in $X$
• further functionals: var, sd, IQR, mad, median, skewness, kurtosis
• truncated moments,
• distances between distributions (Hellinger, Cramer von Mises, Kolmogorov, total variation, "convex contamination")
• lists of distributions,
• conditional distributions in factorized form
• conditional expectations in factorized form

Support for extreme value distributions has moved to package RobExtremes

Details

<table>
<thead>
<tr>
<th>Package</th>
<th>distrEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>2.8.0</td>
</tr>
<tr>
<td>Date</td>
<td>2019-03-29</td>
</tr>
<tr>
<td>Depends</td>
<td>R(&gt;= 3.4), methods, distr(&gt;= 2.8.0)</td>
</tr>
<tr>
<td>Imports</td>
<td>startupmsg, utils, stats</td>
</tr>
<tr>
<td>Suggests</td>
<td>tcltk</td>
</tr>
<tr>
<td>LazyLoad</td>
<td>yes</td>
</tr>
<tr>
<td>License</td>
<td>LGPL-3</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://distr.r-forge.r-project.org/">http://distr.r-forge.r-project.org/</a></td>
</tr>
<tr>
<td>VCS/SVNRevision</td>
<td>1332</td>
</tr>
</tbody>
</table>

Classes

Distribution Classes

"Distribution" (from distr)
  |"UnivariateDistribution" (from distr)
  |"AbscontDistribution" (from distr)
  ||"Gumbel" (moved to package 'RobExtremes')
  ||"Pareto" (moved to package 'RobExtremes')
  ||"GPareto" (moved to package 'RobExtremes')
  |"MultivariateDistribution"
  |"DiscreteMVDistribution-class"
  |"UnivariateCondDistribution"
  |"AbscontCondDistribution"
  ||"PrognCondDistribution"
  ||"DiscreteCondDistribution"

Condition Classes
"Condition"  
|"EuclCondition"  
|"PrognCondition"

Parameter Classes

"OptionalParameter" (from distr)  
|"Parameter" (from distr)  
|"LMParameter"  
|"GumbelParameter"  
|"ParetoParameter"

Functions

Integration:
GLintegrate Gauss-Legendre quadrature
distrExIntegrate Integration of one-dimensional functions

Options:
distrExOptions Function to change the global variables of the package 'distrEx'

Standardization:
make@1 Centering and standardization of univariate distributions

Generating Functions

Distribution Classes
ConvexContamination Generic function for generating convex contaminations
DiscreteMVDistribution Generating function for DiscreteMVDistribution-class
Gumbel Generating function for Gumbel-class
LMCondDistribution Generating function for the conditional distribution of a linear regression model.

Condition Classes
EuclCondition Generating function for EuclCondition-class

Parameter Classes
LMParameter Generating function for LMParameter-class
Methods

Distances:
- ContaminationSize: Generic function for the computation of the convex contamination (Pseudo-)distance of two distributions
- HellingerDist: Generic function for the computation of the Hellinger distance of two distributions
- KolmogorovDist: Generic function for the computation of the Kolmogorov distance of two distributions
- TotalVarDist: Generic function for the computation of the total variation distance of two distributions
- AsymTotalVarDist: Generic function for the computation of the asymmetric total variation distance of two distributions (for given ratio rho of negative to positive part of deviation)
- OAAsymTotalVarDist: Generic function for the computation of the minimal (in rho) asymmetric total variation distance of two distributions
- vonMisesDist: Generic function for the computation of the von Mises distance of two distributions

liesInSupport: Generic function for testing the support of a distribution

Functionals:
- E: Generic function for the computation of (conditional) expectations
- var: Generic functions for the computation of functionals
- IQR: Generic functions for the computation of functionals
- sd: Generic functions for the computation of functionals
- mad: Generic functions for the computation of functionals
- median: Generic functions for the computation of functionals
- skewness: Generic functions for the computation of functionals
- kurtosis: Generic functions for the computation of functionals

truncated Moments:
- m1df: Generic function for the computation of clipped first moments
- m2df: Generic function for the computation of clipped second moments
Demos

Demos are available — see demo(package="distrEx").

Acknowledgement

G. Jay Kerns, <gkerns@ysu.edu>, has provided a major contribution, in particular the functionals skewness and kurtosis are due to him.

Start-up-Banner

You may suppress the start-up banner/message completely by setting options("StartupBanner"="off") somewhere before loading this package by library or require in your R-code / R-session.

If option "StartupBanner" is not defined (default) or setting options("StartupBanner"=NULL) or options("StartupBanner"="complete") the complete start-up banner is displayed.

For any other value of option "StartupBanner" (i.e., not in c(NULL,"off","complete")) only the version information is displayed.

The same can be achieved by wrapping the library or require call into either suppressStartupMessages() or onlytypeStartupMessages(.atypes="version").

As for general packageStartupMessage’s, you may also suppress all the start-up banner by wrapping the library or require call into suppressPackageStartupMessages() from startupmsg-version 0.5 on.

Package versions

Note: The first two numbers of package versions do not necessarily reflect package-individual development, but rather are chosen for the distrXXX family as a whole in order to ease updating "depends" information.

Note

Some functions of package stats have intentionally been masked, but completely retain their functionality — see distrExMASK().

If any of the packages e1071, moments, fBasics is to be used together with distrEx the latter must be attached after any of the first mentioned. Otherwise kurtosis() and skewness() defined as methods in distrEx may get masked.

To re-mask, you may use kurtosis <- distrEx::kurtosis; skewness <- distrEx::skewness.

See also distrExMASK()

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de> and
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>.
Maintainer: Matthias Kohl <Matthias.Kohl@stamats.de>
References


a vignette for packages `distr`, `distrSim`, `distrTEst`,
and `distrEx` is included into the mere documentation package `distrDoc` and may be called by
require("distrDoc");vignette("distr")

a homepage to this package is available under
[http://distr.r-forge.r-project.org/](http://distr.r-forge.r-project.org/)


See Also

`distr-package`

---

**AbscontCondDistribution-class**

Absolutely continuous conditional distribution

**Description**

The class of absolutely continuous conditional univariate distributions.

**Objects from the Class**

Objects can be created by calls of the form `new("AbscontCondDistribution", ...)`.

**Slots**

- `cond` Object of class "Condition": condition
- `img` Object of class "rSpace": the image space.
- `param` Object of class "OptionalParameter": an optional parameter.
- `r` Object of class "function": generates random numbers.
- `d` Object of class "OptionalFunction": optional conditional density function.
- `p` Object of class "OptionalFunction": optional conditional cumulative distribution function.
- `q` Object of class "OptionalFunction": optional conditional quantile function.
- `.withArith` logical: used internally to issue warnings as to interpretation of arithmetics
- `.withSim` logical: used internally to issue warnings as to accuracy
- `.logExact` logical: used internally to flag the case where there are explicit formulae for the log version of density, cdf, and quantile function
- `.lowerExact` logical: used internally to flag the case where there are explicit formulae for the lower tail version of cdf and quantile function
- `Symmetry` object of class "DistributionSymmetry": used internally to avoid unnecessary calculations.
AsymTotalVarDist

Extends

Class "UnivariateCondDistribution", directly.
Class "Distribution", by class "UnivariateCondDistribution".

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

UnivariateCondDistribution-class, Distribution-class

Examples

new("AbscontCondDistribution")

AsymTotalVarDist

Generic function for the computation of asymmetric total variation distance of two distributions

Description

Generic function for the computation of asymmetric total variation distance \( d_v(\rho) \) of two distributions \( P \) and \( Q \) where the distributions may be defined for an arbitrary sample space \((\Omega, A)\). For given ratio of inlier and outlier probability \( \rho \), this distance is defined as

\[
d_v(\rho)(P, Q) = \int (dQ - c \, dP)_+\]

for \( c \) defined by

\[
\rho \int (dQ - c \, dP)_+ = \int (dQ - c \, dP) -
\]

It coincides with total variation distance for \( \rho = 1 \).

Usage

AsymTotalVarDist(e1, e2, ...)  # S4 method for signature 'AbscontDistribution,AbscontDistribution'
AsymTotalVarDist(e1, e2, rho = 1, 
               rel.tol = .Machine$double.eps^0.3, maxiter=1000, Ngrid = 10000, 
               TruncQuantile = getdistrOption("TruncQuantile"), 
               IQR.fact = 15, ..., diagnostic = FALSE)  # S4 method for signature 'AbscontDistribution,DiscreteDistribution'
AsymTotalVarDist(e1, e2, rho = 1, ...)  # S4 method for signature 'DiscreteDistribution,AbscontDistribution'
AsymTotalVarDist(e1, e2, rho = 1, ...)  # S4 method for signature 'DiscreteDistribution,DiscreteDistribution'
AsymTotalVarDist(e1, e2, rho = 1, ...)
AsymTotalVarDist

## S4 method for signature 'numeric,DiscreteDistribution'
AsymTotalVarDist(e1, e2, rho = 1, ...)

## S4 method for signature 'DiscreteDistribution,numeric'
AsymTotalVarDist(e1, e2, rho = 1, ...)

## S4 method for signature 'numeric,AbscontDistribution'
AsymTotalVarDist(e1, e2, rho = 1, asis.smooth.discretize = "discretize",
                 n.discr = getdistrExOption("nDiscretize"),
                 low.discr = getLow(e2),
                 up.discr = getUp(e2),
                 h.smooth = getdistrExOption("hSmooth"),
                 rel.tol = .Machine$double.eps^0.3,
                 maxiter=1000, Ngrid = 10000,
                 TruncQuantile = getdistrOption("TruncQuantile"),
                 IQR.fac = 15, ..., diagnostic = FALSE)

## S4 method for signature 'AbscontDistribution,numeric'
AsymTotalVarDist(e1, e2, rho = 1,
                 asis.smooth.discretize = "discretize",
                 n.discr = getdistrExOption("nDiscretize"),
                 low.discr = getLow(e1),
                 up.discr = getUp(e1),
                 h.smooth = getdistrExOption("hSmooth"),
                 rel.tol = .Machine$double.eps^0.3,
                 maxiter=1000, Ngrid = 10000,
                 TruncQuantile = getdistrOption("TruncQuantile"),
                 IQR.fac = 15, ..., diagnostic = FALSE)

## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
AsymTotalVarDist(e1, e2,
                 rho = 1, rel.tol = .Machine$double.eps^0.3,
                 maxiter=1000, Ngrid = 10000,
                 TruncQuantile = getdistrOption("TruncQuantile"),
                 IQR.fac = 15, ..., diagnostic = FALSE)

### Arguments

e1 object of class "Distribution" or "numeric"
e2 object of class "Distribution" or "numeric"
asis.smooth.discretize possible methods are "asis", "smooth" and "discretize". Default is "discretize".
n.discr if asis.smooth.discretize is equal to "discretize" one has to specify the number of lattice points used to discretize the abs. cont. distribution.
low.discr if asis.smooth.discretize is equal to "discretize" one has to specify the lower end point of the lattice used to discretize the abs. cont. distribution.
up.discr if asis.smooth.discretize is equal to "discretize" one has to specify the upper end point of the lattice used to discretize the abs. cont. distribution.
h.smooth if asis.smooth.discretize is equal to "smooth" – i.e., the empirical distribution of the provided data should be smoothed – one has to specify this parameter.
rho ratio of inlier/outlier radius
rel.tol relative tolerance for distrExIntegrate and uniroot
maxiter parameter for uniroot
Ngrid How many grid points are to be evaluated to determine the range of the likelihood ratio?,
TruncQuantile Quantile the quantile based integration bounds (see details)
Asymmetric Total variation distance of \( e_1 \) and \( e_2 \)

**Methods**

- \( e_1 = "\text{AbscontDistribution}" \), \( e_2 = "\text{AbscontDistribution}" \): total variation distance of two absolutely continuous univariate distributions which is computed using \texttt{distrExIntegrate}.

- \( e_1 = "\text{AbscontDistribution}" \), \( e_2 = "\text{DiscreteDistribution}" \): total variation distance of absolutely continuous and discrete univariate distributions (are mutually singular; i.e., have distance = 1).
AsymTotalVarDist

\textbf{e1 = "DiscreteDistribution", e2 = "DiscreteDistribution":} total variation distance of two discrete univariate distributions which is computed using support and sum.

\textbf{e1 = "DiscreteDistribution", e2 = "AbscontDistribution":} total variation distance of discrete and absolutely continuous univariate distributions (are mutually singular; i.e., have distance $=1$).

\textbf{e1 = "numeric", e2 = "DiscreteDistribution":} Total variation distance between (empirical) data and a discrete distribution.

\textbf{e1 = "DiscreteDistribution", e2 = "numeric":} Total variation distance between (empirical) data and a discrete distribution.

\textbf{e1 = "numeric", e2 = "AbscontDistribution":} Total variation distance between (empirical) data and an abs. cont. distribution.

\textbf{e1 = "AbscontDistribution", e1 = "numeric":} Total variation distance between (empirical) data and an abs. cont. distribution.

\textbf{e1 = "AcDeLcDistribution", e2 = "AcDeLcDistribution":} Total variation distance of mixed discrete and absolutely continuous univariate distributions.

**Author(s)**

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

**References**

to be filled; Agostinelli, C and Ruckdeschel, P. (2009): A simultaneous inlier and outlier model by asymmetric total variation distance.

**See Also**

\texttt{TotalVarDist-methods, ContaminationSize, KolmogorovDist, HellingerDist, Distribution-class}

**Examples**

```r
AsymTotalVarDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
mixCoeff=c(0.2,0.8)), rho=0.3)
AsymTotalVarDist(Norm(), Td(10), rho=0.3)
AsymTotalVarDist(Norm(mean = 50, sd = sqrt(25)), Binom(size = 100), rho=0.3) # mutually singular
AsymTotalVarDist(Pois(10), Binom(size = 20), rho=0.3)

x <- rnorm(100)
AsymTotalVarDist(Norm(), x, rho=0.3)
AsymTotalVarDist(x, Norm(), asis.smooth.discretize = "smooth", rho=0.3)

y <- (rbinom(50, size = 20, prob = 0.5)-10)/sqrt(5)
AsymTotalVarDist(y, Norm(), rho=0.3)
AsymTotalVarDist(y, Norm(), asis.smooth.discretize = "smooth", rho=0.3)

AsymTotalVarDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5), rho=0.3)
```
ContaminationSize

Condition-class Conditions

Description
The class of conditions.

Objects from the Class
Objects can be created by calls of the form new("Condition", ...).

Slots

name Object of class "character": name of the condition

Methods

name signature(object = "Condition"): accessor function for slot name.

cname<- signature(object = "Condition"): replacement function for slot name.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

UnivariateCondDistribution-class

Examples
new("Condition")

ContaminationSize Generic Function for the Computation of the Convex Contamination (Pseudo-)Distance of Two Distributions

Description
Generic function for the computation of convex contamination (pseudo-)distance of two probability distributions P and Q. That is, the minimal size \( \varepsilon \in [0, 1] \) is computed such that there exists some probability distribution \( R \) with

\[
Q = (1 - \varepsilon)P + \varepsilon R
\]
Usage

ContaminationSize(e1, e2, ...)  
## S4 method for signature 'AbscontDistribution,AbscontDistribution'
ContaminationSize(e1, e2)  
## S4 method for signature 'DiscreteDistribution,DiscreteDistribution'
ContaminationSize(e1, e2)  
## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
ContaminationSize(e1,e2)

Arguments

e1 object of class "Distribution"  
e2 object of class "Distribution"  
... further arguments to be used in particular methods (not in package distrEx)

Details

Computes the distance from e1 to e2 respectively P to Q. This is not really a distance as it is not symmetric!

Value

A list containing the following components:

e1 object of class "Distribution"; ideal distribution  
e2 object of class "Distribution"; 'contaminated' distribution  
size.of.contamination size of contamination

Methods

e1 = "AbscontDistribution", e2 = "AbscontDistribution": convex contamination (pseudo-)distance of two absolutely continuous univariate distributions.

e1 = "DiscreteDistribution", e2 = "DiscreteDistribution": convex contamination (pseudo-)distance of two discrete univariate distributions.

e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution": convex contamination (pseudo-)distance of two discrete univariate distributions.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

ConvexContamination

See Also
KolmogorovDist, TotalVarDist, HellingerDist, Distribution-class

Examples
ContaminationSize(Norm(), Norm(mean=0.1))
ContaminationSize(Pois(), Pois(1.5))

ConvexContamination

Generic Function for Generating Convex Contaminations

Description
Generic function for generating convex contaminations. This is also known as gross error model. Given two distributions $P$ (ideal distribution), $R$ (contaminating distribution) and the size $\varepsilon \in [0, 1]$ the convex contaminated distribution

$$Q = (1 - \varepsilon)P + \varepsilon R$$

is generated.

Usage
ConvexContamination(e1, e2, size)

Arguments
- **e1** object of class "Distribution": ideal distribution
- **e2** object of class "Distribution": contaminating distribution
- **size** size of contamination (amount of gross errors)

Value
Object of class "Distribution".

Methods
- **e1 = "UnivariateDistribution", e2 = "UnivariateDistribution", size = "numeric"**: convex combination of two univariate distributions
- **e1 = "AbscontDistribution", e2 = "AbscontDistribution", size = "numeric"**: convex combination of two absolutely continuous univariate distributions
- **e1 = "DiscreteDistribution", e2 = "DiscreteDistribution", size = "numeric"**: convex combination of two discrete univariate distributions
- **e1 = "AcDeLcDistribution", e2 = "AcDeLcDistribution", size = "numeric"**: convex combination of two univariate distributions which may be coerced to "UnivarLebDecDistribution".
CvMDist

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

References

See Also
ContaminationSize, Distribution-class

Examples

# Convex combination of two normal distributions
Cl <- ConvexContamination(e1 = Norm(), e2 = Norm(mean = 5, size = 0.1))
plot(Cl)

---

CvMDist

Generic function for the computation of the Cramer - von Mises distance of two distributions

Description
Generic function for the computation of the Cramer - von Mises distance $d_\mu$ of two distributions $P$ and $Q$ where the distributions are defined on a finite-dimensional Euclidean space $(\mathbb{R}^m, B^m)$ with $B^m$ the Borel-$\sigma$-algebra on $\mathbb{R}^m$. The Cramer - von Mises distance is defined as

$$d_\mu(P, Q)^2 = \int (P\{y \in \mathbb{R}^m | y \leq x\} - Q\{y \in \mathbb{R}^m | y \leq x\})^2 \mu(dx)$$

where $\leq$ is coordinatewise on $\mathbb{R}^m$.

Usage

CvMDist(e1, e2, ...)

## S4 method for signature 'UnivariateDistribution, UnivariateDistribution'
CvMDist(e1, e2, mu = e1, useApply = FALSE, ..., diagnostic = FALSE)

## S4 method for signature 'numeric, UnivariateDistribution'
CvMDist(e1, e2, mu = e1, ..., diagnostic = FALSE)

Arguments

- **e1**: object of class "Distribution" or class "numeric"
- **e2**: object of class "Distribution"
- **...**: further arguments to be used e.g. by E()
- **useApply**: logical; to be passed to E()
- **mu**: object of class "Distribution"; integration measure; defaulting to e2
diagnostic logical; if TRUE, the return value obtains an attribute "diagnostic" with diagnostic information on the integration, i.e., a list with entries method("integrate" or "Gl.Integrate"), call, result (the complete return value of the method), args (the args with which the method was called), and time (the time to compute the integral).

Details

Diagnostics on the involved integrations are available if argument diagnostic is TRUE. Then there is attribute diagnostic attached to the return value, which may be inspected and accessed through showDiagnostic and getDiagnostic.

Value

Cramer - von Mises distance of e1 and e2

Methods

- \texttt{e1 = "UnivariateDistribution", e2 = "UnivariateDistribution"): Cramer - von Mises distance of two univariate distributions.}
- \texttt{e1 = "numeric", e2 = "UnivariateDistribution"): Cramer - von Mises distance between the empirical formed from a data set (e1) and a univariate distribution.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References


See Also

\texttt{ContaminationSize, TotalVarDist, HellingerDist, KolmogorovDist, Distribution-class}

Examples

\begin{verbatim}
CvMDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
  mixCoeff=c(0.2,0.8)))
CvMDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
  mixCoeff=c(0.2,0.8)),mu=Norm())
CvMDist(Norm(), Td(10))
CvMDist(Norm(mean = 50, sd = sqrt(25)), Binom(size = 100))
CvMDist(Pois(10), Binom(size = 20))
CvMDist(rnorm(100),Norm())
CvMDist((rbinom(50, size = 20, prob = 0.5)-10)/sqrt(5), Norm())
CvMDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5))
CvMDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5), mu = Pois())
\end{verbatim}
Description

dim-methods

Methods

\texttt{dim} \ signature(object = "DiscreteMVDistribution"): returns the dimension of the distribution.

See Also

dim-methods, dim

DiscreteCondDistribution-class

\textit{Discrete conditional distribution}

Description

The class of discrete conditional univariate distributions.

Objects from the Class

Objects can be created by calls of the form \texttt{new("DiscreteCondDistribution", \ldots)}.

Slots

- \texttt{support}: Object of class "function": conditional support.
- \texttt{cond}: Object of class "Condition": condition
- \texttt{img}: Object of class "rSpace": the image space.
- \texttt{param}: Object of class "OptionalParameter": an optional parameter.
- \texttt{r}: Object of class "function": generates random numbers.
- \texttt{d}: Object of class "OptionalFunction": optional conditional density function.
- \texttt{p}: Object of class "OptionalFunction": optional conditional cumulative distribution function.
- \texttt{q}: Object of class "OptionalFunction": optional conditional quantile function.
- \texttt{.withArith}: logical: used internally to issue warnings as to interpretation of arithmetics
- \texttt{.withSim}: logical: used internally to issue warnings as to accuracy
- \texttt{.logExact}: logical: used internally to flag the case where there are explicit formulae for the log version of density, cdf, and quantile function.
DiscreteMVDistribution

 lý logical: used internally to flag the case where there are explicit formulae for the lower tail version of cdf and quantile function.

Symmetry object of class "DistributionSymmetry": used internally to avoid unnecessary calculations.

Extends

Class "UnivariateCondDistribution", directly.
Class "Distribution", by class "UnivariateCondDistribution".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

UnivariateCondDistribution-class

Examples

new("DiscreteCondDistribution")

DiscreteMVDistribution

Generating function for multivariate discrete distribution

Description

Generates an object of class "DiscreteMVDistribution".

Usage

DiscreteMVDistribution(supp, prob, Symmetry = NoSymmetry())

Arguments

supp numeric matrix whose rows form the support of the discrete multivariate distribution.
prob vector of probability weights for the elements of supp.
Symmetry you may help R in calculations if you tell it whether the distribution is non-symmetric (default) or symmetric with respect to a center.
Details

Typical usages are

DiscreteMVDistribution(supp, prob)
DiscreteMVDistribution(supp)

Identical rows are collapsed to unique support values. If prob is missing, all elements in supp are equally weighted.

Value

Object of class "DiscreteMVDistribution"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

DiscreteMVDistribution-class

Examples

# Dirac-measure at (0,0,0)
D1 <- DiscreteMVDistribution(supp = c(0,0,0))
support(D1)

# simple discrete distribution
D2 <- DiscreteMVDistribution(supp = matrix(c(0,1,0,2,2,1,1,0), ncol=2),
prob = c(0.3, 0.2, 0.2, 0.3))
support(D2)
r(D2)(10)

Description

The class of discrete multivariate distributions.

Objects from the Class

Objects can be created by calls of the form new("DiscreteMVDistribution", ...). More frequently they are created via the generating function DiscreteMVDistribution.
Slots

  img  Object of class "rSpace". Image space of the distribution. Usually an object of class "EuclideanSpace".
  param Object of class "OptionalParameter". Optional parameter of the multivariate distribution.
  r  Object of class "function": generates (pseudo-)random numbers
  d  Object of class "OptionalFunction": optional density function
  p  Object of class "OptionalFunction": optional cumulative distribution function
  q  Object of class "OptionalFunction": optional quantile function
  support numeric matrix whose rows form the support of the distribution
  .finSupport logical: (later on to be) used internally to check whether the true support is finite;
    the element in the 1st row and ith column indicates whether the ith marginal distribution has
    a finite left endpoint, and the element in the 2nd row and ith column if it is has a finite right
    endpoint); not yet further used.
  .withArith logical: used internally to issue warnings as to interpretation of arithmetics
  .withSim logical: used internally to issue warnings as to accuracy
  .logExact logical: used internally to flag the case where there are explicit formulae for the log
    version of density, cdf, and quantile function
  .lowerExact logical: used internally to flag the case where there are explicit formulae for the
    lower tail version of cdf and quantile function

Extends

  Class "MultivariateDistribution", directly.
  Class "Distribution", by class "MultivariateDistribution".

Methods

  support signature(object = "DiscreteMVDistribution"): accessor function for slot support.

Author(s)

  Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

  Distribution-class, MultivariateDistribution-class, DiscreteMVDistribution, E-methods

Examples

  (D1 <- new("MultivariateDistribution")) # Dirac measure in (0,0)
  r(D1)(5)

  (D2 <- DiscreteMVDistribution(supp = matrix(c(1:5, rep(3, 5)), ncol=2, byrow=TRUE)))
  support(D2)
  r(D2)(10)
  d(D2)(support(D2))
  p(D2)(lower = c(1,1), upper = c(3,3))
**distrExIntegrate**

Integration of One-Dimensional Functions

**Description**

Numerical integration via `integrate`. In case `integrate` fails a Gauss-Legendre quadrature is performed.

**Usage**

```r
distrExIntegrate(f, lower, upper, subdivisions = 100,
    rel.tol = .Machine$double.eps^0.25,
    abs.tol = rel.tol, stop.on.error = TRUE,
    distr, order, ..., diagnostic = FALSE)

showDiagnostic(x, what, withNonShows = FALSE, ...)
getDiagnostic(x, what, reorganized=TRUE)
## S3 method for class 'DiagnosticClass'
print(x, what, withNonShows = FALSE, xname, ...)
```

**Arguments**

- `f` an R function taking a numeric first argument and returning a numeric vector of the same length. Returning a non-finite element will generate an error.
- `lower` lower limit of integration. Can be `-Inf`.
- `upper` upper limit of integration. Can be `Inf`.
- `subdivisions` the maximum number of subintervals.
- `rel.tol` relative accuracy requested.
- `abs.tol` absolute accuracy requested.
- `stop.on.error` logical. If TRUE (the default) an error stops the function. If false some errors will give a result with a warning in the message component.
- `distr` object of class `UnivariateDistribution`.
- `order` order of Gauss-Legendre quadrature.
- `diagnostic` logical; if TRUE, the return value obtains an attribute "diagnostic" with diagnostic information on the integration, i.e., a list with entries `method` ("integrate" or "GLIntegrate"), `call`, `result` (the complete return value of the method), `args` (the args with which the method was called), and `time` (the time to compute the integral).
In case of integrators: additional arguments to be passed to \( f \). Remember to use argument names not matching those of integrate and GLIntegrate! In case of showDiagnostic, print.Diagnostics: additional arguments to be passed on to print methods called for particular items in the diagnostic list.

**x**

the item for which the diagnostic is to be shown.

**what**

a character vector with all the diagnostic items to be selected/shown. If empty or missing all items are selected/shown.

**withNonShows**

internally we distinguish items which are easily printed (first kind) (numeric, logical, character) and more difficult ones (second kind), e.g., calls, functions, lists. The distinction is made according to the list item name. If withNonShows==TRUE one also attempts to show the selected items of the second kind, otherwise they are not shown (but returned).

**xname**

an optional name for the diagnostic object to be shown.

**reorganized**

should the diagnostic information be reorganized (using internal function .reorganizeDiagnosticList?)

### Details

distrExIntegrate calls integrate. In case integrate returns an error a Gauss-Legendre integration is performed using GLIntegrate. If lower or (and) upper are infinite the GLIntegrateTruncQuantile, respectively the 1-GLIntegrateTruncQuantile quantile of distr is used instead.

distrExIntegrate is called from many places in the distr and robast families of packages. At every such instance, diagnostic information can be collected (setting a corresponding argument diagnostic to TRUE in the calling function. This diagnostic information is originally stored in a tree like list structure of S3 class DiagnosticClass which is then attached as attribute diagnostic to the respective object. It can be inspected and accessed through showDiagnostic and getDiagnostic. More specifically, for any object with attribute diagnostic, showDiagnostic shows the diagnostic collected during integration, and getDiagnostic returns the diagnostic collected during integration. To this end, print.Diagnostics is an S3 method for print for objects of S3 class DiagnosticClass.

### Value

The value of distrExIntegrate is a numeric approximation of the integral. If argument diagnostic==TRUE in distrExIntegrate, the return value has an attribute diagnostic of S3 class DiagnosticClass containing diagnostic information on the integration.

showDiagnostic, getDiagnostic, print.Diagnostics all return (invisibly) a list with the selected items, reorganized by internal function .reorganizeDiagnosticList, respectively, in case of argument reorganized==FALSE, getDiagnostic returns (invisibly) the diagnostic information as is.

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>
References

Based on QUADPACK routines dqags and dqagi by R. Piessens and E. deDoncker-Kapenga, available from Netlib.


See Also

integrate, GLIntegrate, distrExOptions

Examples

fkt <- function(x){x*dchisq(x+1, df = 1)}
integrate(fkt, lower = -1, upper = 3)
GLIntegrate(fkt, lower = -1, upper = 3)
try(integrate(fkt, lower = -1, upper = 5))
distrExIntegrate(fkt, lower = -1, upper = 5)

distrExMASK

Masking of/by other functions in package “distrEx”

Description

Provides information on the (intended) masking of and (non-intended) masking by other other functions in package distrEx

Usage

distrExMASK(library = NULL)

Arguments

library a character vector with path names of R libraries, or NULL. The default value of NULL corresponds to all libraries currently known. If the default is used, the loaded packages are searched before the libraries

Value

no value is returned

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

Examples

distrExMASK()
distrExMOVED

Moved functionality from package "distrEx"

Description

Provides information on moved of functionality from package distrEx.

Usage

distrExMOVED(library = NULL)

Arguments

library a character vector with path names of R libraries, or NULL. The default value
of NULL corresponds to all libraries currently known. If the default is used, the
loaded packages are searched before the libraries

Value

no value is returned

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

Examples

distrExMOVED()

distrExOptions

Function to change the global variables of the package 'distrEx'

Description

With distrExOptions you can inspect and change the global variables of the package distrEx.

Usage

distrExOptions(...)
distrExoptions(...)
getdistrExOption(x)

Arguments

... any options can be defined, using name = value or by passing a list of such
tagged values.
x a character string holding an option name.
Value

distrExOptions() returns a list of the global variables.
distrExOptions(x) returns the global variable x.
getdistrExOption(x) returns the global variable x.
distrExOptions(x=y) sets the value of the global variable x to y.

distrExoptions

For compatibility with spelling in package distr, distrExoptions is just a synonym to distrExOptions.

Global Options

MCIterations: number of Monte-Carlo iterations used for crude Monte-Carlo integration; defaults to 1e5.

GLIntegrateTruncQuantile: If integrate fails and there are infinite integration limits, the function GLIntegrate is called inside of distrExIntegrate with the corresponding quantiles GLIntegrateTruncQuantile respectively, 1 - GLIntegrateTruncQuantile as finite integration limits; defaults to 10×.Machine$double.eps.

GLIntegrateOrder: The order used for the Gauss-Legendre integration inside of distrExIntegrate; defaults to 500.

ElowerTruncQuantile: The lower limit of integration used inside of E which corresponds to the ElowerTruncQuantile-quantile; defaults to 1e-7.

EupperTruncQuantile: The upper limit of integration used inside of E which corresponds to the (1-ElowerTruncQuantile)-quantile; defaults to 1e-7.

ERelativeTolerance: The relative tolerance used inside of E when calling distrExIntegrate; defaults to .Machine$double.eps^0.25.

m1dfLowerTruncQuantile: The lower limit of integration used inside of m1df which corresponds to the m1dfLowerTruncQuantile-quantile; defaults to 0.

m1dfRelativeTolerance: The relative tolerance used inside of m1df when calling distrExIntegrate; defaults to .Machine$double.eps^0.25.

m2dfLowerTruncQuantile: The lower limit of integration used inside of m2df which corresponds to the m2dfLowerTruncQuantile-quantile; defaults to 0.

m2dfRelativeTolerance: The relative tolerance used inside of m2df when calling distrExIntegrate; defaults to .Machine$double.eps^0.25.

nDiscretize: number of support values used for the discretization of objects of class "AbscontDistribution"; defaults to 100.

hSmooth: smoothing parameter to smooth objects of class "DiscreteDistribution". This is done via convolution with the normal distribution Norm(mean = 0, sd = hSmooth); defaults to 0.05.

IQR.fac: for determining sensible integration ranges, we use both quantile and scale based methods; for the scale based method we use the median of the distribution ± IQR.fac× the IQR; defaults to 15.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>
See Also

options.getOption

Examples

distrExOptions()
distrExOptions("ElowerTruncQuantile")
distrExOptions("ElowerTruncQuantile" = 1e-6)
# or
distrExOptions(ElowerTruncQuantile = 1e-6)
getdistrExOption("ElowerTruncQuantile")

---

Generic Function for the Computation of (Conditional) Expectations

Description

Generic function for the computation of (conditional) expectations.

Usage

E(object, fun, cond, ...)

## S4 method for signature 'UnivariateDistribution,missing,missing'
E(object,
   low = NULL, upp = NULL, Nsim = getdistrExOption("MCIterations"), ...)

## S4 method for signature 'UnivariateDistribution,function,missing'
E(object, fun,
   useApply = TRUE, low = NULL, upp = NULL,
   Nsim = getdistrExOption("MCIterations"), ...)

## S4 method for signature 'AbscontDistribution,missing,missing'
E(object, low = NULL, upp = NULL,
   rel.toI = getdistrExOption("ErelativeTolerance"),
   lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
   upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
   iqr.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)

## S4 method for signature 'AbscontDistribution,function,missing'
E(object, fun, useApply = TRUE,
   low = NULL, upp = NULL,
   rel.toI = getdistrExOption("ErelativeTolerance"),
   lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
   upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
   iqr.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)
## S4 method for signature 'UnivarMixingDistribution,missing,missing'
E(object, low = NULL,
    upp = NULL, rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)

## S4 method for signature 'UnivarMixingDistribution,function,missing'
E(object, fun,
    useApply = TRUE, low = NULL, upp = NULL,
    rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)

## S4 method for signature 'UnivarMixingDistribution,missing,ANY'
E(object, cond, low = NULL,
    upp = NULL, rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)

## S4 method for signature 'UnivarMixingDistribution,function,ANY'
E(object, fun, cond,
    useApply = TRUE, low = NULL, upp = NULL,
    rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)

## S4 method for signature 'DiscreteDistribution,function,missing'
E(object, fun, useApply = TRUE,
    low = NULL, upp = NULL, ...)

## S4 method for signature 'AffLinDistribution,missing,missing'
E(object, low = NULL, upp = NULL, 
    ..., diagnostic = FALSE)

## S4 method for signature 'AffLinUnivarLebDecDistribution,missing,missing'
E(object, low = NULL, 
    upp = NULL, ..., diagnostic = FALSE)

## S4 method for signature 'MultivariateDistribution,missing,missing'
E(object,
    Nsim = getdistrExOption("MCIterations"), ...)

## S4 method for signature 'MultivariateDistribution,function,missing'
E(object, fun,
    useApply = TRUE, Nsim = getdistrExOption("MCIterations"), ...)
## S4 method for signature 'DiscreteMVDistribution,missing,missing'
E(object, low = NULL,
    upp = NULL, ...)

## S4 method for signature 'DiscreteMVDistribution,function,missing'
E(object, fun,
    useApply = TRUE, ...)

## S4 method for signature 'AbscontCondDistribution,missing,numeric'
E(object, cond,
    useApply = TRUE, low = NULL, upp = NULL,
    rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE)

## S4 method for signature 'DiscreteCondDistribution,missing,numeric'
E(object, cond,
    useApply = TRUE, low = NULL, upp = NULL, ...)

## S4 method for signature 'UnivariateCondDistribution,function,numeric'
E(object, fun, cond,
    withCond = FALSE, useApply = TRUE, low = NULL, upp = NULL,
    Nsim = getdistrExOption("MCIterations"), ...)

## S4 method for signature 'AbscontCondDistribution,function,numeric'
E(object, fun, cond,
    withCond = FALSE, useApply = TRUE, low = NULL, upp = NULL,
    rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac")
    ..., diagnostic = FALSE)

## S4 method for signature 'DiscreteCondDistribution,function,numeric'
E(object, fun, cond,
    withCond = FALSE, useApply = TRUE, low = NULL, upp = NULL,...)

## S4 method for signature 'UnivarLebDecDistribution,missing,missing'
E(object, low = NULL,
    upp = NULL, rel.tol= getdistrExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
    IQR.fac = getdistrExOption("IQR.fac"), ..., diagnostic = FALSE )

## S4 method for signature 'UnivarLebDecDistribution,function,missing'
E(object, fun,
    useApply = TRUE, low = NULL, upp = NULL,
rel.tol = getdistExOption("ErelativeTolerance"),
lowerTruncQuantile = getdistExOption("ElowerTruncQuantile"),
upperTruncQuantile = getdistExOption("EupperTruncQuantile"),
IQR.fac = getdistExOption("IQR.fac"), ..., diagnostic = FALSE )

# S4 method for signature 'UnivarLebDecDistribution,missing,ANY'
E(object, cond, low = NULL,
    upp = NULL, rel.tol= getdistExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistExOption("EupperTruncQuantile"),
    IQR.fac = getdistExOption("IQR.fac"), ..., diagnostic = FALSE )

# S4 method for signature 'UnivarLebDecDistribution,function,ANY'
E(object, fun, cond,
    useApply = TRUE, low = NULL, upp = NULL,
    rel.tol= getdistExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistExOption("EupperTruncQuantile"),
    IQR.fac = getdistExOption("IQR.fac"), ..., diagnostic = FALSE )

# S4 method for signature 'AcDcLcDistribution,ANY,ANY'
E(object, fun, cond, low = NULL,
    upp = NULL, rel.tol= getdistExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistExOption("EupperTruncQuantile"),
    IQR.fac = getdistExOption("IQR.fac"), ..., diagnostic = FALSE )

# S4 method for signature 'CompoundDistribution,missing,missing'
E(object, low = NULL,
    upp = NULL, ..., diagnostic = FALSE)

# S4 method for signature 'Ariacsine,missing,missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)

# S4 method for signature 'Beta,missing,missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)

# S4 method for signature 'Binom,missing,missing'
E(object, low = NULL, upp = NULL, ...)

# S4 method for signature 'Cauchy,missing,missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)

# S4 method for signature 'Cauchy,functio,n,missing'
E(object, fun, low = NULL, upp = NULL,
    rel.tol = getdistExOption("ErelativeTolerance"),
    lowerTruncQuantile = getdistExOption("ElowerTruncQuantile"),
    upperTruncQuantile = getdistExOption("EupperTruncQuantile"),
    IQR.fac = max(1e4, getdistExOption("IQR.fac")), ..., diagnostic = FALSE)

# S4 method for signature 'Chiql,missing,missing'
E(object, low = NULL, upp = NULL, ...)

# S4 method for signature 'Dirac,missing,missing'
E(object, low = NULL, upp = NULL, ...)

# S4 method for signature 'DExp,missing,missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Exp, missing, missing'
E(object, low = NULL, upp = NULL, ...)
## S4 method for signature 'Fd, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Gammad, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Gammad, function, missing'
E(object, fun, low = NULL, upp = NULL,
   rel.tol = getdistrexOption("ErelativeTolerance"),
   lowerTruncQuantile = getdistrexOption("ELowerTruncQuantile"),
   upperTruncQuantile = getdistrexOption("EUpperTruncQuantile"),
   IQR.fac = max(1e4, getdistrexOption("IQR.fac")), ..., diagnostic = FALSE)
## S4 method for signature 'Geom, missing, missing'
E(object, low = NULL, upp = NULL, ...)

## S4 method for signature 'Hyper, missing, missing'
E(object, low = NULL, upp = NULL, ...)
## S4 method for signature 'Logis, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Lnorm, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Nbinom, missing, missing'
E(object, low = NULL, upp = NULL, ...)
## S4 method for signature 'Norm, missing, missing'
E(object, low = NULL, upp = NULL, ...)
## S4 method for signature 'Pois, missing, missing'
E(object, low = NULL, upp = NULL, ...)
## S4 method for signature 'Unif, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Td, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Weibull, missing, missing'
E(object, low = NULL, upp = NULL, ..., diagnostic = FALSE)
## S4 method for signature 'Weibull, function, missing'
E(object, fun, low = NULL, upp = NULL,
   rel.tol = getdistrexOption("ErelativeTolerance"),
   lowerTruncQuantile = getdistrexOption("ELowerTruncQuantile"),
   upperTruncQuantile = getdistrexOption("EUpperTruncQuantile"),
   IQR.fac = max(1e4, getdistrexOption("IQR.fac")), ..., diagnostic = FALSE)
.qtlIntegrate(object, fun, low = NULL, upp = NULL,
   rel.tol = getdistrexOption("ErelativeTolerance"),
lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
IQR.fac = max(1e4, getdistrExOption("IQR.fac")), …,
.withLeftTail = FALSE, .withRightTail = FALSE, diagnostic = FALSE)

Arguments

object object of class "Distribution"
fun if missing the (conditional) expectation is computed else the (conditional) ex-
pection of fun is computed.
cond if not missing the conditional expectation given cond is computed.
Nsim number of MC simulations used to determine the expectation.
rel.tol relative tolerance for distrExIntegrate.
low lower bound of integration range.
upp upper bound of integration range.
lowerTruncQuantile lower quantile for quantile based integration range.
upperTruncQuantile upper quantile for quantile based integration range.
IQR.fac factor for scale based integration range (i.e.; median of the distribution ±IQR.fac × IQR).
useApply logical: should sapply, respectively apply be used to evaluate fun.
withCond logical: is cond in the argument list of fun.
.withLeftTail logical: should left tail (falling into quantile range [0,0.02]) be computed sepa-
rately to enhance accuracy?
.withRightTail logical: should right tail (falling into quantile range [0.98,1]) be computed sep-
arately to enhance accuracy?
diagnostic logical; if TRUE, the return value obtains an attribute "diagnostic" with diag-
nostic information on the integration, i.e., a list with entries method ("integrate"
 or "GLIntegrate"), call, result (the complete return value of the method),
args (the args with which the method was called), and time (the time to com-
pute the integral).

Details

The precision of the computations can be controlled via certain global options; cf. distrExOptions.
Also note that arguments low and upp should be given as named arguments in order to prevent them
to be matched by arguments fun or cond. Also the result, when arguments low or upp is given, is the
unconditional value of the expectation; no conditioning with respect to low <= object <= upp
is done.

For the Cauchy, the Gamma and Weibull distribution for integration with missing argument cond
but given argument fun, we use integration on [0,1] (i.e. via the respective probability transformation).
This done via helper function .qtlIntegrate, where both arguments .withLeftTail and
.withRightTail are TRUE for the Cauchy and Gamma distributions, and only .withRightTail is
TRUE for the Weibull distribution.
Diagnostics on the involved integrations are available if argument diagnostic is TRUE. Then there is attribute diagnostic attached to the return value, which may be inspected and accessed through showDiagnostic and getDiagnostic.

Value

The (conditional) expectation is computed.

Methods

object = "UnivariateDistribution", fun = "missing", cond = "missing": expectation of univariate distributions using crude Monte-Carlo integration.

object = "AbscontDistribution", fun = "missing", cond = "missing": expectation of absolutely continuous univariate distributions using distrExIntegrate.

object = "DiscreteDistribution", fun = "missing", cond = "missing": expectation of discrete univariate distributions using support and sum.

object = "MultivariateDistribution", fun = "missing", cond = "missing": expectation of multivariate distributions using crude Monte-Carlo integration.

object = "DiscreteMVDistribution", fun = "missing", cond = "missing": expectation of discrete multivariate distributions. The computation is based on support and sum.

object = "UnivariateDistribution", fun = "missing", cond = "missing": expectation of univariate Lebesgue decomposed distributions by separate calculations for discrete and absolutely continuous part.

object = "AffLinDistribution", fun = "missing", cond = "missing": expectation of an affine linear transformation \( aX + b \) as \( aE[X] + b \) for \( x \) either "DiscreteDistribution" or "AbscontDistribution".

object = "AffLinUnivarLebDecDistribution", fun = "missing", cond = "missing": expectation of an affine linear transformation \( aX + b \) as \( aE[X] + b \) for \( x \) either "UnivarLebDecDistribution".

object = "UnivariateDistribution", fun = "function", cond = "missing": expectation of fun under univariate distributions using crude Monte-Carlo integration.

object = "UnivariateDistribution", fun = "function", cond = "missing": expectation of fun under univariate Lebesgue decomposed distributions by separate calculations for discrete and absolutely continuous part.

object = "AbscontDistribution", fun = "function", cond = "missing": expectation of fun under absolutely continuous univariate distributions using distrExIntegrate.

object = "DiscreteDistribution", fun = "function", cond = "missing": expectation of fun under discrete univariate distributions using support and sum.

object = "MultivariateDistribution", fun = "function", cond = "missing": expectation of multivariate distributions using crude Monte-Carlo integration.

object = "DiscreteMVDistribution", fun = "function", cond = "missing": expectation of fun under discrete multivariate distributions. The computation is based on support and sum.

object = "UnivariateCondDistribution", fun = "missing", cond = "numeric": conditional expectation for univariate conditional distributions given cond. The integral is computed using crude Monte-Carlo integration.
object = "AbscontCondDistribution", fun = "missing", cond = "numeric": conditional expectation for absolutely continuous, univariate conditional distributions given cond. The computation is based on distrExIntegrate.

object = "DiscreteCondDistribution", fun = "missing", cond = "numeric": conditional expectation for discrete, univariate conditional distributions given cond. The computation is based on support and sum.

object = "UnivariateCondDistribution", fun = "function", cond = "numeric": conditional expectation of fun under univariate conditional distributions given cond. The integral is computed using crude Monte-Carlo integration.

object = "AbscontCondDistribution", fun = "function", cond = "numeric": conditional expectation of fun under absolutely continuous, univariate conditional distributions given cond. The computation is based on distrExIntegrate.

object = "DiscreteCondDistribution", fun = "function", cond = "numeric": conditional expectation of fun under discrete, univariate conditional distributions given cond. The computation is based on support and sum.

object = "UnivarLebDecDistribution", fun = "missing", cond = "missing": expectation by separate evaluation of expectation of discrete and abs. continuous part and subsequent weighting.

object = "UnivarLebDecDistribution", fun = "function", cond = "missing": expectation by separate evaluation of expectation of discrete and abs. continuous part and subsequent weighting.

object = "UnivarLebDecDistribution", fun = "missing", cond = "ANY": expectation by separate evaluation of expectation of discrete and abs. continuous part and subsequent weighting.

object = "UnivarLebDecDistribution", fun = "function", cond = "ANY": expectation by separate evaluation of expectation of discrete and abs. continuous part and subsequent weighting.

object = "UnivarMixingDistribution", fun = "missing", cond = "missing": expectation is computed component-wise with subsequent weighting acc. to mixCoeff.

object = "UnivarMixingDistribution", fun = "function", cond = "missing": expectation is computed component-wise with subsequent weighting acc. to mixCoeff.

object = "UnivarMixingDistribution", fun = "missing", cond = "ANY": expectation is computed component-wise with subsequent weighting acc. to mixCoeff.

object = "UnivarMixingDistribution", fun = "function", cond = "ANY": expectation is computed component-wise with subsequent weighting acc. to mixCoeff.

object = "AcDcLcDistribution", fun = "ANY", cond = "ANY": expectation by first coercing to class "UnivarLebDecDistribution" and using the corresponding method.

object = "CompoundDistribution", fun = "missing", cond = "missing": if we are in i.i.d. situation (i.e., slot SummandsDistri of class UnivariateDistribution) the formula $E[N]E[S]$ for $N$ the frequency distribution and $S$ the summand distribution; else we coerce to "UnivarLebDecDistribution".

object = "Arcsine", fun = "missing", cond = "missing": exact evaluation using explicit expressions.

object = "Beta", fun = "missing", cond = "missing": for noncentrality 0 exact evaluation using explicit expressions.

object = "Binom", fun = "missing", cond = "missing": exact evaluation using explicit expressions.

object = "Cauchy", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Chisq", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Dirac", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "DExp", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Exp", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Fd", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Gammad", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Gammad", fun = "function", cond = "missing": use integration over the quantile range for numerical integration via helper function.qtlIntegrate.
object = "Geom", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Hyper", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Logis", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Lnorm", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Nbinom", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Norm", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Pois", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Unif", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Td", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Weibull", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Weibull", fun = "function", cond = "missing": use integration over the quantile range for numerical integration via helper function.qtlIntegrate.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de> and Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

See Also
distrExIntegrate.m1df,m2df,Distribution-class

Examples
# mean of Exp(1) distribution
E <- Exp()

E(E) ## uses explicit terms
E(as(E,"AbscontDistribution")) ## uses numerical integration
E(as(E,"UnivariateDistribution")) ## uses simulations
E(E, fun = function(x)(2*x^2)) ## uses simulations
EmpiricalMVDistribution

Generating function for multivariate discrete distribution

Description

Generates an object of class "DiscreteMVDistribution".
Usage

`EmpiricalMVDistribution(data, Symmetry = NoSymmetry())`

Arguments

data numeric matrix with data where the rows are interpreted as observations.

Symmetry you may help \( \mathcal{R} \) in calculations if you tell it whether the distribution is non-symmetric (default) or symmetric with respect to a center.

Details

The function is a simple utility function providing a wrapper to the generating function `DiscreteDistribution`. Typical usages are

`EmpiricalMVDistribution(data)`

Identical rows are collapsed to unique support values. If `prob` is missing, all elements in `supp` are equally weighted.

Value

Object of class "DiscreteMVDistribution"

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

`DiscreteMVDistribution`

Examples

```r
## generate some data
X <- matrix(rnorm(50), ncol = 5)

## empirical distribution of X
D1 <- EmpiricalMVDistribution(data = X)
support(D1)
r(D1)(10)
```
EuclCondition

Generating function for EuclCondition-class

Description
Generates an object of class "EuclCondition".

Usage
EuclCondition(dimension)

Arguments
dimension positive integer: dimension of the Euclidean space

Value
Object of class "EuclCondition"

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>

See Also
EuclCondition-class

Examples
EuclCondition(dimension = 3)

## The function is currently defined as
function(dimension){
  new("EuclCondition", Range = EuclideanSpace(dimension = dimension))
}

EuclCondition-class Conditioning by an Euclidean space.

Description
Conditioning by an Euclidean space.

Objects from the Class
Objects can be created by calls of the form new("EuclCondition", ...). More frequently they are created via the generating function EuclCondition.
GLIntegrate

Slots

Range  Object of class "EuclideanSpace".
name    Object of class "character": name of condition.

Extends

Class "Condition", directly.

Methods

Range signature(object = "EuclCondition") accessor function for slot Range.
show signature(object = "EuclCondition")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

Condition-class, EuclCondition

Examples

new("EuclCondition")

---

GLIntegrate  Gauss-Legendre Quadrature

Description

Gauss-Legendre quadrature over a finite interval.

Usage

GLIntegrate(f, lower, upper, order = 500, ...)

Arguments

f          an R function taking a numeric first argument and returning a numeric vector of the same length. Returning a non-finite element will generate an error.
lower      finite lower limit of integration.
upper      finite upper limit of integration.
order      order of Gauss-Legendre quadrature.
...        additional arguments to be passed to f. Remember to use argument names not matching those of GLIntegrate!
Details

In case order = 100, 500, 1000 saved abscissas and weights are used. Otherwise the corresponding abscissas and weights are computed using the algorithm given in Section 4.5 of Press et al. (1992).

Value

Estimate of the integral.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

integrate, distrExIntegrate

Examples

integrate(dnorm, -1.96, 1.96)
GLIntegrate(dnorm, -1.96, 1.96)

HellingerDist

Generic function for the computation of the Hellinger distance of two distributions

Description

Generic function for the computation of the Hellinger distance \( d_h \) of two distributions \( P \) and \( Q \) which may be defined for an arbitrary sample space \((\Omega, \mathcal{A})\). The Hellinger distance is defined as

\[
d_h(P, Q) = \frac{1}{2} \int |\sqrt{dP} - \sqrt{dQ}|^2
\]

where \( \sqrt{dP} \), respectively \( \sqrt{dQ} \) denotes the square root of the densities.
Usage

HellingerDist(e1, e2, ...)

Usage for signature 'AbscontDistribution,AbscontDistribution'

HellingerDist(e1,e2,
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistroption("TruncQuantile"),
  IQR.fac = 15, ..., diagnostic = FALSE)

Usage for signature 'AbscontDistribution,DiscreteDistribution'

HellingerDist(e1,e2, ...)

Usage for signature 'DiscreteDistribution,AbscontDistribution'

HellingerDist(e1,e2, ...)

Usage for signature 'DiscreteDistribution,DiscreteDistribution'

HellingerDist(e1,e2, ...)

Usage for signature 'numeric,DiscreteDistribution'

HellingerDist(e1, e2, ...)

Usage for signature 'DiscreteDistribution,numeric'

HellingerDist(e1, e2, ...)

Usage for signature 'numeric,AbscontDistribution'

HellingerDist(e1, e2, asis.smooth.discretize = "discretize",
  n дискрет.= getdistrexOption("nDiscretize"), low. дискрет.= getLow(e2),
  up. дискрет.= getUp(e2), h. smooth = getdistrexOption("hSmooth"),
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistroption("TruncQuantile"),
  IQR.fac = 15, ..., diagnostic = FALSE)

Usage for signature 'AbscontDistribution,numeric'

HellingerDist(e1, e2, asis.smooth.discretize = "discretize",
  n дискрет.= getdistrexOption("nDiscretize"), low. дискрет.= getLow(e1),
  up. дискрет.= getUp(e1), h. smooth = getdistrexOption("hSmooth"),
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistroption("TruncQuantile"),
  IQR.fac = 15, ..., diagnostic = FALSE)

Usage for signature 'AcDcLcDistribution,AcDcLcDistribution'

HellingerDist(e1,e2,
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistroption("TruncQuantile"),
  IQR.fac = 15, ..., diagnostic = FALSE)

Arguments

e1 object of class "Distribution" or class "numeric"
e2 object of class "Distribution" or class "numeric"
asis.smooth.discretize possible methods are "asis", "smooth" and "discretize". Default is "discretize".
n дискрет. if asis.smooth.discretize is equal to "discretize" one has to specify the number of lattice points used to discretize the abs. cont. distribution.
low. дискрет. if asis.smooth.discretize is equal to "discretize" one has to specify the lower end point of the lattice used to discretize the abs. cont. distribution.
If `asis.smooth.discretize` is equal to "discretize" one has to specify the upper end point of the lattice used to discretize the abs. cont. distribution.

If `asis.smooth.discretize` is equal to "smooth" – i.e., the empirical distribution of the provided data should be smoothed – one has to specify this parameter.

The parameter `up.discr` specifies the upper end point of the lattice used to discretize the abs. cont. distribution.

The parameter `h.smooth` specifies the smoothing parameter for the empirical distribution of the provided data.

The parameter `rel.tol` specifies the relative accuracy requested in integration.

The parameter `TruncQuantile` specifies the quantile based integration bounds (see details).

The parameter `IQR.fac` specifies the factor for the scale based integration bounds (see details).

Further arguments are passed to particular methods – (in package `distrEx`: just used for distributions with a.c. parts, where it is used to pass on arguments to `distrExIntegrate`).

The parameter `diagnostic` is a logical; if `TRUE`, the return value obtains an attribute "diagnostic" with diagnostic information on the integration, i.e., a list with entries method ("integrate" or "GLIntegrate"), call, result (the complete return value of the method), args (the args with which the method was called), and time (the time to compute the integral).

Details

For distances between absolutely continuous distributions, we use numerical integration; to determine sensible bounds we proceed as follows: by means of `min(getLow(e1,eps=TruncQuantile),getLow(e2,eps=TruncQuantile))` and `max(getUp(e1,eps=TruncQuantile),getUp(e2,eps=TruncQuantile))` we determine quantile based bounds `c(low,up,0)`, and by means of `s1 <- max(IQR(e1),IQR(e2)); m1 <- median(e1); m2 <- median(e2) and low.1 <- min(m1,m2)-s1*IQR.fac, up.1 <- max(m1,m2)+s1*IQR.fac` we determine scale based bounds; these are combined by `low <- max(low,low.1), up <- max(up,up1)`. In case we want to compute the Hellinger distance between (empirical) data and an abs. cont. distribution, we can specify the parameter `asis.smooth.discretize` to avoid trivial distances (distance = 1).

Using `asis.smooth.discretize = "discretize"`, which is the default, leads to a discretization of the provided abs. cont. distribution and the distance is computed between the provided data and the discretized distribution.

Using `asis.smooth.discretize = "smooth"` causes smoothing of the empirical distribution of the provided data. This is, the empirical data is convoluted with the normal distribution `Norm(mean = 0, sd = h.smooth)` which leads to an abs. cont. distribution. Afterwards the distance between the smoothed empirical distribution and the provided abs. cont. distribution is computed.

Diagnostics on the involved integrations are available if argument `diagnostic` is `TRUE`. Then there is attribute `diagnostic` attached to the return value, which may be inspected and accessed through `showDiagnostic` and `getDiagnostic`.

Value

Hellinger distance of `e1` and `e2`

Methods

`e1 = "AbscontDistribution", e2 = "AbscontDistribution"`: Hellinger distance of two absolutely continuous univariate distributions which is computed using `distrExIntegrate`. 

---

**HellingerDist**

**up.discr** if `asis.smooth.discretize` is equal to "discretize" one has to specify the upper end point of the lattice used to discretize the abs. cont. distribution.

**h.smooth** if `asis.smooth.discretize` is equal to "smooth" – i.e., the empirical distribution of the provided data should be smoothed – one has to specify this parameter.

**rel.tol** relative accuracy requested in integration

**TruncQuantile** Quantile the quantile based integration bounds (see details)

**IQR.fac** Factor for the scale based integration bounds (see details)

**...** further arguments to be used in particular methods – (in package `distrEx`: just used for distributions with a.c. parts, where it is used to pass on arguments to `distrExIntegrate`).

**diagnostic** logical; if `TRUE`, the return value obtains an attribute "diagnostic" with diagnostic information on the integration, i.e., a list with entries method ("integrate" or "GLIntegrate"), call, result (the complete return value of the method), args (the args with which the method was called), and time (the time to compute the integral).
e1 = "AbscontDistribution", e2 = "DiscreteDistribution": Hellinger distance of absolutely continuous and discrete univariate distributions (are mutually singular; i.e., have distance =1).

e1 = "DiscreteDistribution", e2 = "DiscreteDistribution": Hellinger distance of two discrete univariate distributions which is computed using support and sum.

e1 = "DiscreteDistribution", e2 = "AbscontDistribution": Hellinger distance of discrete and absolutely continuous univariate distributions (are mutually singular; i.e., have distance =1).

e1 = "numeric", e2 = "DiscreteDistribution": Hellinger distance between (empirical) data and a discrete distribution.

e1 = "DiscreteDistribution", e2 = "numeric": Hellinger distance between (empirical) data and a discrete distribution.

e1 = "numeric", e2 = "AbscontDistribution": Hellinger distance between (empirical) data and an abs. cont. distribution.

e1 = "AbscontDistribution", e1 = "numeric": Hellinger distance between (empirical) data and an abs. cont. distribution.

e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution": Hellinger distance of mixed discrete and absolutely continuous univariate distributions.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also
distrExIntegrate, ContaminationSize, TotalVarDist, KolmogorovDist, Distribution-class

Examples
HellingerDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
mixCoeff=c(0.2,0.8)))
HellingerDist(Norm(), Td(10))
HellingerDist(Norm(mean = 50, sd = sqrt(25)), Binom(size = 100)) # mutually singular
HellingerDist(Pois(10), Binom(size = 20))

x <- rnorm(100)
HellingerDist(Norm(), x)
HellingerDist(x, Norm(), asis.smooth.discretize = "smooth")

y <- rbinom(50, size = 20, prob = 0.5)-10/sqrt(5)
HellingerDist(y, Norm())
HellingerDist(y, Norm(), asis.smooth.discretize = "smooth")

HellingerDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5))
Description

Generic function for the computation of the Kolmogorov distance \( d_\kappa \) of two distributions \( P \) and \( Q \) where the distributions are defined on a finite-dimensional Euclidean space \( (\mathbb{R}^m, \mathcal{B}^m) \) with \( \mathcal{B}^m \) the Borel-\( \sigma \)-algebra on \( \mathbb{R}^m \). The Kolmogorov distance is defined as

\[
d_\kappa(P, Q) = \sup \{ |P(\{ y \in \mathbb{R}^m | y \leq x \}) - Q(\{ y \in \mathbb{R}^m | y \leq x \})| | x \in \mathbb{R}^m \} \]

where \( \leq \) is coordinatewise on \( \mathbb{R}^m \).

Usage

KolmogorovDist(e1, e2, ...)

## S4 method for signature 'AbscontDistribution,AbscontDistribution'
KolmogorovDist(e1,e2)

## S4 method for signature 'AbscontDistribution,DiscreteDistribution'
KolmogorovDist(e1,e2)

## S4 method for signature 'DiscreteDistribution,AbscontDistribution'
KolmogorovDist(e1,e2)

## S4 method for signature 'DiscreteDistribution,DiscreteDistribution'
KolmogorovDist(e1,e2)

## S4 method for signature 'numeric,UnivariateDistribution'
KolmogorovDist(e1, e2)

## S4 method for signature 'UnivariateDistribution,numeric'
KolmogorovDist(e1, e2)

## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
KolmogorovDist(e1, e2)

Arguments

e1 object of class "Distribution" or class "numeric"
e2 object of class "Distribution" or class "numeric"
... further arguments to be used in particular methods (not in package distrEx)

Value

Kolmogorov distance of e1 and e2

Methods

e1 = "AbscontDistribution", e2 = "AbscontDistribution": Kolmogorov distance of two absolutely continuous univariate distributions which is computed using a union of a (pseudo-)random and a deterministic grid.
e1 = "DiscreteDistribution", e2 = "DiscreteDistribution": Kolmogorov distance of two discrete univariate distributions. The distance is attained at some point of the union of the supports of e1 and e2.

e1 = "AbscontDistribution", e2 = "DiscreteDistribution": Kolmogorov distance of absolutely continuous and discrete univariate distributions. It is computed using a union of a (pseudo-)random and a deterministic grid in combination with the support of e2.

e1 = "DiscreteDistribution", e2 = "AbscontDistribution": Kolmogorov distance of discrete and absolutely continuous univariate distributions. It is computed using a union of a (pseudo-)random and a deterministic grid in combination with the support of e1.

e1 = "numeric", e2 = "UnivariateDistribution": Kolmogorov distance between (empirical) data and a univariate distribution. The computation is based on ks.test.

e1 = "UnivariateDistribution", e2 = "numeric": Kolmogorov distance between (empirical) data and a univariate distribution. The computation is based on ks.test.

e1 = "AcDcLeDistribution", e2 = "AcDcLeDistribution": Kolmogorov distance of mixed discrete and absolutely continuous univariate distributions. It is computed using a union of the discrete part, a (pseudo-)random and a deterministic grid in combination with the support of e1.

Author(s)
Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

See Also
ContaminationSize, TotalVarDist, HellingerDist, Distribution-class

Examples
KolmogorovDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
mixCoeff=c(0.2,0.8)))
KolmogorovDist(Norm(), Td(10))
KolmogorovDist(Norm(mean = 50, sd = sqrt(25)), Binom(size = 100))
KolmogorovDist(Pois(10), Binom(size = 20))
KolmogorovDist(Norm(), rnorm(100))
KolmogorovDist((rbinom(50, size = 20, prob = 0.5)-10)/sqrt(5), Norm())
KolmogorovDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5))
Generic Function for Testing the Support of a Distribution

Description

The function tests if \( x \) lies in the support of the distribution object.

Usage

```r
## S4 method for signature 'DiscreteMVDistribution,numeric'
liesInSupport(object, x, checkFin = FALSE)
## S4 method for signature 'DiscreteMVDistribution,numeric'
liesInSupport(object, x, checkFin = FALSE)
```

Arguments

- `object` object of class "Distribution"
- `x` numeric vector or matrix
- `checkFin` logical: in case FALSE, we simply check whether \( x \) lies exactly in the numerical support (of finitely many support points); later on we might try to mimick the univariate case more closely in case TRUE, but so far this is not yet used.

Value

logical vector

Methods

- `object = "DiscreteMVDistribution", x = "numeric"`: does \( x \) lie in the support of `object`.
- `object = "DiscreteMVDistribution", x = "matrix"`: does \( x \) lie in the support of `object`.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

- `Distribution-class`

Examples

```r
M <- matrix(rpois(30, lambda = 10), ncol = 3)
D1 <- DiscreteMVDistribution(M)
M1 <- rbind(r(D1)(10), matrix(rpois(30, lam = 10), ncol = 3))
liesInSupport(D1, M1)
```
**LMCondDistribution**

*Generating function for the conditional distribution of a linear regression model.*

**Description**

Generates an object of class "AbscontCondDistribution" which is the conditional distribution of a linear regression model (given the regressor).

**Usage**

```r
LMCondDistribution( Error = Norm(), theta = 0, intercept = 0, scale = 1 )
```

**Arguments**

- **Error**: Object of class "AbscontDistribution": error distribution.
- **theta**: numeric vector: regression parameter.
- **intercept**: real number: intercept parameter.
- **scale**: positive real number: scale parameter.

**Value**

Object of class "AbscontCondDistribution"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

-AbscontCondDistribution-class, E-methods

**Examples**

```r
# normal error distribution
(D1 <- LMCondDistribution(theta = 1)) # corresponds to Norm(cond, 1)
plot(D1)
r(D1)
d(D1)
p(D1)
q(D1)
# in RStudio or Jupyter IRKernel, use q.l(.)(.) instead of q(.)(.)
param(D1)
cond(D1)
d(D1)(0, cond = 1)
d(Norm(mean=1))(0)
```
**LMPParameter**

Generating function for LMPParameter-class

---

**Description**

Generates an object of class "LMPParameter".

**Usage**

LMPParameter(theta = 0, intercept = 0, scale = 1)

**Arguments**

- **theta** numeric vector: regression parameter (default = 0).
- **intercept** real number: intercept parameter (default = 0).
- **scale** positive real number: scale parameter (default = 1).

**Value**

Object of class "LMPParameter"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

LMPParameter-class

**Examples**

LMPParameter(theta = c(1,1), intercept = 2, scale = 0.5)

## The function is currently defined as

```
function(theta = 0, intercept = 0, scale = 1){
    new("LMPParameter", theta = theta, intercept = intercept, scale = 1)
}
```
LMPParameter-class  
Parameter of a linear regression model

Description

Parameter of a linear regression model

\[ y = \mu + x^T \theta + \sigma u \]

with intercept \( \mu \), regression parameter \( \theta \) and error scale \( \sigma \).

Objects from the Class

Objects can be created by calls of the form `new("LMPParameter", ...)`. More frequently they are created via the generating function `LMPParameter`.

Slots

- `theta` numeric vector: regression parameter.
- `intercept` real number: intercept parameter.
- `scale` positive real number: scale parameter.
- `name` character vector: the default name is “parameter of a linear regression model”.

Extends

Class "Parameter", directly.
Class "OptionalParameter", by class "Parameter".

Methods

- `show` signature(object = "LMPParameter")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

  Parameter-class, LMPParameter

Examples

  `new("LMPParameter")`
Generic Function for the Computation of Clipped First Moments

Description

Generic function for the computation of clipped first moments. The moments are clipped at upper.

Usage

m1df(object, upper, ...)  # S4 method for signature 'AbscontDistribution'
object, upper,
lowerTruncQuantile = getdistrExOption("m1dfLowerTruncQuantile"),
rel.tol = getdistrExOption("m1dfRelativeTolerance"), ...)

Arguments

object object of class "Distribution"
upper clipping bound
rel.tol relative tolerance for distreXIntegrate.
lowerTruncQuantile lower quantile for quantile based integration range.
... additional arguments to E

Details

The precision of the computations can be controlled via certain global options; cf. distreXOptions.

Value

The first moment of object clipped at upper is computed.

Methods

object = "UnivariateDistribution": uses call E(object, upp=upper, ...).
object = "AbscontDistribution": clipped first moment for absolutely continuous univariate distributions which is computed using integrate.
object = "LatticeDistribution": clipped first moment for discrete univariate distributions which is computed using support and sum.
object = "AffLinDistribution": clipped first moment for affine linear distributions which is computed on basis of slot X0.
object = "Binom": clipped first moment for Binomial distributions which is computed using pbinom.
object = "Pois": clipped first moment for Poisson distributions which is computed using ppois.
object = "Norm": clipped first moment for normal distributions which is computed using dnorm and pnorm.
object = "Exp": clipped first moment for exponential distributions which is computed using pexp.
object = "Chisq": clipped first moment for \( \chi^2 \) distributions which is computed using pchisq.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

distrExIntegrate, m2df, E

Examples

# standard normal distribution
N1 <- Norm()
m1df(N1, 0)

# Poisson distribution
P1 <- Pois(lambda=2)
m1df(P1, 3)
m1df(P1, 3, fun = function(x)sin(x))

# absolutely continuous distribution
D1 <- Norm() + Exp() # convolution
m1df(D1, 2)
m1df(D1, Inf)
E(D1)

m2df

Generic function for the computation of clipped second moments

Description

Generic function for the computation of clipped second moments. The moments are clipped at upper.

Usage

m2df(object, upper, ...)
## S4 method for signature 'AbscontDistribution'
m2df(object, upper,
     lowerTruncQuantile = getdistrExOption("m2dfLowerTruncQuantile"),
     rel.tol = getdistrExOption("m2dfRelativeTolerance"), ...)
**Arguments**

- `object` object of class "Distribution"
- `upper` clipping bound
- `rel.tol` relative tolerance for `distrExIntegrate`
- `lowerTruncQuantile` lower quantile for quantile based integration range.
- `...` additional arguments to `E`

**Details**

The precision of the computations can be controlled via certain global options; cf. `distrExOptions`.

**Value**

The second moment of `object` clipped at `upper` is computed.

**Methods**

- `object = "UnivariateDistribution"`: uses call `E(object, upp=upper, fun = function, ...)`. 
- `object = "AbscontDistribution"`: clipped second moment for absolutely continuous univariate distributions which is computed using `integrate`. 
- `object = "LatticeDistribution"`: clipped second moment for discrete univariate distributions which is computed using `support` and `sum`. 
- `object = "AffLinDistribution"`: clipped second moment for affine linear distributions which is computed on basis of slot `X0`. 
- `object = "Binom"`: clipped second moment for Binomial distributions which is computed using `pbinom`. 
- `object = "Pois"`: clipped second moment for Poisson distributions which is computed using `ppois`. 
- `object = "Norm"`: clipped second moment for normal distributions which is computed using `dnorm` and `pnorm`. 
- `object = "Exp"`: clipped second moment for exponential distributions which is computed using `pexp`. 
- `object = "Chisq"`: clipped second moment for $\chi^2$ distributions which is computed using `pchisq`.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

`m2df-methods, E-methods`
Examples

    # standard normal distribution
    N1 <- Norm()
    m2df(N1, 0)

    # Poisson distribution
    P1 <- Pois(lambda=2)
    m2df(P1, 3)
    m2df(P1, 3, fun = function(x)sin(x))

    # absolutely continuous distribution
    D1 <- Norm() + Exp() # convolution
    m2df(D1, 2)
    m2df(D1, Inf)
    E(D1, function(x)(x^2))

make01  Centering and Standardization of Univariate Distributions

Description

The function make01 produces a new centered and standardized univariate distribution.

Usage

make01(x)

Arguments

x    an object of class "UnivariateDistribution"

Details

Thanks to the functionals provided in this package, the code is a one-liner: (x-E(x))/sd(x).

Value

Object of class "UnivariateDistribution" with expectation 0 and variance 1.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

See Also

E, Var
Examples

```r
X <- sin(exp(2*log(abs(Norm())))) ## something weird
X01 <- make01(X)
print(X01)
plot(X01)
sd(X01); E(X01)
```

**MultivariateDistribution-class**

**Multivariate Distributions**

Description

The class of multivariate distributions. One has at least to specify the image space of the distribution and a function generating (pseudo-)random numbers. The slot `q` is usually filled with `NULL` for dimensions $> 1$.

Objects from the Class

Objects can be created by calls of the form `new("MultivariateDistribution", ...)`.  

Slots

- `img` Object of class "rSpace". Image space of the distribution. Usually an object of class "EuclideanSpace".
- `param` Object of class "OptionalParameter". Optional parameter of the multivariate distribution.
- `r` Object of class "function": generates (pseudo-)random numbers
- `d` Object of class "OptionalFunction": optional density function
- `p` Object of class "OptionalFunction": optional cumulative distribution function
- `q` Object of class "OptionalFunction": optional quantile function
- `withArith` logical: used internally to issue warnings as to interpretation of arithmetics
- `withSim` logical: used internally to issue warnings as to accuracy
- `logExact` logical: used internally to flag the case where there are explicit formulae for the log version of density, cdf, and quantile function
- `lowerExact` logical: used internally to flag the case where there are explicit formulae for the lower tail version of cdf and quantile function
- `Symmetry` object of class "DistributionSymmetry"; used internally to avoid unnecessary calculations.

Extends

Class "Distribution", directly.
Methods

    show signature(object = "MultivariateDistribution")
    plot signature(object = "MultivariateDistribution"): not yet implemented.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

    Distribution-class

Examples

    # Dirac-measure in (0,0)
    new("MultivariateDistribution")

OAsymTotalVarDist

Generic function for the computation of (minimal) asymmetric total variation distance of two distributions

Description

Generic function for the computation of (minimal) asymmetric total variation distance $d^*_{v}$ of two distributions $P$ and $Q$ where the distributions may be defined for an arbitrary sample space $(\Omega, \mathcal{A})$. This distance is defined as

$$d^*_{v}(P, Q) = \min_{c} \int |dQ - cdP|$$

Usage

OAsymTotalVarDist(e1, e2, ...)
## S4 method for signature 'AbscontDistribution,AbscontDistribution'
OAsymTotalVarDist(e1,e2,    
    rel.tol = .Machine$double.eps^0.3, Ngrid = 10000, 
    TruncQuantile = getdistrOption("TruncQuantile"),  
    IQR.fac = 15, ..., diagnostic = FALSE)

## S4 method for signature 'AbscontDistribution,DiscreteDistribution'
OAsymTotalVarDist(e1,e2, ...)

## S4 method for signature 'DiscreteDistribution,AbscontDistribution'
OAsymTotalVarDist(e1,e2, ...)

## S4 method for signature 'DiscreteDistribution,DiscreteDistribution'
OAsymTotalVarDist(e1,e2, ...)

## S4 method for signature 'numeric,DiscreteDistribution'
OAsymTotalVarDist(e1, e2, ...)

## S4 method for signature 'DiscreteDistribution,numeric'
OAsymTotalVarDist(e1, e2, ...)
## Arguments

e1, e2  
object of class "Distribution" or "numeric"

asis.smooth.discretize  
possible methods are "asis", "smooth" and "discretize". Default is "discretize".

n.discr  
if asis.smooth.discretize is equal to "discretize" one has to specify the number of lattice points used to discretize the abs. cont. distribution.

low.dscr  
if asis.smooth.discretize is equal to "discretize" one has to specify the lower end point of the lattice used to discretize the abs. cont. distribution.

up.dscr  
if asis.smooth.discretize is equal to "discretize" one has to specify the upper end point of the lattice used to discretize the abs. cont. distribution.

h.smooth  
if asis.smooth.discretize is equal to "smooth" – i.e., the empirical distribution of the provided data should be smoothed – one has to specify this parameter.

rel.tol  
relative tolerance for distrExIntegrate and uniroot

Ngrid  
How many grid points are to be evaluated to determine the range of the likelihood ratio?;

TruncQuantile  
Quantile the quantile based integration bounds (see details)

IQR.fac  
Factor for the scale based integration bounds (see details)

...  
further arguments to be used in particular methods – (in package distrEx: just used for distributions with a.c. parts, where it is used to pass on arguments to distrExIntegrate).

diagnostic  
logical; if TRUE, the return value obtains an attribute "diagnostic" with diagnostic information on the integration, i.e., a list with entries method ("integrate"
or "GLIntegrate"), call, result (the complete return value of the method),
args (the args with which the method was called), and time (the time to com-
pute the integral).

Details

For distances between absolutely continuous distributions, we use numerical integration; to deter-
mine sensible bounds we proceed as follows: by means of $\min(\text{getLow}(e1, \text{eps} = \text{TruncQuantile}), \text{getLow}(e2, \text{eps} = \text{TruncQuantile}))$, $\max(\text{getUp}(e1, \text{eps} = \text{TruncQuantile}), \text{getUp}(e2, \text{eps} = \text{TruncQuantile}))$ we determine quantile
based bounds $c(\text{low}, 0, \text{up}, 0)$, and by means of $s1 \leftarrow \max(\text{IQR}(e1), \text{IQR}(e2)); m1 < \text{median}(e1)$;
$m2 < \text{median}(e2)$ and $\text{low}.1 \leftarrow \min(m1, m2) - s1*\text{IQR}.\text{fac}, \text{up}.1 \leftarrow \max(m1, m2) + s1*\text{IQR}.\text{fac}$
we determine scale based bounds; these are combined by $\text{low} \leftarrow \max(\text{low}, 0, \text{low}.1), \text{up} \leftarrow \max(\text{up}, 0, \text{up}.1)$.

Again in the absolutely continuous case, to determine the range of the likelihood ratio, we evaluate
this ratio on a grid constructed as follows: $\text{x.range} \leftarrow \text{c(seq(\text{low}, \text{up}, \text{length}=\text{Ngrid}/3)}, q.1(\text{e1}(\text{...}))$.

Finally, for both discrete and absolutely continuous case, we clip this ratio downwards by $1e^{-10}$
and upwards by $1e10$

In case we want to compute the total variation distance between (empirical) data and an abs. cont.
distribution, we can specify the parameter asis.smooth.discretize to avoid trivial distances
(distance = 1).

Using asis.smooth.discretize = "discretize", which is the default, leads to a discretization
of the provided abs. cont. distribution and the distance is computed between the provided data and
the discretized distribution.

Using asis.smooth.discretize = "smooth" causes smoothing of the empirical distribution of
the provided data. This is, the empirical data is convoluted with the normal distribution $\text{Norm}(\text{mean} = 0, \text{sd} = h.\text{smooth})$
which leads to an abs. cont. distribution. Afterwards the distance between the smoothed empirical
distribution and the provided abs. cont. distribution is computed.

Diagnostics on the involved integrations are available if argument diagnostic is TRUE. Then there
is attribute diagnostic attached to the return value, which may be inspected and accessed through
showDiagnostic and getDiagnostic.

Value

OAsymmetric Total variation distance of e1 and e2

Methods

e1 = "AbscontDistribution", e2 = "AbscontDistribution": total variation distance of two abso-
lutely continuous univariate distributions which is computed using distrExIntegrate.

e1 = "AbscontDistribution", e2 = "DiscreteDistribution": total variation distance of absolutely
continuous and discrete univariate distributions (are mutually singular; i.e., have distance =1).

e1 = "DiscreteDistribution", e2 = "DiscreteDistribution": total variation distance of two dis-
crete univariate distributions which is computed using support and sum.

e1 = "DiscreteDistribution", e2 = "AbscontDistribution": total variation distance of discrete and
absolutely continuous univariate distributions (are mutually singular; i.e., have distance =1).

e1 = "numeric", e2 = "DiscreteDistribution": Total variation distance between (empirical) data
and a discrete distribution.
e1 = "DiscreteDistribution", e2 = "numeric": Total variation distance between (empirical) data and a discrete distribution.

e1 = "numeric", e2 = "AbscontDistribution": Total variation distance between (empirical) data and an abs. cont. distribution.

e1 = "AbscontDistribution", e1 = "numeric": Total variation distance between (empirical) data and an abs. cont. distribution.

e1 = "AcDeLcDistribution", e2 = "AcDeLcDistribution": Total variation distance of mixed discrete and absolutely continuous univariate distributions.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

References

to be filled; Agostinelli, C and Ruckdeschel, P. (2009): A simultaneous inlier and outlier model by asymmetric total variation distance.

See Also

TotalVarDist-methods, ContaminationSize, KolmogorovDist, HellingerDist, Distribution-class

Examples

OAsymTotalVarDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
mixCoeff=c(0.2,0.8)))
OAsymTotalVarDist(Norm(), Td(10))
OAsymTotalVarDist(Norm(mean = 50, sd = sqrt(25)), Binom(size = 100)) # mutually singular
OAsymTotalVarDist(Pois(10), Binom(size = 20))

x <- rnorm(100)
OAsymTotalVarDist(Norm(), x)
OAsymTotalVarDist(x, Norm(), asis.smooth.discretize = "smooth")

y <- (rbinom(50, size = 20, prob = 0.5)-10)/sqrt(5)
OAsymTotalVarDist(y, Norm())
OAsymTotalVarDist(y, Norm(), asis.smooth.discretize = "smooth")

OAsymTotalVarDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5))

plot-methods

Methods for Function plot in Package ‘distrEx’

Description

plot-methods
Usage

plot(x, y, ...)
## S4 method for signature 'UnivariateCondDistribution,missing'
plot(x, y, ...)
## S4 method for signature 'MultivariateDistribution,missing'
plot(x, y, ...)

Arguments

x object of class "UnivariateCondDistribution" or class "MultivariateDistribution":
distribution(s) which should be plotted
y missing
... additional arguments

Details

upto now only warnings are issued that the corresponding method is not yet implemented;

---

PrognCondDistribution  Generating function for PrognCondDistribution-class

Description

Generates an object of class "PrognCondDistribution".

Usage

PrognCondDistribution(Regr, Error,
  rel.tol= getdistrExOption("ErelativeTolerance"),
  lowerTruncQuantile = getdistrExOption("ElowerTruncQuantile"),
  upperTruncQuantile = getdistrExOption("EupperTruncQuantile"),
  IQR.fac = getdistrExOption("IQR.fac"))

Arguments

Regr object of class AbscontDistribution; the distribution of X.
Error object of class AbscontDistribution; the distribution of eps.
rel.tol relative tolerance for distrExIntegrate.
lowerTruncQuantile lower quantile for quantile based integration range.
upperTruncQuantile upper quantile for quantile based integration range.
IQR.fac factor for scale based integration range (i.e.; median of the distribution ±IQR.fac×IQR).
Details

For independent r.v.'s $X, E$ with univariate, absolutely continuous (a.c.) distributions $Regr$ and $Error$, respectively, `PrognCondDistribution()` returns the (factorized, conditional) posterior distribution of $X$ given $X+E=y$. as an object of class `PrognCondDistribution`.

Value

Object of class "PrognCondDistribution"

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>.

See Also

`PrognCondDistribution-class`; `demo('Prognose.R')`.

Examples

```r
PrognCondDistribution(Error = ConvexContamination(Norm(), Norm(4,1), size=0.1))
```

Posterior distribution in convolution

Description

The posterior distribution of $X$ given $(X+E)=y$

Objects from the Class

Objects can be created by calls of the form `PrognCondDistribution` where `Regr` and `error` are the respective (a.c.) distributions of $X$ and $E$ and the other arguments control accuracy in integration.

Slots

- `cond`: Object of class "PrognCondition": condition
- `img`: Object of class "rSpace": the image space.
- `param`: Object of class "OptionalParameter": an optional parameter.
- `r`: Object of class "function": generates random numbers.
- `d`: Object of class "OptionalFunction": optional conditional density function.
- `p`: Object of class "OptionalFunction": optional conditional cumulative distribution function.
- `q`: Object of class "OptionalFunction": optional conditional quantile function.
- `gaps`: (numeric) matrix or NULL
- `.withArith`: logical: used internally to issue warnings as to interpretation of arithmetics
.withSim: logical: used internally to issue warnings as to accuracy
.logExact: logical: used internally to flag the case where there are explicit formulae for the log
version of density, cdf, and quantile function
.lowerExact: logical: used internally to flag the case where there are explicit formulae for the
lower tail version of cdf and quantile function

Extends

Class "AbscontCondDistribution", directly.
Class "Distribution", by classes "UnivariateCondDistribution" and "AbscontCondDistribution".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

PrognCondition-class, UnivariateCondDistribution-class, AbscontCondDistribution-class, Distribution-class

Examples

PrognCondition()
Slots

- **name**: Object of class "character": name of the PrognCondition
- **range**: Object of class "EuclideanSpace": range of the PrognCondition

Extends

Class "Condition", directly.

Methods

- **show**: signature(object = "PrognCondition")

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

PrognCondDistribution-class, Condition-class

Examples

PrognCondition()

---

**TotalVarDist**

*Generic function for the computation of the total variation distance of two distributions*

Description

Generic function for the computation of the total variation distance $d_v$ of two distributions $P$ and $Q$ where the distributions may be defined for an arbitrary sample space $(\Omega, \mathcal{A})$. The total variation distance is defined as

$$d_v(P, Q) = \sup_{B \in \mathcal{A}} |P(B) - Q(B)|$$

Usage

TotalVarDist(e1, e2, ...)

## S4 method for signature 'AbscontDistribution,AbscontDistribution'

TotalVarDist(e1, e2, rel.tol=.Machine$double.eps^0.3, TruncQuantile = getdistrOption("TruncQuantile"), IQR.fac = 15, ..., diagnostic = FALSE)

## S4 method for signature 'AbscontDistribution,DiscreteDistribution'

TotalVarDist(e1, e2, ...)

## S4 method for signature 'DiscreteDistribution,AbscontDistribution'

TotalVarDist(e1, e2, ...)
Arguments

e1  object of class "Distribution" or "numeric"
e2  object of class "Distribution" or "numeric"
asis.smooth.discretize
   possible methods are "asis", "smooth" and "discretize". Default is "discretize".
n.discrim  if asis.smooth.discretize is equal to "discretize" one has to specify the
   number of lattice points used to discretize the abs. cont. distribution.
low.discrim if asis.smooth.discretize is equal to "discretize" one has to specify the
   lower end point of the lattice used to discretize the abs. cont. distribution.
up.discrim  if asis.smooth.discretize is equal to "discretize" one has to specify the
   upper end point of the lattice used to discretize the abs. cont. distribution.
h.smooth  if asis.smooth.discretize is equal to "smooth" – i.e., the empirical distribu-
   tion of the provided data should be smoothed – one has to specify this parameter.
rel.tol  relative accuracy requested in integration
TruncQuantile  Quantile the quantile based integration bounds (see details)
IQR.fac  Factor for the scale based integration bounds (see details)
...  further arguments to be used in particular methods – (in package distrEx: just
   used for distributions with a.c. parts, where it is used to pass on arguments to
distrExIntegrate).
diagnostic logical; if TRUE, the return value obtains an attribute "diagnostic" with diagnostic information on the integration, i.e., a list with entries method ("integrate" or "GLIntegrate"), call, result (the complete return value of the method), args (the args with which the method was called), and time (the time to compute the integral).

Details

For distances between absolutely continuous distributions, we use numerical integration; to determine sensible bounds we proceed as follows: by means of \( \min(\text{getLow}(e1, \text{eps} = \text{TruncQuantile}), \text{getLow}(e2, \text{eps} = \text{TruncQuantile})) \), we determine quantile based bounds \( c(\text{low}, \text{up}) \), and by means of \( s1 = \max(\text{IQR}(e1), \text{IQR}(e2)); m1 = \text{median}(e1); m2 = \text{median}(e2) \) and \( \text{low.1} = \min(m1, m2) - s1*\text{IQR. fac. up.1} = \max(m1, m2) + s1*\text{IQR.fac} \), we determine scale based bounds; these are combined by \( \text{low} = \max(\text{low.0}, \text{low.1}); \text{up} = \max(\text{up.0}, \text{up.1}) \).

In case we want to compute the total variation distance between (empirical) data and an abs. cont. distribution, we can specify the parameter \text{asis.smooth.discretize} \text{discretize} \text{smooth} \text{to avoid trivial distances (distance = 1).}

Using \text{asis.smooth.discretize} = "discretize", which is the default, leads to a discretization of the provided abs. cont. distribution and the distance is computed between the provided data and the discretized distribution.

Using \text{asis.smooth.discretize} = "smooth" causes smoothing of the empirical distribution of the provided data. This is, the empirical data is convoluted with the normal distribution \( \text{Norm}(\text{mean} = 0, \text{sd} = h.\text{smooth}) \) which leads to an abs. cont. distribution. Afterwards the distance between the smoothed empirical distribution and the provided abs. cont. distribution is computed.

Diagnostics on the involved integrations are available if argument diagnostic is TRUE. Then there is attribute diagnostic attached to the return value, which may be inspected and accessed through \text{showDiagnostic} and \text{getDiagnostic}.

Value

Total variation distance of \( e1 \) and \( e2 \)

Methods

\( e1 = "\text{AbscontDistribution}" \), \( e2 = "\text{AbscontDistribution}" \): total variation distance of two absolutely continuous univariate distributions which is computed using \text{distrExIntegrate}.

\( e1 = "\text{AbscontDistribution}" \), \( e2 = "\text{DiscreteDistribution}" \): total variation distance of absolutely continuous and discrete univariate distributions (are mutually singular; i.e., have distance \( =1 \)).

\( e1 = "\text{DiscreteDistribution}" \), \( e2 = "\text{DiscreteDistribution}" \): total variation distance of two discrete univariate distributions which is computed using \text{support} and \text{sum}.

\( e1 = "\text{DiscreteDistribution}" \), \( e2 = "\text{AbscontDistribution}" \): total variation distance of discrete and absolutely continuous univariate distributions (are mutually singular; i.e., have distance \( =1 \)).

\( e1 = "\text{numeric}" \), \( e2 = "\text{DiscreteDistribution}" \): Total variation distance between (empirical) data and a discrete distribution.

\( e1 = "\text{DiscreteDistribution}" \), \( e2 = "\text{numeric}" \): Total variation distance between (empirical) data and a discrete distribution.
**UnivariateCondDistribution-class**

**Description**

Class of univariate conditional distributions.

---

**e1 = "numeric", e2 = "AbscontDistribution":** Total variation distance between (empirical) data and an abs. cont. distribution.

**e1 = "AbscontDistribution", e1 = "numeric":** Total variation distance between (empirical) data and an abs. cont. distribution.

**e1 = "AcDcLeDistribution", e2 = "AcDcLeDistribution":** Total variation distance of mixed discrete and absolutely continuous univariate distributions.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>,
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

**References**


**See Also**

`totalvardistMmethods`, `ContaminationSize`, `KolmogorovDist`, `HellingerDist`, `Distribution-class`

**Examples**

```r
TotalVarDist(Norm(), UnivarMixingDistribution(Norm(1,2),Norm(0.5,3),
        mixCoeff=c(0.2,0.8)))
TotalVarDist(Norm(), Td(10))
TotalVarDist(Norm(mean = 50, sd = sqrt(25)), Binom(size = 100)) # mutually singular
TotalVarDist(Pois(10), Binom(size = 20))

x <- rnorm(100)
TotalVarDist(Norm(), x)
TotalVarDist(x, Norm(), asis.smooth.discretize = "smooth")

y <- (rbinom(50, size = 20, prob = 0.5)-10)/sqrt(5)
TotalVarDist(y, Norm())
TotalVarDist(y, Norm(), asis.smooth.discretize = "smooth")

TotalVarDist(rbinom(50, size = 20, prob = 0.5), Binom(size = 20, prob = 0.5))
```
Objects from the Class

Objects can be created by calls of the form `new("UnivariateCondDistribution", ...)`. 

Slots

- `cond` Object of class "Condition": condition
- `img` Object of class "rSpace": the image space.
- `param` Object of class "OptionalParameter": an optional parameter.
- `r` Object of class "function": generates random numbers.
- `d` Object of class "OptionalFunction": optional conditional density function.
- `p` Object of class "OptionalFunction": optional conditional cumulative distribution function.
- `q` Object of class "OptionalFunction": optional conditional quantile function.
- `.withArith` logical: used internally to issue warnings as to interpretation of arithmetics
- `.withSim` logical: used internally to issue warnings as to accuracy
- `.logExact` logical: used internally to flag the case where there are explicit formulae for the log version of density, cdf, and quantile function
- `.lowerExact` logical: used internally to flag the case where there are explicit formulae for the lower tail version of cdf and quantile function
- `Symmetry` object of class "DistributionSymmetry": used internally to avoid unnecessary calculations.

Extends

Class "UnivariateDistribution", directly.
Class "Distribution", by class "UnivariateDistribution".

Methods

- `cond` signature(object = "UnivariateCondDistribution"): accessor function for slot `cond`.
- `show` signature(object = "UnivariateCondDistribution")
- `plot` signature(object = "UnivariateCondDistribution"): not yet implemented.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

- `Distribution-class`

Examples

- `new("UnivariateCondDistribution")`
Generic Functions for the Computation of Functionals

Description

Generic functions for the computation of functionals on distributions.

Usage

IQR(x, ...)

## S4 method for signature 'UnivariateDistribution'
IQR(x)
## S4 method for signature 'UnivariateCondDistribution'
IQR(x, cond)
## S4 method for signature 'AffLinDistribution'
IQR(x)
## S4 method for signature 'DiscreteDistribution'
IQR(x)
## S4 method for signature 'Arcsine'
IQR(x)
## S4 method for signature 'Cauchy'
IQR(x)
## S4 method for signature 'Dirac'
IQR(x)
## S4 method for signature 'DExp'
IQR(x)
## S4 method for signature 'Exp'
IQR(x)
## S4 method for signature 'Geom'
IQR(x)
## S4 method for signature 'Logis'
IQR(x)
## S4 method for signature 'Norm'
IQR(x)
## S4 method for signature 'Unif'
IQR(x)

median(x, ...)

## S4 method for signature 'UnivariateDistribution'
median(x)
## S4 method for signature 'UnivariateCondDistribution'
median(x, cond)
## S4 method for signature 'AffLinDistribution'
median(x)
## S4 method for signature 'Arcsine'
median(x)
## S4 method for signature 'Cauchy'
median(x)
## S4 method for signature 'Dirac'
median(x)
## S4 method for signature 'DExp'
median(x)
## S4 method for signature 'Exp'
median(x)
## S4 method for signature 'Geom'
median(x)
## S4 method for signature 'Logis'
median(x)
## S4 method for signature 'Lnorm'
median(x)
## S4 method for signature 'Norm'
median(x)
## S4 method for signature 'Unif'
median(x)
mad(x, ...)
## S4 method for signature 'UnivariateDistribution'
mad(x)
## S4 method for signature 'AffLinDistribution'
mad(x)
## S4 method for signature 'Cauchy'
mad(x)
## S4 method for signature 'Dirac'
mad(x)
## S4 method for signature 'DExp'
mad(x)
## S4 method for signature 'Exp'
mad(x)
## S4 method for signature 'Geom'
mad(x)
## S4 method for signature 'Logis'
mad(x)
## S4 method for signature 'Norm'
mad(x)
## S4 method for signature 'Unif'
mad(x)
## S4 method for signature 'Arcsine'
mad(x)

sd(x, ...)
## S4 method for signature 'UnivariateDistribution'
sd(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'Norm'

sd(x, fun, cond, withCond = FALSE, useApply = TRUE, ...)

var(x, ...)
## S4 method for signature 'UnivariateDistribution'

var(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'AfflinDistribution'

var(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'CompoundDistribution'

var(x, ...)
## S4 method for signature 'Arcsine'

var(x, ...)
## S4 method for signature 'Binom'

var(x, ...)
## S4 method for signature 'Beta'

var(x, ...)
## S4 method for signature 'Cauchy'

var(x, ...)
## S4 method for signature 'Chisq'

var(x, ...)
## S4 method for signature 'Dirac'

var(x, ...)
## S4 method for signature 'DExp'

var(x, ...)
## S4 method for signature 'Exp'

var(x, ...)
## S4 method for signature 'Fd'

var(x, ...)
## S4 method for signature 'Gammad'

var(x, ...)
## S4 method for signature 'Geom'

var(x, ...)
## S4 method for signature 'Hyper'

var(x, ...)
## S4 method for signature 'Logis'

var(x, ...)
## S4 method for signature 'Lnorm'

var(x, ...)
## S4 method for signature 'Nbinom'

var(x, ...)
## S4 method for signature 'Norm'

var(x, ...)
## S4 method for signature 'Pois'

var(x, ...)
## S4 method for signature 'Td'

var(x, ...)
## S4 method for signature 'Unif'
var(x, ...)
## S4 method for signature 'Weibull'
var(x, ...)

skewness(x, ...)
## S4 method for signature 'UnivariateDistribution'
skewness(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'AffLinDistribution'
skewness(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'Arcsine'
skewness(x, ...)
## S4 method for signature 'Binom'
skewness(x, ...)
## S4 method for signature 'Beta'
skewness(x, ...)
## S4 method for signature 'Cauchy'
skewness(x, ...)
## S4 method for signature 'Chisq'
skewness(x, ...)
## S4 method for signature 'Dirac'
skewness(x, ...)
## S4 method for signature 'DExp'
skewness(x, ...)
## S4 method for signature 'Exp'
skewness(x, ...)
## S4 method for signature 'Fd'
skewness(x, ...)
## S4 method for signature 'Gammad'
skewness(x, ...)
## S4 method for signature 'Geom'
skewness(x, ...)
## S4 method for signature 'Hyper'
skewness(x, ...)
## S4 method for signature 'Logis'
skewness(x, ...)
## S4 method for signature 'Lnorm'
skewness(x, ...)
## S4 method for signature 'Nbinom'
skewness(x, ...)
## S4 method for signature 'Norm'
skewness(x, ...)
## S4 method for signature 'Pois'
skewness(x, ...)
## S4 method for signature 'Td'
skewness(x, ...)
## S4 method for signature 'Unif'
skewness(x, ...)
## S4 method for signature 'Weibull'
skewness(x, ...)

kurtosis(x, ...)
## S4 method for signature 'UnivariateDistribution'
kurtosis(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'AfflinDistribution'
kurtosis(x, fun, cond, withCond, useApply, ...)
## S4 method for signature 'Arcsine'
kurtosis(x, ...)
## S4 method for signature 'Binom'
kurtosis(x, ...)
## S4 method for signature 'Beta'
kurtosis(x, ...)
## S4 method for signature 'Cauchy'
kurtosis(x, ...)
## S4 method for signature 'Chisq'
kurtosis(x, ...)
## S4 method for signature 'Dirac'
kurtosis(x, ...)
## S4 method for signature 'DExp'
kurtosis(x, ...)
## S4 method for signature 'Exp'
kurtosis(x, ...)
## S4 method for signature 'Fd'
kurtosis(x, ...)
## S4 method for signature 'Gammad'
kurtosis(x, ...)
## S4 method for signature 'Geom'
kurtosis(x, ...)
## S4 method for signature 'Hyper'
kurtosis(x, ...)
## S4 method for signature 'Logis'
kurtosis(x, ...)
## S4 method for signature 'Lnorm'
kurtosis(x, ...)
## S4 method for signature 'Nbinom'
kurtosis(x, ...)
## S4 method for signature 'Norm'
kurtosis(x, ...)
## S4 method for signature 'Pois'
kurtosis(x, ...)
## S4 method for signature 'Td'
kurtosis(x, ...)
## S4 method for signature 'Unif'
kurtosis(x, ...)
## S4 method for signature 'Weibull'
kurtosis(x, ...)
Arguments

x object of class "UnivariateDistribution"

fun if missing the (conditional) variance resp. standard deviation is computed else the (conditional) variance resp. standard deviation of fun is computed.

cond if not missing the conditional variance resp. standard deviation given cond is computed.

... additional arguments to fun or E

useApply logical: should sapply, respectively apply be used to evaluate fund.

withCond logical: is cond in the argument list of fun.

Value

The value of the corresponding functional at the distribution in the argument is computed.

Methods

var, signature(x = "Any") interface to the stats-function var — see var resp. help(var, package="stats").

var, signature(x = "UnivariateDistribution"): variance of univariate distributions using corresponding E()-method.

var, signature(x = "AfflinDistribution"): if arguments fun, cond are missing: x@a^2 * var(x@X0) else uses method for signature(x = "UnivariateDistribution")

var, signature(x = "CompoundDistribution"): if we are in i.i.d. situation (i.e., slot SummandsDistr is of class UnivariateDistribution) the formula E[N]var[S]+(E[S]^2 + var(S))var(N) for N the frequency distribution and S the summand distribution; else we coerce to "UnivarLebDecDistribution".

sd, signature(x = "Any"): interface to the stats-function sd — see sd resp. help(sd, package="stats").

sd, signature(x = "NormParameter"): returns the slot sd of the parameter of a normal distribution — see sd resp. help(sd, package="distr").

sd, signature(x = "Norm"): returns the slot sd of the parameter of a normal distribution — see sd resp. help(sd, package="distr").

sd, signature(x = "UnivariateDistribution"): standard deviation of univariate distributions using corresponding E()-method.

IQR, signature(x = "Any"): interface to the stats-function IQR — see IQR resp. help(IQR, package="stats").

IQR, signature(x = "UnivariateDistribution"): interquartile range of univariate distributions using corresponding q()-method.

IQR, signature(x = "UnivariateCondDistribution"): interquartile range of univariate conditional distributions using corresponding q()-method.

IQR, signature(x = "DiscreteDistribution"): interquartile range of discrete distributions using corresponding q()-method but taking care that between upper and lower quartile there is 50% probability

IQR, signature(x = "AfflinDistribution"): abs(x@a) * IQR(x@X0)

median, signature(x = "Any"): interface to the stats-function median — see median resp. help(var, package="stats").
median, signature(x = "UnivariateDistribution"): median of univariate distributions using corresponding q()-method.

median, signature(x = "UnivariateCondDistribution"): median of univariate conditional distributions using corresponding q()-method.

median, signature(x = "AffLinDistribution"): \( x@a \ast \text{median}(x@X0) + x@b \)

mad, signature(x = "Any"): interface to the stats-function mad — see mad.

mad, signature(x = "UnivariateDistribution"): mad of univariate distributions using corresponding q()-method applied to abs(x-median(x)).

mad, signature(x = "AffLinDistribution"): abs(x@a) * mad(x@X0)

skewness, signature(x = "Any"): bias free estimation of skewness under normal distribution (default) as well as sample version (by argument sample.version = TRUE).

skewness, signature(x = "UnivariateDistribution"): skewness of univariate distributions using corresponding E()-method.

skewness, signature(x = "AffLinDistribution"): if arguments fun, cond are missing: skewness(x@X0)

else uses method for signature(x = "UnivariateDistribution")

kurtosis, signature(x = "Any"): bias free estimation of kurtosis under normal distribution (default) as well as sample version (by argument sample.version = TRUE).

kurtosis, signature(x = "UnivariateDistribution"): kurtosis of univariate distributions using corresponding E()-method.

kurtosis, signature(x = "AffLinDistribution"): if arguments fun, cond are missing: kurtosis(x@X0)

else uses method for signature(x = "UnivariateDistribution")

var, signature(x = "Arcsine"): exact evaluation using explicit expressions.

var, signature(x = "Beta"): for noncentrality 0 exact evaluation using explicit expressions.

var, signature(x = "Binom"): exact evaluation using explicit expressions.

var, signature(x = "Cauchy"): exact evaluation using explicit expressions.

var, signature(x = "Chisq"): exact evaluation using explicit expressions.

var, signature(x = "Dirac"): exact evaluation using explicit expressions.

var, signature(x = "DExp"): exact evaluation using explicit expressions.

var, signature(x = "Exp"): exact evaluation using explicit expressions.

var, signature(x = "Fd"): exact evaluation using explicit expressions.

var, signature(x = "Gamma"): exact evaluation using explicit expressions.

var, signature(x = "Geometric"): exact evaluation using explicit expressions.

var, signature(x = "Hyper"): exact evaluation using explicit expressions.

var, signature(x = "Logis"): exact evaluation using explicit expressions.

var, signature(x = "Lnorm"): exact evaluation using explicit expressions.

var, signature(x = "Nbinom"): exact evaluation using explicit expressions.

var, signature(x = "Norm"): exact evaluation using explicit expressions.

var, signature(x = "Pois"): exact evaluation using explicit expressions.

var, signature(x = "Td"): exact evaluation using explicit expressions.
var, signature(x = "Unif"): exact evaluation using explicit expressions.
var, signature(x = "Weibull"): exact evaluation using explicit expressions.
IQR, signature(x = "Arcsine"): exact evaluation using explicit expressions.
IQR, signature(x = "Cauchy"): exact evaluation using explicit expressions.
IQR, signature(x = "Dirac"): exact evaluation using explicit expressions.
IQR, signature(x = "DExp"): exact evaluation using explicit expressions.
IQR, signature(x = "Exp"): exact evaluation using explicit expressions.
IQR, signature(x = "Logis"): exact evaluation using explicit expressions.
IQR, signature(x = "Norm"): exact evaluation using explicit expressions.
IQR, signature(x = "Unif"): exact evaluation using explicit expressions.
median, signature(x = "Arcsine"): exact evaluation using explicit expressions.
median, signature(x = "Cauchy"): exact evaluation using explicit expressions.
median, signature(x = "Dirac"): exact evaluation using explicit expressions.
median, signature(x = "DExp"): exact evaluation using explicit expressions.
median, signature(x = "Exp"): exact evaluation using explicit expressions.
median, signature(x = "Geom"): exact evaluation using explicit expressions.
median, signature(x = "Logis"): exact evaluation using explicit expressions.
median, signature(x = "Lnorm"): exact evaluation using explicit expressions.
median, signature(x = "Norm"): exact evaluation using explicit expressions.
median, signature(x = "Unif"): exact evaluation using explicit expressions.
mad, signature(x = "Arcsine"): exact evaluation using explicit expressions.
mad, signature(x = "Cauchy"): exact evaluation using explicit expressions.
mad, signature(x = "Dirac"): exact evaluation using explicit expressions.
mad, signature(x = "DExp"): exact evaluation using explicit expressions.
mad, signature(x = "Exp"): exact evaluation using explicit expressions.
mad, signature(x = "Geom"): exact evaluation using explicit expressions.
mad, signature(x = "Logis"): exact evaluation using explicit expressions.
mad, signature(x = "Norm"): exact evaluation using explicit expressions.
mad, signature(x = "Unif"): exact evaluation using explicit expressions.
skewness, signature(x = "Arcsine"): exact evaluation using explicit expressions.
skewness, signature(x = "Beta"): for noncentrality 0 exact evaluation using explicit expressions.
skewness, signature(x = "Binom"): exact evaluation using explicit expressions.
skewness, signature(x = "Cauchy"): exact evaluation using explicit expressions.
skewness, signature(x = "Chisq"): exact evaluation using explicit expressions.
skewness, signature(x = "Dirac"): exact evaluation using explicit expressions.
skewness, signature(x = "DExp") : exact evaluation using explicit expressions.
skewness, signature(x = "Exp") : exact evaluation using explicit expressions.
skewness, signature(x = "Fd") : exact evaluation using explicit expressions.
skewness, signature(x = "Gammad") : exact evaluation using explicit expressions.
skewness, signature(x = "Geom") : exact evaluation using explicit expressions.
skewness, signature(x = "Hyper") : exact evaluation using explicit expressions.
skewness, signature(x = "Logis") : exact evaluation using explicit expressions.
skewness, signature(x = "Lnorm") : exact evaluation using explicit expressions.
skewness, signature(x = "Nbinom") : exact evaluation using explicit expressions.
skewness, signature(x = "Norm") : exact evaluation using explicit expressions.
skewness, signature(x = "Pois") : exact evaluation using explicit expressions.
skewness, signature(x = "Td") : exact evaluation using explicit expressions.
skewness, signature(x = "Unif") : exact evaluation using explicit expressions.
skewness, signature(x = "Weibull") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Arcsine") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Beta") : for noncentrality 0 exact evaluation using explicit expressions.
kurtosis, signature(x = "Binom") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Cauchy") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Chisq") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Dirac") : exact evaluation using explicit expressions.
kurtosis, signature(x = "DExp") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Exp") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Fd") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Gammad") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Geom") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Hyper") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Logis") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Lnorm") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Nbinom") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Norm") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Pois") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Td") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Unif") : exact evaluation using explicit expressions.
kurtosis, signature(x = "Weibull") : exact evaluation using explicit expressions.
Caveat

If any of the packages e1071, moments, fBasics is to be used together with distrEx the latter must be attached after any of the first mentioned. Otherwise kurtosis() and skewness() defined as methods in distrEx may get masked. To re-mask, you may use kurtosis <- distrEx::kurtosis; skewness <- distrEx::skewness. See also distrExMASK().

Acknowledgement

G. Jay Kerns, <gkerns@ysu.edu>, has provided a major contribution, in particular the functionals skewness and kurtosis are due to him.

Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>

See Also
distrExIntegrate, m1df, m2df, Distribution-class, sd, var, IQR, median, mad, sd, Sn, Qn

Examples

# Variance of Exp(1) distribution
var(Exp())

#median(Exp())
IQR(Exp())
mad(Exp())

# Variance of N(1,4)^2
var(Norm(mean=1, sd=2), fun = function(x)(x^2))
var(Norm(mean=1, sd=2), fun = function(x)(x^2), useApply = FALSE)

## sd -- may equivalently be replaced by var
sd(Pois()) ## uses explicit terms
sd(as(Pois(),"DiscreteDistribution")) ## uses sums
ds(as(Pois(),"UnivariateDistribution")) ## uses simulations
sd(Norm(mean=2), fun = function(x){2*x^2}) ## uses simulations
#
mad(sin(exp(Norm()+2*Pois())))) ## weird
Index

*Topic distribution
  AbscontCondDistribution-class, 7
  AsymTotalVarDist, 8
  Condition-class, 12
  ContaminationSize, 12
  ConvexContamination, 14
  CvMDist, 15
  DiscreteCondDistribution-class, 17
  DiscreteMVDistribution, 18
  DiscreteMVDistribution-class, 19
  distrExMASK, 23
  distrExMOVED, 24
  distrExOptions, 24
  E, 26
  EmpiricalMVDistribution, 35
  EuclCondition, 37
  EuclCondition-class, 37
  HellingerDist, 39
  KolmogorovDist, 43
  liesInSupport, 45
  LMCondDistribution, 46
  LMPParameter-class, 48
  m1df, 49
  m2df, 50
  makeEq, 52
  MultivariateDistribution-class, 53
  OAsymTotalVarDist, 54
  PrognCondDistribution, 58
  PrognCondDistribution-class, 59
  PrognCondition-class, 60
  TotalVarDist, 61
  UnivariateCondDistribution-class, 64
  var, 66
*Topic documentation
  distrExMASK, 23
  distrExMOVED, 24
*Topic math
  distrExIntegrate, 21
  GLIntegrate, 38
*Topic methods
  ConvexContamination, 14
  dim-methods, 17
  E, 26
  liesInSupport, 45
  m1df, 49
  m2df, 50
  var, 66
*Topic misc
  distrExOptions, 24
*Topic models
  LMCondDistribution, 46
  LMPParameter, 47
*Topic package
  distrEx-package, 2
*Topic programming
  distrExMASK, 23
  distrExMOVED, 24
*Topic utilities
  distrExIntegrate, 21
  GLIntegrate, 38
  liesInSupport, 45
  .qtIntegrate (E), 26
  .reorganizeDiagnosticList, 22

AbscontCondDistribution-class, 7
AsymTotalVarDist, 8
AsymTotalVarDist,AbscontDistribution,AbscontDistribution-method
(AsymTotalVarDist), 8
AsymTotalVarDist,AbscontDistribution,DiscreteDistribution-
(AsymTotalVarDist), 8
AsymTotalVarDist,AbscontDistribution,numeric-method
(AsymTotalVarDist), 8
AsymTotalVarDist,AcDcLcDistribution,AcDcLcDistribution-method
(AsymTotalVarDist), 8
AsymTotalVarDist,DiscreteDistribution,AbscontDistribution-
(AsymTotalVarDist), 8
AsymTotalVarDist,DiscreteDistribution,DiscreteDistribution-
(AsymTotalVarDist), 8
E, AffLinLatticeDistribution, missing, missing-method (E), 26
E, AffLinUnivarLebDecDistribution, missing, missing-method (E), 26
E, Arcsine, missing, missing-method (E), 26
E, Beta, missing, missing-method (E), 26
E, Binom, missing, missing-method (E), 26
E, Cauchy, function, missing-method (E), 26
E, Cauchy, missing, missing-method (E), 26
E, Chisq, missing, missing-method (E), 26
E, CompoundDistribution, missing, missing-method (E), 26
E, DExp, missing, missing-method (E), 26
E, Dirac, missing, missing-method (E), 26
E, DiscreteCondDistribution, function, numeric-method (E), 26
E, DiscreteCondDistribution, missing, numeric-method (E), 26
E, DiscreteDistribution, function, missing-method (E), 26
E, DiscreteDistribution, missing, missing-method (E), 26
E, DiscreteMVNDistribution, function, missing-method (E), 26
E, DiscreteMVNDistribution, missing, missing-method (E), 26
E, Exp, missing, missing-method (E), 26
E, Fd, missing, missing-method (E), 26
E, Gammad, function, missing-method (E), 26
E, Gammad, missing, missing-method (E), 26
E, Geom, missing, missing-method (E), 26
E, Hyper, missing, missing-method (E), 26
E, LatticeDistribution, function, missing-method (E), 26
E, LatticeDistribution, missing, missing-method (E), 26
E, Lnorm, missing, missing-method (E), 26
E, Logis, missing, missing-method (E), 26
E, MultivariateDistribution, function, missing-method (E), 26
E, MultivariateDistribution, missing, missing-method (E), 26
E, Nbinom, missing, missing-method (E), 26
E, Norm, missing, missing-method (E), 26
E, Pois, missing, missing-method (E), 26
E, Td, missing, missing-method (E), 26
E, Unif, missing, missing-method (E), 26
E, UnivariateCondDistribution, function, numeric-method (HellingerDist), 39
E, UnivariateCondDistribution, function, missing-method (HellingerDist), 39
E, UnivariateCondDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateCondDistribution, missing, missing-method (HellingerDist), 39
E, UnivariateDecDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateDecDistribution, missing, missing-method (HellingerDist), 39
E, UnivariateFunctionDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateFunctionDistribution, missing, missing-method (HellingerDist), 39
E, UnivariateKDEDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateKDEDistribution, missing, missing-method (HellingerDist), 39
E, UnivariateKernelDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateKernelDistribution, missing, missing-method (HellingerDist), 39
E, UnivariateKernelDistribution, numeric-method (HellingerDist), 39
E, UnivariateKernelDistribution, missing-method (HellingerDist), 39
E, UnivariateMVNDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateMVNDistribution, missing, missing-method (HellingerDist), 39
E, UnivariateMVNDistribution, function, numeric-method (HellingerDist), 39
E, UnivariateMVNDistribution, function, missing-method (HellingerDist), 39
E, UnivariateMVNDistribution, missing, numeric-method (HellingerDist), 39
E, UnivariateMVNDistribution, missing, method (HellingerDist), 39
E, UnivariateMVNDistribution, missing-method (HellingerDist), 39
E, UnivariateMVNDistribution, numeric-method (HellingerDist), 39
E, UnivariateMVNDistribution, method (HellingerDist), 39
E, UnivariateMVNDistribution, missing-method (HellingerDist), 39
HellingerDist, AbscontDistribution, numeric-method
  (HellingerDist), 39
HellingerDist, AcDcDistribution, AcDcDistribution-method
  (HellingerDist), 39
HellingerDist, DiscreteDistribution, AbscontDistribution-method
  (HellingerDist), 39
HellingerDist, DiscreteDistribution, DiscreteDistribution-method
  (HellingerDist), 39
HellingerDist, DiscreteDistribution, LatticeDistribution-method
  (HellingerDist), 39
HellingerDist, LatticeDistribution, DiscreteDistribution-method
  (HellingerDist), 39
HellingerDist, LatticeDistribution, LatticeDistribution-method
  (HellingerDist), 39
HellingerDist, numeric, AbscontDistribution-method
  (HellingerDist), 39
HellingerDist, numeric, DiscreteDistribution-method
  (HellingerDist), 39
HellingerDist-methods (HellingerDist), 39
hSmooth (distrExOptions), 24

integrate, 23, 39
IQR, 71, 75
IQR, (var), 66
IQR, AffLinAbscontDistribution-method
  (var), 66
IQR, AffLinDiscreteDistribution-method
  (var), 66
IQR, AffLinDiscreteDistribution-method (var), 66
IQR, AffLinDiscreteDistribution-method (var), 66
IQR, ANY-method (var), 66
IQR, Arcsine-method (var), 66
IQR, Cauchy-method (var), 66
IQR, DEmp-method (var), 66
IQR, Dirac-method (var), 66
IQR, DiscreteDistribution-method (var), 66
IQR, Exp-method (var), 66
IQR, Geom-method (var), 66
IQR, Logis-method (var), 66
IQR, Norm-method (var), 66
IQR, Unif-method (var), 66

KolmogorovDist, AbscontDistribution, AbscontDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, AbscontDistribution, DiscreteDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, AcDcDistribution, AcDcDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, DiscreteDistribution, AbscontDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, DiscreteDistribution, DiscreteDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, DiscreteDistribution, LatticeDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, LatticeDistribution, DiscreteDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, LatticeDistribution, LatticeDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, numeric, UnivariateDistribution-method
  (KolmogorovDist), 43
KolmogorovDist, UnivariateDistribution, numeric-method
  (KolmogorovDist), 43
KolmogorovDist-methods (KolmogorovDist), 43

Kurtosis, (var), 66
Kurtosis, AffLinAbscontDistribution-method
  (var), 66
Kurtosis, AffLinDiscreteDistribution-method
  (var), 66
Kurtosis, AffLinDiscreteDistribution-method (var), 66
Kurtosis, AffLinDistribution-method
  (var), 66
Kurtosis, AffLinLatticeDistribution-method
  (var), 66
Kurtosis, ANY-method (var), 66
Kurtosis, Arcsine-method (var), 66
Kurtosis, Beta-method (var), 66
Kurtosis, Binom-method (var), 66
Kurtosis, Cauchy-method (var), 66
Kurtosis, Chisq-method (var), 66
Kurtosis, DEmp-method (var), 66
Kurtosis, Dirac-method (var), 66
Kurtosis, Exp-method (var), 66
Kurtosis, Fd-method (var), 66
kurtosis, Gammad-method (var), 66
kurtosis, Geom-method (var), 66
kurtosis, Hyper-method (var), 66
kurtosis, Lnorm-method (var), 66
kurtosis, Logis-method (var), 66
kurtosis, Nbinom-method (var), 66
kurtosis, Norm-method (var), 66
kurtosis, Pois-method (var), 66
kurtosis, Td-method (var), 66
kurtosis, Unif-method (var), 66
kurtosis, UnivariateDistribution-method (var), 66
kurtosis, Weibull-method (var), 66
kurtosis-methods (var), 66

liesInSupport, 45
liesInSupport, DiscreteMVNormalDistribution, matrix-method
(liesInSupport), 45
liesInSupport, DiscreteMVNormalDistribution, numeric-method
(liesInSupport), 45
LMCondDistribution, 46
LMPar, 47, 48
LMPar-class, 48

mad, 72, 75
mad (var), 66

mad, AffLinAbscontDistribution-method (var), 66
mad, AffLinDiscreteDistribution-method (var), 66
mad, AffLinDistribution-method (var), 66
mad, AffLinLatticeDistribution-method (var), 66
mad, ANY-method (var), 66
mad, Arcsine-method (var), 66
mad, Cauchy-method (var), 66
mad, DExp-method (var), 66
mad, Dirac-method (var), 66
mad, Exp-method (var), 66
mad, Geom-method (var), 66
mad, Logis-method (var), 66
mad, Norm-method (var), 66
mad, Unif-method (var), 66
mad, UnivariateDistribution-method (var), 66
mad-methods (var), 66
makePQ, 52
MAPPING (distrExMAPPING), 23
MCIterations (distrExOptions), 24
median, 71, 75
median (var), 66
median, AffLinAbscontDistribution-method (var), 66
median, AffLinDiscreteDistribution-method (var), 66
median, AffLinDistribution-method (var), 66
median, AffLinLatticeDistribution-method (var), 66
median, ANY-method (var), 66
median, Arcsine-method (var), 66
median, Cauchy-method (var), 66
median, DEpx-method (var), 66
median, Dirac-method (var), 66
median, Exp-method (var), 66
median, Geom-method (var), 66
median, Lnorm-method (var), 66
median, Logis-method (var), 66
median, Norm-method (var), 66
median, Unif-method (var), 66
median, UnivariateCondDistribution-method (var), 66
median, UnivariateDistribution-method (var), 66
median-methods (var), 66
MOVEDING (distrExMOVED), 24
MultivariateDistribution-class, 53

name, Condition-method
  (Condition-class), 12
name<-, Condition-method
  (Condition-class), 12
nDiscretize (distrExOptions), 24

OAsymTotalVarDist, numeric, DiscreteDistribution-method
  (OAsymTotalVarDist), 54
OAsymTotalVarDist-methods
  (OAsymTotalVarDist), 54
options, 26
plot (plot-methods), 57
plot, MultivariateDistribution, missing-method
  (plot-methods), 57
plot, MultivariateDistribution-method
  (MultivariateDistribution-class), 53
plot, UnivariateCondDistribution, missing-method
  (plot-methods), 57
plot, UnivariateCondDistribution-method
  (UnivariateCondDistribution-class), 64
plot-methods, 57
print.DiagnosticClass
  (distrExIntegrate), 21
PrognCondDistribution, 58, 59
PrognCondDistribution-class, 59
PrognCondition (PrognCondition-class), 60
PrognCondition-class, 60

Qn, 75

Range (EuclCondition-class), 37
Range, EuclCondition-method
  (EuclCondition-class), 37
sd (var), 66
sd, SumDistribution-method, 66
sd, UnivariateDistribution-method (var), 66
sd-methods (var), 66
show, CONDITION-method
  (EuclCondition-class), 37
show, QuadraticParameter-method
  (LMPParameter-class), 48
show, MultivariateDistribution-method
  (MultivariateDistribution-class), 60
show, PrognCondition-method
  (PrognCondition-class), 60
show, UnivariateCondDistribution-method
  (UnivariateCondDistribution-class), 64
showDiagnostic, 10, 16, 32, 41, 56, 63
showDiagnostic (distrExIntegrate), 21
skewness (var), 66
skewness, AffLinAbscontDistribution-method (var), 66
skewness, AffLinDiscreteDistribution-method (var), 66
skewness, AffLinDistribution-method (var), 66
skewness, AffLinLatticeDistribution-method (var), 66
skewness, ANY-method (var), 66
skewness, Arcsine-method (var), 66
skewness, Beta-method (var), 66
skewness, Binom-method (var), 66
skewness, Cauchy-method (var), 66
skewness, Chisq-method (var), 66
skewness, DExp-method (var), 66
skewness, Dirac-method (var), 66
skewness, Exp-method (var), 66
skewness, Fd-method (var), 66
skewness, Gammad-method (var), 66
skewness, Geom-method (var), 66
skewness, Hyper-method (var), 66
skewness, Lnorm-method (var), 66
skewness, Logis-method (var), 66
skewness, Nbinom-method (var), 66
skewness, Norm-method (var), 66
skewness, Pois-method (var), 66
skewness, Td-method (var), 66
skewness, Unif-method (var), 66
skewness, UnivariateDistribution-method (var), 66
skewness, Weibull-method (var), 66
skewness-methods (var), 66
Sn, 75
support, DiscreteMVDistribution-method (DiscreteMVDistribution-class), 19
TotalVarDist, 14, 16, 42, 44, 61
TotalVarDist, AbscontDistribution, AbscontDistribution-method (TotalVarDist), 61
TotalVarDist, AbscontDistribution, DiscreteDistribution-method (TotalVarDist), 61
TotalVarDist, AbscontDistribution, numeric-method (TotalVarDist), 61
TotalVarDist, AcDcLcDistribution, AcDcLcDistribution-method (TotalVarDist), 61
TotalVarDist, DiscreteDistribution, numeric-method (TotalVarDist), 61
TotalVarDist, DiscreteDistribution, numeric-method (TotalVarDist), 61
TotalVarDist, DiscreteDistribution, LatticeDistribution-method (TotalVarDist), 61
TotalVarDist, DiscreteDistribution, numeric-method (TotalVarDist), 61
TotalVarDist, DiscreteMVDistribution, DiscreteMVDistribution-method (TotalVarDist), 61
TotalVarDist, LatticeDistribution, DiscreteDistribution-method (TotalVarDist), 61
TotalVarDist, LatticeDistribution, LatticeDistribution-method (TotalVarDist), 61
TotalVarDist, numeric, AbscontDistribution-method (TotalVarDist), 61
TotalVarDist, numeric, DiscreteDistribution-method (TotalVarDist), 61
TotalVarDist, methods (TotalVarDist), 61
UnivariateCondDistribution-class, 64
var, 66, 71, 75
var, AffLinAbscontDistribution-method (var), 66
var, AffLinDiscreteDistribution-method (var), 66
var, AffLinDistribution-method (var), 66
var, AffLinLatticeDistribution-method (var), 66
var, ANY-method (var), 66
var, Arcsine-method (var), 66
var, Beta-method (var), 66
var, Binom-method (var), 66
var, Cauchy-method (var), 66
var, Chisq-method (var), 66
var, CompoundDistribution-method (var), 66
var, DExp-method (var), 66
var, Dirac-method (var), 66
var, Exp-method (var), 66
var, Gammad-method (var), 66
var, Hyper-method (var), 66
var, Lnorm-method (var), 66
var, Logis-method (var), 66
var, Normal-method (var), 66
INDEX

var,Pois-method (var), 66
var,Td-method (var), 66
var,Unif-method (var), 66
var,UnivariateDistribution-method (var), 66
var,Weibull-method (var), 66
var-methods (var), 66