Package ‘distrEx’

April 23, 2017

Version 2.6.1
Date 2017-04-23
Title Extensions of Package 'distr'
Description Extends package 'distr' by functionals, distances, and conditional distributions.
Depends R(>= 2.10.0), methods, distr(>= 2.2)
Imports startupmsg, utils, stats
Suggests tcltk
ByteCompile yes
License LGPL-3
Encoding latin1
URL http://distr.r-forge.r-project.org/
LastChangedDate {$LastChangedDate: 2017-04-22 12:23:44 +0200 (Sa, 22. Apr 2017) $}
LastChangedRevision {$LastChangedRevision: 1133 $}
SVNRevision 1132
NeedsCompilation yes
Author Matthias Kohl [cre, cph, aut],
   Peter Ruckdeschel [aut, cph]
Maintainer Matthias Kohl <Matthias.Kohl@stamats.de>
Repository CRAN
Date/Publication 2017-04-23 13:11:30 UTC

R topics documented:

  distrEx-package .................................................. 2
  AbscontCondDistribution-class ................................. 7
  AsymTotalVarDist ............................................... 8
  Condition-class ............................................... 11
  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

Package: distrEx
Version: 2.6.1
Date: 2017-04-22
Depends: R(>= 2.10.0), methods, distr(>= 2.2)
Imports: startupmsg, utils, stats
Suggests: tcltk
LazyLoad: yes
License: LGPL-3
URL: http://distr.r-forge.r-project.org/
SVNRevision: 1132

Classes

Distribution Classes

"Distribution" (from distr)
|>"UnivariateDistribution" (from distr)
|>|>|"AbscontDistribution" (from distr)
|>|>|>|"Gumbel"
|>|>|>|"Pareto"
|>|>|>|"GPareto"
|>|>|"MultivariateDistribution"
|>|>|"DiscreteMVDistribution-class"
|>|"UnivariateCondDistribution"
|>|>|"AbscontCondDistribution"
|>|>|>|"PrognCondDistribution"
|>|>|>|"DiscreteCondDistribution"

Condition Classes
Parameter Classes

"OptionalParameter" (from distr)
|>"Parameter" (from distr)
|>"LMPParameter"
|>"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:
makeQ1           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
LMPParameter       Generating function for LMPParameter-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)
- QAymTotalVarDist: 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. Natalyia Horbenko, <natalyia.horbenko@itwm.fraunhofer.de> has ported the actuar code for the Pareto distribution to this setup.

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>
AbscontCondDistribution-class

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 \((\Omega, \mathcal{A})\). 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.fac = 15)  
## 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, ...)

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.dscr  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 distribu-
tion 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 likeli-
hood 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 (not in package distrEx)

Details

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

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

Finally, for both discrete and absolutely continuous case, we clip this ratio downwards by \( 10^{-10} \) and upwards by \( 10^{10} \)

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 \( \text{Norm}(\text{mean} = \theta, \text{sd} = h.\text{smooth}) \) which leads to an abs. cont. distribution. Afterwards the distance between the smoothed empirical distribution and the provided abs. cont. distribution is computed.

Value

Asymmetric Total variation distance of e1 and e2

Methods

e1 = "AbscontDistribution", e2 = "AbscontDistribution": total variation distance of two absolutely 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 discrete 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.
**Condition-class**

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

---

**Description**

The class of conditions.

**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)
```

---

**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`
Methods

- **name** signature(object = "Condition"): accessor function for slot name.
- **name<-** 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 = "AcDeLcDistribution", e2 = "AcDeLcDistribution"**: convex contamination (pseudo-)distance of two discrete univariate distributions.

Author(s)

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

References


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.
ConvexContamination

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 = "AcDcLcDistribution", e2 = "AcDcLcDistribution", size = "numeric": convex combination of two univariate distributions which may be coerced to "UnivarLebDecDistribution".

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

References


See Also

ContaminationSize, Distribution-class

Examples

# Convex combination of two normal distributions
C1 <- ConvexContamination(e1 = Norm(), e2 = Norm(mean = 5), size = 0.1)
plot(C1)
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

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

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, ...)

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

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

Value

Cramer - von Mises distance of e1 and e2

Methods

e1 = "UnivariateDistribution", e2 = "UnivariateDistribution": Cramer - von Mises distance of two univariate distributions.
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

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

Examples

```r
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())
```
DiscreteCondDistribution-class

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", ...)}.

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
- \texttt{.lowerExact}: logical: used internally to flag the case where there are explicit formulae for the lower tail version of cdf and quantile function
- \texttt{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

```r
new("DiscreteCondDistribution")
```

---

DiscreteMVDistribution

*Generating function for multivariate discrete distribution*

---

Description

Generates an object of class "DiscreteMVDistribution".

Usage

```r
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

```r
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

```r
# 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
  - `.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**

```r
(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))
q(D2)
param(D2)
img(D2)

e1 <- E(D2) # expectation
```

---

**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, ...)
```

**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.
...          additional arguments to be passed to f. Remember to use argument names not matching those of integrate and GLIntegrate!

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

Value
Estimate of the integral.

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 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
**distrExOptions**

**Value**

no value is returned

**Author(s)**

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

**Examples**

```r
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**

```r
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)
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,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"), ...)

## 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"), ...)

## 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"), ...)
## 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"), ...) 

## 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"), ...) 

## 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,
    ...) 

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

## 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"), ...)

## 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") 
    , ...)

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

## S4 method for signature 'DiscreteCondDistribution,missing,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"), ...)

## S4 method for signature 'UnivarLebDecDistribution,missing,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"), ...)

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

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

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

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

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

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

## 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, ...)

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

## 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, ...)

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

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

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

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

## S4 method for signature 'GPareto,missing,missing'
E(object, low = NULL, upp = NULL, ...)  
## S4 method for signature 'GPareto, function, missing'
E(object, fun, low = NULL, upp = NULL,
   rel.tol = getdistrexOption("ERelativeTolerance"),
   lowerTruncQuantile = getdistrexOption("ELowerTruncQuantile"),
   upperTruncQuantile = getdistrexOption("EUpperTruncQuantile"),
   IQR.fac = max(10000, getdistrexOption("IQR.fac")), ...)
## 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, ...)  
## S4 method for signature 'Lnorm, missing, missing'
E(object, low = NULL, upp = NULL, ...)  
## 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 'Pareto, 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, ...)  
## S4 method for signature 'Td, missing, missing'
E(object, low = NULL, upp = NULL, ...)  
## S4 method for signature 'Weibull, missing, missing'
E(object, low = NULL, upp = NULL, ...)  

Arguments

object  object of class "Distribution"
fun  if missing the (conditional) expectation is computed else the (conditional) ex-
     pectation 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).
...  additional arguments to fun
useApply  logical: should sapply, respectively apply be used to evaluate fun.
withCond  logical: is cond in the argument list of fun.
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.

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 = "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 SummandsDistr is 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 = "F", 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 substitution method \( y := \log(x) \) for numerical integration.
object = "Geom", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Gumbel", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "GPareto", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "GPareto", fun = "function", cond = "missing": use substitution method \( y := \log(x) \) for numerical integration.
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 = "Pareto", 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 = "T", fun = "missing", cond = "missing": exact evaluation using explicit expressions.
object = "Weibull", fun = "missing", cond = "missing": exact evaluation using explicit expressions.

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

# the same operator for discrete distributions:
P <- Pois(lambda=2)

E(P) ## uses explicit terms
E(as(P,"DiscreteDistribution")) ## uses sums
E(as(P,"UnivariateDistribution")) ## uses simulations
E(P, fun = function(x)(2*x^2)) ## uses simulations

# second moment of N(1,4)
E(Norm(mean=1, sd=2), fun = function(x){x^2})
E(Norm(mean=1, sd=2), fun = function(x){x^2}, useApply = FALSE)

# conditional distribution of a linear model
D1 <- LMCondDistribution(theta = 1)
E(D1, cond = 1)
E(D1, function(x)^2, cond = 1)
E(Norm(mean=1), fun = function(x){x^2})
E(D1, function(x, cond)^2, cond = 2, withCond = TRUE, useApply = FALSE)
E(Norm(mean=2), function(x)(2*x^2))

E(as(Norm(mean=2),"AbscontDistribution"))
### somewhat less accurate:
E(as(Norm(mean=2),"AbscontDistribution"),
   lowerTruncQuantil=1e-4,upperTruncQuantil=1e-4, IQR.fac= 4)
### even less accurate:
E(as(Norm(mean=2),"AbscontDistribution"),
   lowerTruncQuantil=1e-2,upperTruncQuantil=1e-2, IQR.fac= 4)
### no good idea, but just as an example:
E(as(Norm(mean=2),"AbscontDistribution"),
   lowerTruncQuantil=1e-2,upperTruncQuantil=1e-2, IQR.fac= .1)

### truncation of integration range; see also m1df...
E(Norm(mean=2), low=2, upp=4)
E(Cauchy())
E(Cauchy(),upp=3,low=-2)

# some Lebesgue decomposed distribution
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 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

mymix <- UnivarLebDecDistribution(acPart = Norm(), discretePart = Binom(4,.4),
acWeight = 0.4)
E(mymix)
EuclCondition

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**

```r
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**

```r
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`.

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 $(Ω, A)$. 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, ...)
## S4 method for signature 'AbscontDistribution,AbscontDistribution'
HellingerDist(e1,e2,
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistrOption("TruncQuantile"),
  IQR.fac = 15, ...)
## S4 method for signature 'AbscontDistribution,DiscreteDistribution'
HellingerDist(e1,e2, ...)
## S4 method for signature 'DiscreteDistribution,AbscontDistribution'
HellingerDist(e1,e2, ...)
## S4 method for signature 'DiscreteDistribution,DiscreteDistribution'
HellingerDist(e1,e2, ...)
## S4 method for signature 'numeric,DiscreteDistribution'
HellingerDist(e1, e2, ...)
## S4 method for signature 'DiscreteDistribution,numeric'
HellingerDist(e1, e2, ...)
## S4 method for signature 'numeric,AbscontDistribution'
HellingerDist(e1, e2, asis.smooth.discretize = "discretize",
  n.dscr = getdistrExOption("nDiscretize"),
  low.dscr = getLow(e2),
  up.dscr = getUp(e2),
  h.smooth = getdistrExOption("hSmooth"),
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistrOption("TruncQuantile"),
  IQR.fac = 15, ...)
## S4 method for signature 'AbscontDistribution,numeric'
HellingerDist(e1, e2, asis.smooth.discretize = "discretize",
  n.dscr = getdistrExOption("nDiscretize"),
  low.dscr = getLow(e1),
  up.dscr = getUp(e1),
  h.smooth = getdistrExOption("hSmooth"),
  rel.tol=.Machine$double.eps^0.3,
  TruncQuantile = getdistrOption("TruncQuantile"),
  IQR.fac = 15, ...)
## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
HellingerDist(e1,e2,  
    rel.tol=.Machine$double.eps^0.3,  
    TruncQuantile = getdistrOption("TruncQuantile"),  
    IQR.fac = 15, ...)  

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.distr  
    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.distr  
    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.distr  
    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 (not in package distrEx)  

Details  
For distances between absolutely continuous distributions, we use numerical integration; to deter-  
mine sensible bounds we proceed as follows: by means of min(getLow(e1,eps=TruncQuantile),getLow(e2,eps=TruncQuantile),  
max(getUp(e1,eps=TruncQuantile),getUp(e2,eps=TruncQuantile))) we determine quantile  
based bounds c(low.0,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.0,low.1), up <= max(up.0,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.  

Value  
Hellinger distance of e1 and e2
Methods

c1 = "AbscontDistribution", c2 = "AbscontDistribution": Hellinger distance of two absolutely continuous univariate distributions which is computed using \( \text{distrExintegrate} \).

c1 = "AbscontDistribution", c2 = "DiscreteDistribution": Hellinger distance of absolutely continuous and discrete univariate distributions (are mutually singular; i.e., have distance \( \neq 1 \)).

c1 = "DiscreteDistribution", c2 = "DiscreteDistribution": Hellinger distance of two discrete univariate distributions which is computed using \( \text{support} \) and \( \text{sum} \).

c1 = "DiscreteDistribution", c2 = "AbscontDistribution": Hellinger distance of discrete and absolutely continuous univariate distributions (are mutually singular; i.e., have distance \( \neq 1 \)).

c1 = "numeric", c2 = "DiscreteDistribution": Hellinger distance between (empirical) data and a discrete distribution.

c1 = "DiscreteDistribution", c2 = "numeric": Hellinger distance between (empirical) data and a discrete distribution.

c1 = "numeric", c2 = "AbscontDistribution": Hellinger distance between (empirical) data and an abs. cont. distribution.

c1 = "AbscontDistribution", c2 = "numeric": Hellinger distance between (empirical) data and an abs. cont. distribution.

c1 = "AcDcLcDistribution", c2 = "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

```r
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)
```
KolmogorovDist

HellingerDist(y, Norm())
HellingerDist(y, Norm(), asis.smooth.discretize = "smooth")

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

---

KolmogorovDist

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

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 = "AcDcLcDistribution", e2 = "AcDcLcDistribution": 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))
liesInSupport  

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)
## S4 method for signature 'DiscreteMVDistribution,matrix'
liesInSupport(object, x)
```

Arguments

- `object`: object of class "Distribution"
- `x`: numeric vector or matrix

Value

logical vector

Methods

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

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)
param(D1)
cond(D1)

  d(D1)(0, cond = 1)
  d(Norm(mean=1))(0)

  E(D1, cond = 1)
```
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")`
m1df

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, 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 = distrExOption("m2dfLowerTruncQuantile"),
   rel.tol = distrExOption("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 χ² distributions which is computed using pchisq.

Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

m2df-methods, E-methods
Examples

```r
# 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))
```

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 $(\Omega, A)$. 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$double.eps^0.3, Ngrid = 10000,
    TruncQuantile = getdistrOption("TruncQuantile"),
    IQR.fac = 15)
## S4 method for signature 'AbscontDistribution,DiscreteDistribution'
OAsymTotalVarDist(e1,e2, ...)
## S4 method for signature 'numeric, AbscontDistribution'
```
OAsymTotalVarDist(e1, e2, asis.smooth.discretize = "discretize",
                   n.discr = getdistrExOption("nDiscretize"),
                   low.discr = getLow(e2),
                   up.discr = getUp(e2),
                   h.smooth = getdistrExOption("hSmooth"),
                   rel.tol = .Machine$double.eps^0.3,
                   TruncQuantile = getdistrOption("TruncQuantile"),
                   IQR.fac = 15)
```

## S4 method for signature 'AbscontDistribution, numeric'
```
OAsymTotalVarDist(e1, e2,
                   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,
                   TruncQuantile = getdistrOption("TruncQuantile"),
                   IQR.fac = 15)
```

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

### 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.
- **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 (not in package distrex)

### Details

For distances between absolutely continuous distributions, we use numerical integration; to determine sensible bounds we proceed as follows: by means of \( \min(\text{getLow}(e1, \text{eps}=\text{TruncQuantile}), \text{getLow}(e2, \text{eps}=\text{TruncQuantile})) \) and \( \max(\text{getUp}(e1, \text{eps}=\text{TruncQuantile}), \text{getUp}(e2, \text{eps}=\text{TruncQuantile})) \)
We determine quantile based bounds \( c_\text{Hlow}, N_{PL}, \text{up}_N, \text{Q}_{PL} \), and by means of \( s_1 < \max(\text{IQR}(e_1), \text{IQR}(e_2)) \); \( m_1 \leftarrow \text{median}(e_1) \), \( m_2 \leftarrow \text{median}(e_2) \) and \( \text{low}.1 \leftarrow \min(m_1, m_2) - s_1 \times \text{IQR}. \text{fac} \), \( \text{up}.1 \leftarrow \max(m_1, m_2) + s_1 \times \text{IQR}. \text{fac} \) 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: \( x.\text{range} \leftarrow c(\text{seq}(\text{low}, \text{up}, \text{length}=\text{Ngrid}/3 \text{)}} \), \( q(e_1)(\text{seq}(\text{range})) \).

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 \( \text{asis.smooth.discretize} \) to avoid trivial distances (distance = 1).

Using \( \text{asis.smooth.discretize} = "\text{discretize}" \), 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 \( \text{asis.smooth.discretize} = "\text{smooth}" \) 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.

**Value**

**OAsymmetric 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 \( \text{distrExIntegrate} \).
- \( e_1 = "\text{AbscontDistribution}, e_2 = "\text{DiscreteDistribution}" \): total variation distance of absolutely continuous and discrete univariate distributions (are mutually singular; i.e., have distance =1).
- \( e_1 = "\text{DiscreteDistribution}, e_2 = "\text{DiscreteDistribution}" \): total variation distance of two discrete univariate distributions which is computed using \( \text{support} \) and \( \text{sum} \).
- \( e_1 = "\text{numeric}, e_2 = "\text{DiscreteDistribution}" \): Total variation distance between (empirical) data and a discrete distribution.
- \( e_1 = "\text{DiscreteDistribution}, e_2 = "\text{numeric}" \): Total variation distance between (empirical) data and a discrete distribution.
- \( e_1 = "\text{numeric}, e_2 = "\text{AbscontDistribution}" \): Total variation distance between (empirical) data and an abs. cont. distribution.
- \( e_1 = "\text{AbscontDistribution}, e_1 = "\text{numeric}" \): Total variation distance between (empirical) data and an abs. cont. distribution.
- \( e_1 = "\text{AcDcLcDistribution}, e_2 = "\text{AcDcLcDistribution}" \): Total variation distance of mixed discrete and absolutely continuous univariate distributions.
plot-methods

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))
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 dis-
tribution of X given X+E=y. as an object of class PrognCondDistribution.

Value

Object of class "PrognCondDistribution"
PrognCondDistribution-class

Author(s)

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

See Also

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

Examples

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

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

PrognCondDistribution()

---

Description

The class `PrognCondition` realizes the condition that `X+E=y` in a convolution setup

Usage

`PrognCondition(range = EuclideanSpace())`

Arguments

range an object of class "EuclideanSpace"

Value

Object of class "PrognCondition"

Objects from the Class

Objects can be created by calls of the form `PrognCondition(range)`.

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 $(Ω, A)$. The total variation distance is defined as

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

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, ...)

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

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

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

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

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

## S4 method for signature 'numeric, AbscontDistribution'
TotalVarDist(e1, e2, asis.smooth.discretize = "discretize",
    n.dscr = getdistrExOption("nDiscretize"),
    low.dscr = getLow(e2),
    up.dscr = getUp(e2), h.smooth = getdistrExOption("hSmooth"),
    rel.tol = .Machine$double.eps^0.3,
    TruncQuantile = getdistrOption("TruncQuantile"),
    IQR.fac = 15, ...)
TotalVarDist(e1, e2, asis.smooth.discretize = "discretize",
   n.discr = getdistrExOption("nDiscretize"),
   low.discr = getLow(e1),
   up.discr = getUp(e1),
   h.smooth = getdistrExOption("hSmooth"),
   rel.tol = .Machine$double.eps^0.3,
   TruncQuantile = getdistrOption("TruncQuantile"),
   IQR.fac = 15, ...
)
## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
TotalVarDist(e1, e2,
   rel.tol = .Machine$double.eps^0.3,
   TruncQuantile = getdistrOption("TruncQuantile"),
   IQR.fac = 15, ...
)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1, e2</td>
<td>object of class &quot;Distribution&quot; or &quot;numeric&quot;</td>
</tr>
<tr>
<td>asis.smooth.discretize</td>
<td>possible methods are &quot;asis&quot;, &quot;smooth&quot; and &quot;discretize&quot;. Default is &quot;discretize&quot;.</td>
</tr>
<tr>
<td>n.discr</td>
<td>if asis.smooth.discretize is equal to &quot;discretize&quot; one has to specify the</td>
</tr>
<tr>
<td></td>
<td>number of lattice points used to discretize the abs. cont. distribution.</td>
</tr>
<tr>
<td>low.discr</td>
<td>if asis.smooth.discretize is equal to &quot;discretize&quot; one has to specify the</td>
</tr>
<tr>
<td></td>
<td>lower end point of the lattice used to discretize the abs. cont. distribution.</td>
</tr>
<tr>
<td>up.discr</td>
<td>if asis.smooth.discretize is equal to &quot;discretize&quot; one has to specify the</td>
</tr>
<tr>
<td></td>
<td>upper end point of the lattice used to discretize the abs. cont. distribution.</td>
</tr>
<tr>
<td>h.smooth</td>
<td>if asis.smooth.discretize is equal to &quot;smooth&quot; — i.e., the empirical distribu-</td>
</tr>
<tr>
<td></td>
<td>tion of the provided data should be smoothed — one has to specify this parameter.</td>
</tr>
<tr>
<td>rel.tol</td>
<td>relative accuracy requested in integration</td>
</tr>
<tr>
<td>TruncQuantile</td>
<td>Quantile the quantile based integration bounds (see details)</td>
</tr>
<tr>
<td>IQR.fac</td>
<td>Factor for the scale based integration bounds (see details)</td>
</tr>
<tr>
<td>...</td>
<td>further arguments to be used in particular methods (not in package distrEx)</td>
</tr>
</tbody>
</table>

**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)), max(getUp(e1, eps=TruncQuantile), getUp(e2, eps=TruncQuantile)) we determine quantile based bounds \((\text{low}.0, \text{up}.0)\), and by means of \(s1 \leftarrow \max(\text{IQR}(e1), \text{IQR}(e2))\); \(m1 \leftarrow \text{median}(e1)\); \(m2 \leftarrow \text{median}(e2)\) and \(\text{low}.1 \leftarrow \min(m1,m2) - s1*\text{IQR}.\fac\); \(\text{up}.1 \leftarrow \max(m1,m2) + s1*\text{IQR}.\fac\) we determine scale based bounds; these are combined by \(\text{low} \leftarrow \max(\text{low}.0, \text{low}.1)\), \(\text{up} \leftarrow \max(\text{up}.0, \text{up}.1)\).

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 $\text{Norm}(\text{mean} = 0, \text{sd} = \text{h.smooth})$ which leads to an abs. cont. distribution. Afterwards the distance between the smoothed empirical distribution and the provided abs. cont. distribution is computed.

**Value**

Total variation distance of e1 and e2

**Methods**

- **e1 = "AbscontDistribution", e2 = "AbscontDistribution"**: total variation distance of two absolutely 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 $\neq 1$).

- **e1 = "DiscreteDistribution", e2 = "DiscreteDistribution"**: total variation distance of two discrete 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 $\neq 1$).

- **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 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"**: 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**

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

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))

UnivariateCondDistribution-class

Univariate conditional distribution

Description

Class of univariate conditional distributions.

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")

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 'Arctsine'
```
mad(x)
```

\[
\text{sd}(x, \ldots)
\]

## S4 method for signature 'UnivariateDistribution'
```
\text{sd}(x, \text{fun}, \text{cond}, \text{withCond}, \text{useApply}, \ldots)
```

## S4 method for signature 'Norm'
```
\text{sd}(x, \text{fun}, \text{cond}, \text{withCond} = \text{FALSE}, \text{useApply} = \text{TRUE}, \ldots)
```

\[
\text{var}(x, \ldots)
\]

## S4 method for signature 'UnivariateDistribution'
```
\text{var}(x, \text{fun}, \text{cond}, \text{withCond}, \text{useApply}, \ldots)
```

## S4 method for signature 'AfflinDistribution'
```
\text{var}(x, \text{fun}, \text{cond}, \text{withCond}, \text{useApply}, \ldots)
```

## S4 method for signature 'CompoundDistribution'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'Arctsine'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'Binom'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'Beta'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'Cauchy'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'Chisq'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'Dirac'
```
\text{var}(x, \ldots)
```

## S4 method for signature 'DExp'
```
\text{var}(x, \ldots)
```
## 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'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Geom'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Hyper'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Logis'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Lnorm'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Nbinom'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Norm'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Pois'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Td'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Unif'

\texttt{kurtosis(x, \ldots)}

## S4 method for signature 'Weibull'

\texttt{kurtosis(x, \ldots)}

### 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 \texttt{fun} is computed.
- **cond**: if not missing the conditional variance resp. standard deviation given \texttt{cond} is computed.
- **\ldots**: additional arguments to \texttt{fun} or \texttt{E}
- **useApply**: logical: should \texttt{sapply}, respectively \texttt{apply} be used to evaluate \texttt{fun}.
- **withCond**: logical: is \texttt{cond} in the argument list of \texttt{fun}.

### Value

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

### Methods

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

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

\texttt{var}, \texttt{signature(x = "AffLinDistribution")}: if arguments \texttt{fun}, \texttt{cond} are missing: \texttt{x@a*2 + \texttt{var(x@X@0)}} else uses method for signature(\texttt{x = "UnivariateDistribution"})
var, signature(x = "CompoundDistribution"): if we are in i.i.d. situation (i.e., slot SummandsDistr is of class UnivariateDistribution) the formula $E[N]\text{var}[S]+(E[S])^2 + \text{var}(S))\text{var}(N)$ for $N$ the frequency distribution and $S$ the summand distribution; else we coerce to "UnivarLebDecDistribution".

ds, signature(x = "Any"): interface to the stats-function sd — see sd resp. help(sd, package="stats").
ds, signature(x = "NormParameter"): returns the slot sd of the parameter of a normal distribution — see sd resp. help(sd, package="distr").
ds, signature(x = "Norm"): returns the slot sd of the parameter of a normal distribution — see sd resp. help(sd, package="distr").
ds, 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@0 * 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 sampleNversion = 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 sampleNversion = 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 = "F"): exact evaluation using explicit expressions.
var, signature(x = "Gammad"): exact evaluation using explicit expressions.
var, signature(x = "Geom"): 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.
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 = "Geom"): 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 = "Gammd"): 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 of 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
sd(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, 11
ContaminationSize, 12
ConvexContamination, 13
CvMDist, 15
DiscreteCondDistribution-class, 17
DiscreteMVDistribution, 18
DiscreteMVDistribution-class, 19
distrExMASK, 22
distrExMOVED, 22
distrExOptions, 23
E, 25
EmpiricalMVDistribution, 34
EuclCondition, 35
EuclCondition-class, 36
HellingerDist, 38
KolmogorovDist, 41
liesInSupport, 43
LMCondDistribution, 44
LMPParameter-class, 46
m1df, 47
m2df, 48
makeEst, 50
MultivariateDistribution-class, 51
OAsymTotalVarDist, 52
PrognCondDistribution, 56
PrognCondDistribution-class, 57
PrognCondition-class, 58
TotalVarDist, 59
UnivariateCondDistribution-class, 62
var, 63

*Topic documentation
distrExMASK, 22
distrExMOVED, 22

*Topic math
distrExIntegrate, 20

GLIntegrate, 37

*Topic methods
ConvexContamination, 13
dim-methods, 16
E, 25
liesInSupport, 43
m1df, 47
m2df, 48
var, 63

*Topic misc
distrExOptions, 23

*Topic models
LMCondDistribution, 44
LMPParameter, 45

*Topic package
distrEx-package, 2

*Topic programming
distrExMASK, 22
distrExMOVED, 22

*Topic utilities
distrExIntegrate, 20
GLIntegrate, 37
liesInSupport, 43

AbscontCondDistribution-class, 7
AsymTotalVarDist, 8
AsymTotalVarDist, AbscontDistribution, AbscontDistribution-m
(AsymTotalVarDist), 8
AsymTotalVarDist, AbscontDistribution, DiscreteDistribution-
(AsymTotalVarDist), 8
AsymTotalVarDist, AbscontDistribution, numeric-method
(AsymTotalVarDist), 8
AsymTotalVarDist, AcDcLcDistribution, AcDcLcDistribution-m
(AsymTotalVarDist), 8
AsymTotalVarDist, DiscreteDistribution, AbscontDistribution-
(AsymTotalVarDist), 8
AsymTotalVarDist, DiscreteDistribution, DiscreteDistribution-
(AsymTotalVarDist), 8
AsymTotalVarDist, DiscreteDistribution, LatticeDistribution-
(AsymTotalVarDist), 8
AsymTotalVarDist, DiscreteDistribution, numeric-method
(AsymTotalVarDist), 8
AsymTotalVarDist, LatticeDistribution, DiscreteDistribution-method
(AsymTotalVarDist), 8
AsymTotalVarDist, LatticeDistribution, LatticeDistribution-method
(AsymTotalVarDist), 8
AsymTotalVarDist, numeric, AbscontDistribution-method
(AsymTotalVarDist), 8
AsymTotalVarDist-methods
(AsymTotalVarDist), 8

cond
(UnivariateCondDistribution-class), 62
dim
(dim-methods), 16
dim, DiscreteMVDistribution-method
(dim-methods), 16
dim-methods, 16
DiscreteCondDistribution-class, 17
DiscreteDistribution, 34
DiscreteMVDistribution, 18, 20, 34
DiscreteMVDistribution-class, 19
distribution-package, 2
distrEx-package, 2
distrEx-integrate, 20, 33, 37, 40, 48, 72
distrExMASK, 22
distrExMV, 22
distrEx-package, 23
distrExOptions, 21, 23, 30, 47, 49
distributions, 23

ContaminationSize, 11, 12, 14, 16, 40, 42, 55, 61

ContaminationSize, AbscontDistribution, AbscontDistribution-method
(ContaminationSize), 12
ContaminationSize, AcDcLCDistribution, AcDcLCDistribution-method
(ContaminationSize), 12
ContaminationSize, DiscreteDistribution, DiscreteDistribution-method
(ContaminationSize), 12
ContaminationSize, DiscreteDistribution, LatticeDistribution-method
(ContaminationSize), 12
ContaminationSize-methods
(ContaminationSize), 12

ConvexContamination, 13
ConvexContamination, AbscontDistribution, AbscontDistribution-method
(ConvexContamination), 13
ConvexContamination, AbscontDistribution, UnivariateDistribution-method
(ConvexContamination), 13
ConvexContamination, AcDcLCDistribution, AcDcLCDistribution-method
(ConvexContamination), 13
ConvexContamination, DiscreteDistribution, DiscreteDistribution-method
(ConvexContamination), 13
ConvexContamination, DiscreteDistribution, LatticeDistribution-method
(ConvexContamination), 13
ConvexContamination, LatticeDistribution, DiscreteDistribution-method
(ConvexContamination), 13
ConvexContamination, LatticeDistribution, LatticeDistribution-method
(ConvexContamination), 13
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>E, AffLinUnivarLebDecDistribution, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Arcsine, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Beta, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Binom, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Cauchy, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Chisq, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, CompoundDistribution, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DExp, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Dirac, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DiscreteCondDistribution, function, numeric-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DiscreteCondDistribution, missing, numeric-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DiscreteDistribution, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DiscreteDistribution, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DiscreteMVDistribution, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, DiscreteMVDistribution, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Exp, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Fd, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Gammad, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Gammad, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Geom, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, GEV, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, GEV, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, GPareto, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, GPareto, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Gumbel, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Hyper, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, LatticeDistribution, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, LatticeDistribution, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Lnorm, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Logis, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, MultivariateDistribution, function, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, MultivariateDistribution, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Nbinom, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Norm, missing, missing-method (E)</td>
<td>25</td>
</tr>
<tr>
<td>E, Pareto, missing, missing-method (E)</td>
<td>25</td>
</tr>
</tbody>
</table>

Methods in italics indicate that they are specific to the mentioned distribution.
HellingerDist, AbscontDistribution, numeric-method
(HellingerDist), 38
IQR, UnivariateDistribution-method
(var), 63
IQR-method(var), 63
IQR.fact(distExOptions), 23
HellingerDist, AcDcLcDistribution, AcDcLcDistribution-method
(HellingerDist), 38
KolmogorovDist, 11, 13, 16, 40, 41, 55, 61
KolmogorovDist, AbscontDistribution, AbscontDistribution-method
(KolmogorovDist), 41
KolmogorovDist, AcDcLcDistribution, AcDcLcDistribution-method
(KolmogorovDist), 41
KolmogorovDist, DiscreteDistribution, DiscreteDistribution-method
(KolmogorovDist), 41
KolmogorovDist, DiscreteDistribution, AbscontDistribution-method
(KolmogorovDist), 41
KolmogorovDist, LatticeDistribution, LatticeDistribution-method
(KolmogorovDist), 41
KolmogorovDist, LatticeDistribution, DiscreteDistribution-method
(KolmogorovDist), 41
KolmogorovDist, DiscreteDistribution, LatticeDistribution-method
(KolmogorovDist), 41
KolmogorovDist, numeric, AbscontDistribution-method
(HellingerDist), 38
KolmogorovDist, numeric, DiscreteDistribution-method
(HellingerDist), 38
KolmogorovDist-methods (HellingerDist), 38
hSmooth (distExOptions), 23
integrate, 21, 37
IQR, 69, 72
IQR (var), 63
IQR, AffLinAbscontDistribution-method (var), 63
Kurtosis (var), 63
Kurtosis, AffLinAbscontDistribution-method (var), 63
Kurtosis, AffLinDiscreteDistribution-method (var), 63
Kurtosis, AffLinDiscreteDistribution-method (var), 63
Kurtosis, AffLinLatticeDistribution-method (var), 63
Kurtosis, ANY-method (var), 63
Kurtosis, Arcsine-method (var), 63
Kurtosis, Cauchy-method (var), 63
Kurtosis, DExp-method (var), 63
Kurtosis, Dirac-method (var), 63
Kurtosis, DiscreteDistribution-method (var), 63
Kurtosis, Exp-method (var), 63
Kurtosis, Geom-method (var), 63
Kurtosis, Logis-method (var), 63
Kurtosis, Norm-method (var), 63
Kurtosis, Unif-method (var), 63
Kurtosis, UnivariateCondDistribution-method (var), 63
kurtosis, Hyper-method (var), 63
kurtosis, Lnorm-method (var), 63
kurtosis, Logis-method (var), 63
kurtosis, Nbinom-method (var), 63
kurtosis, Norm-method (var), 63
kurtosis, Pois-method (var), 63
kurtosis, Td-method (var), 63
kurtosis, Unif-method (var), 63
kurtosis, UnivariateDistribution-method (var), 63
kurtosis, Weibull-method (var), 63
kurtosis-methods (var), 63

liesInSupport, 43
liesInSupport, DiscreteMVDistribution, matrix-method (liesInSupport), 43
liesInSupport, DiscreteMVDistribution, numeric-method (liesInSupport), 43
LMCondDistribution, 44
LMPParameter, 45, 46
LMPParameter-class, 46

m1df, 33, 47, 72
m1df, AbscontDistribution-method (m1df), 47
m1df, AffLinDistribution-method (m1df), 47
m1df, Binom-method (m1df), 47
m1df, Chisq-method (m1df), 47
m1df, Exp-method (m1df), 47
m1df, LatticeDistribution-method (m1df), 47
m1df, Norm-method (m1df), 47
m1df, Pois-method (m1df), 47
m1df, UnivariateDistribution-method (m1df), 47
m1df-methods (m1df), 47
m1dfLowerTruncQuantile (distrExOptions), 23
m1dfRelativeTolerance (distrExOptions), 23
m2df, 33, 48, 48, 72
m2df, AbscontDistribution-method (m2df), 48
m2df, AffLinDistribution-method (m2df), 48
m2df, Binom-method (m2df), 48
m2df, Chisq-method (m2df), 48
m2df, Exp-method (m2df), 48
m2df, LatticeDistribution-method (m2df), 48
m2df, Norm-method (m2df), 48
m2df, Pois-method (m2df), 48
m2df, UnivariateDistribution-method (m2df), 48
m2df-methods (m2df), 48
m2dfLowerTruncQuantile (distrExOptions), 23
m2dfRelativeTolerance (distrExOptions), 23
mad, 69, 72
mad (var), 63
mad, AffLinAbscontDistribution-method (var), 63
mad, AffLinDiscreteDistribution-method (var), 63
mad, AffLinDistribution-method (var), 63
mad, AffLinLatticeDistribution-method (var), 63
mad, ANY-method (var), 63
mad, Arcsine-method (var), 63
mad, Cauchy-method (var), 63
mad, Dexp-method (var), 63
mad, Dirac-method (var), 63
mad, Exp-method (var), 63
mad, Geom-method (var), 63
mad, Logis-method (var), 63
mad, Norm-method (var), 63
mad, Unif-method (var), 63
mad, UnivariateDistribution-method (var), 63
mad-methods (var), 63
make01, 50
MASKING (distrExMASK), 22
MCIterations (distrExOptions), 23
median, 69, 72
median (var), 63
median, AffLinAbscontDistribution-method (var), 63
median, AffLinDiscreteDistribution-method (var), 63
median, AffLinDistribution-method (var), 63
median, AffLinLatticeDistribution-method (var), 63
median, ANY-method (var), 63
median, Arcsine-method (var), 63
median, Cauchy-method (var), 63
median, DExp-method (var), 63
median, Dirac-method (var), 63
median, Exp-method (var), 63
median, Geom-method (var), 63
median, Lnorm-method (var), 63
median, Logis-method (var), 63
median, Norm-method (var), 63
median, Unif-method (var), 63
median, UnivariateCondDistribution-method (var), 63
median, UnivariateDistribution-method (var), 63
median-methods (var), 63
MOVEDING (distrExMOVED), 22
MultivariateDistribution-class, 51

name, Condition-method
(Condition-class), 11
name<-, Condition-method
(Condition-class), 11
nDiscretize (distrExOptions), 23

OAsymTotalVarDist, 52
OAsymTotalVarDist, AbscontDistribution, AbscontDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, AbscontDistribution, DiscreteDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, AbscontDistribution, numeric-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, AcDclCldistribution, AcDclCldistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, DiscreteDistribution, AbscontDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, DiscreteDistribution, DiscreteDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, DiscreteDistribution, LatticeDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, DiscreteDistribution, numeric-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, LatticeDistribution, DiscreteDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, LatticeDistribution, LatticeDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, numeric, AbscontDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist, numeric, DiscreteDistribution-method
(OAsymTotalVarDist), 52
OAsymTotalVarDist-methods
(OAsymTotalVarDist), 52
options, 24
plot (plot-methods), 55
plot, MultivariateDistribution, missing-method
(plot-methods), 55
plot, MultivariateDistribution-method
(MultivariateDistribution-class), 51
plot, UnivariateCondDistribution, missing-method
(plot-methods), 55
plot, UnivariateCondDistribution-method
(UnivariateCondDistribution-class), 62
plot-methods, 55
PrognCondDistribution, 56, 57
PrognCondDistribution-class, 57
PrognCondition (PrognCondition-class), 58
PrognCondition-class, 58
Qn, 72

Range (EuClCondition-class), 36
Range, EuClCondition-method
(EuClCondition-class), 36
sd, 69, 72
sd, Norm-method (var), 63
sd, numeric-method
(OAsymTotalVarDist), 63
show, EuClCondition-method
(EuClCondition-class), 36
show, LMPReparameter-method
(LMPReparameter-class), 46
show, MultivariateDistribution-method
(MultivariateDistribution-class), 51
skewness (var), 63
skewness-method
(OAsymTotalVarDist), 63
skewness, AffLinDistribution-method (var), 63
skewness, AffLinDistribution-method (var), 63
skewness, AffLinLatticeDistribution-method (var), 63
skewness, ANY-method (var), 63
skewness, Arcsine-method (var), 63
skewness, Beta-method (var), 63
skewness, Binom-method (var), 63
skewness, Cauchy-method (var), 63
skewness, Chisq-method (var), 63
skewness, DExp-method (var), 63
skewness, Dirac-method (var), 63
skewness, Exp-method (var), 63
skewness, Fd-method (var), 63
skewness, Gammad-method (var), 63
skewness, Geom-method (var), 63
skewness, Hyper-method (var), 63
skewness, Lnorm-method (var), 63
skewness, Logis-method (var), 63
skewness, Nbinom-method (var), 63
skewness, Norm-method (var), 63
skewness, Pois-method (var), 63
skewness, Td-method (var), 63
skewness, Unif-method (var), 63
skewness, UnivariateDistribution-method (var), 63
skewness, Weibull-method (var), 63
skewness-methods (var), 63
Sn, 72
support, DiscreteMVDistribution-method (DiscreteMVDistribution-class), 19

TotalVarDist, 13, 16, 40, 42, 59
TotalVarDist, AbscontDistribution, AbscontDistribution-method (TotalVarDist), 59
TotalVarDist, AbscontDistribution, DiscreteDistribution-method (TotalVarDist), 59
TotalVarDist, AbscontDistribution, numeric-method (TotalVarDist), 59
TotalVarDist, AcDlcDistribution, AcDlcDistribution-method (TotalVarDist), 59
TotalVarDist, DiscreteDistribution, AbscontDistribution-method (TotalVarDist), 59
TotalVarDist, DiscreteDistribution, DiscreteDistribution-method (TotalVarDist), 59

UnivariateCondDistribution-class, 62
var, 63, 68, 72
var, AffLinAbscontDistribution-method (var), 63
var, AffLinDiscreteDistribution-method (var), 63
var, AffLinDistribution-method (var), 63
var, ANY-method (var), 63
var, Arcsine-method (var), 63
var, Beta-method (var), 63
var, Binom-method (var), 63
var, Cauchy-method (var), 63
var, Chisq-method (var), 63
var, CompoundDistribution-method (var), 63
var, DExp-method (var), 63
var, Dirac-method (var), 63
var, Exp-method (var), 63
var, Fd-method (var), 63
var, Gammad-method (var), 63
var, Hyper-method (var), 63
var, Logis-method (var), 63
var, Nbinom-method (var), 63
var, Norm-method (var), 63
var, UnivariateDistribution-method (var), 63
var, Weibull-method (var), 63
var-methods (var), 63