

# Package ‘distr’

May 3, 2021

**Version** 2.8.0

**Date** 2019-03-11

**Title** Object Oriented Implementation of Distributions

**Description** S4-classes and methods for distributions.

**Depends** R(>= 2.14.0), methods, graphics, startupmsg, sfsmisc

**Suggests** distrEx, svUnit (>= 0.7-11), knitr

**Imports** stats, grDevices, utils, MASS

**VignetteBuilder** knitr

**ByteCompile** yes

**Encoding** latin1

**License** LGPL-3

**URL** <http://distr.r-forge.r-project.org/>

**LastChangedDate** {`$LastChangedDate: 2019-03-11 16:33:22 +0100 (Mo, 11 Mrz 2019)` \$}

**LastChangedRevision** {`$LastChangedRevision: 1315` \$}

**VCS/SVNRevision** 1314

**NeedsCompilation** yes

**Author** Florian Camphausen [ctb] (contributed as student in the initial phase --2005),  
Matthias Kohl [aut, cph],  
Peter Ruckdeschel [cre, cph],  
Thomas Stabla [ctb] (contributed as student in the initial phase --2005),  
R Core Team [ctb, cph] (for source file ks.c/ routines 'pKS2' and 'pKolmogorov2x')

**Maintainer** Peter Ruckdeschel <[peter.ruckdeschel@uni-oldenburg.de](mailto:peter.ruckdeschel@uni-oldenburg.de)>

**Repository** CRAN

**Date/Publication** 2019-03-11 20:32:54 UTC

**R topics documented:**

distr-package	5
AbscontDistribution	12
AbscontDistribution-class	14
Arcsine-class	18
Beta-class	19
BetaParameter-class	21
Binom-class	22
BinomParameter-class	24
Cauchy-class	25
CauchyParameter-class	27
Chisq-class	28
ChisqParameter-class	30
CompoundDistribution	31
CompoundDistribution-class	32
convpow-methods	34
d-methods	36
decomposePM-methods	36
DExp-class	37
df-methods	39
df1-methods	39
df2-methods	40
dim-methods	40
dimension-methods	40
Dirac-class	41
DiracParameter-class	42
DiscreteDistribution	44
DiscreteDistribution-class	46
distr-defunct	49
distrARITH	50
Distribution-class	51
DistributionSymmetry-class	52
DistrList	53
DistrList-class	54
distrMASK	55
distroptions	56
DistrSymmList	58
DistrSymmList-class	59
EllipticalSymmetry	59
EllipticalSymmetry-class	60
EmpiricalDistribution	61
EuclideanSpace-class	62
Exp-class	64
ExpParameter-class	65
Fd-class	67
flat.LCD	69
flat.mix	70

FParameter-class . . . . .	71
Gammad-class . . . . .	72
GammaParameter-class . . . . .	74
gaps-methods . . . . .	75
Geom-class . . . . .	76
getLabel . . . . .	78
getLow,getUp . . . . .	79
Huberize-methods . . . . .	80
Hyper-class . . . . .	81
HyperParameter-class . . . . .	82
igamma . . . . .	84
img-methods . . . . .	84
k-methods . . . . .	85
lambda-methods . . . . .	85
Lattice-class . . . . .	86
LatticeDistribution . . . . .	87
LatticeDistribution-class . . . . .	89
Length-methods . . . . .	92
liesIn-methods . . . . .	93
liesInSupport . . . . .	93
Lnorm-class . . . . .	95
LnormParameter-class . . . . .	97
location-methods . . . . .	98
Logis-class . . . . .	99
LogisParameter-class . . . . .	101
m-methods . . . . .	102
makeAbscontDistribution . . . . .	102
Math-methods . . . . .	103
Max-methods . . . . .	104
mean-methods . . . . .	105
meanlog-methods . . . . .	105
Min-methods . . . . .	106
Minimum-methods . . . . .	106
n-methods . . . . .	108
name-methods . . . . .	108
Naturals-class . . . . .	109
Nbinom-class . . . . .	110
NbinomParameter-class . . . . .	112
ncp-methods . . . . .	113
Norm-class . . . . .	114
NormParameter-class . . . . .	116
NoSymmetry . . . . .	117
NoSymmetry-class . . . . .	118
operators-methods . . . . .	118
OptionalParameter-class . . . . .	123
options . . . . .	124
p-methods . . . . .	125
p.l-methods . . . . .	125

param-methods . . . . .	126
Parameter-class . . . . .	126
pivot-methods . . . . .	127
plot-methods . . . . .	127
Pois-class . . . . .	133
PoisParameter-class . . . . .	135
PosDefSymmMatrix . . . . .	136
PosDefSymmMatrix-class . . . . .	137
print-methods . . . . .	138
prob-methods . . . . .	138
q-methods . . . . .	139
q.r-methods . . . . .	140
qqbounds . . . . .	140
qqplot . . . . .	142
r-methods . . . . .	146
rate-methods . . . . .	146
Reals-class . . . . .	147
rSpace-class . . . . .	148
RtoDPQ . . . . .	148
RtoDPQ.d . . . . .	150
RtoDPQ.LC . . . . .	151
scale-methods . . . . .	153
sd-methods . . . . .	154
sdlog-methods . . . . .	154
shape-methods . . . . .	155
shape1-methods . . . . .	155
shape2-methods . . . . .	156
simplifyD-methods . . . . .	156
simplifyr-methods . . . . .	158
size-methods . . . . .	158
solve-methods . . . . .	159
SphericalSymmetry . . . . .	160
SphericalSymmetry-class . . . . .	161
sqrt-methods . . . . .	162
standardMethods . . . . .	162
support-methods . . . . .	163
Symmetry-class . . . . .	163
Td-class . . . . .	164
TParameter-class . . . . .	166
Truncate-methods . . . . .	167
Unif-class . . . . .	168
UnifParameter-class . . . . .	170
UniNormParameter-class . . . . .	171
UnivarDistrList . . . . .	172
UnivarDistrList-class . . . . .	173
UnivariateDistribution-class . . . . .	174
UnivarLebDecDistribution . . . . .	176
UnivarLebDecDistribution-class . . . . .	177

UnivarMixingDistribution . . . . .	181
UnivarMixingDistribution-class . . . . .	182
Version Management . . . . .	184
Weibull-class . . . . .	185
WeibullParameter-class . . . . .	187
width-methods . . . . .	188

<b>Index</b>	<b>189</b>
--------------	------------

---

distr-package	<i>distr – Object Oriented Implementation of Distributions</i>
---------------	--

---

## Description

**distr** provides a conceptual treatment of distributions by means of S4 classes. A mother class `Distribution` is introduced with slots for a parameter and —most important— for the four constitutive methods `r`, `d`, `p`, and `q` for simulation respectively for evaluation of density / c.d.f.\ and quantile function of the corresponding distribution.

Most distributions of package **stats** (like normal, Poisson, etc.) are implemented as subclasses of either `AbscontDistribution` or `DiscreteDistribution`, which themselves are again subclasses of `Distribution`.

Up to arguments referring to a parameter of the distribution (like mean for the normal distribution), these function slots have the same arguments as those of package **stats**, i.e.; for a distribution object `X` we may call these functions as

- `r(X)(n)`
- `d(X)(x, log = FALSE)`
- `p(X)(q, lower.tail = TRUE, log.p = FALSE)`
- `q(X)(p, lower.tail = TRUE, log.p = FALSE)`

For the arguments of these function slots see e.g. [rnorm](#). Note that, as usual, slots `d`, `p`, and `q` are vectorized in their first argument, but are not on the subsequent ones.

In the environments of RStudio, see <https://www.rstudio.com/> and Jupyter IRKernel, see <https://github.com/IRkernel/IRkernel>, calls to `q` are caught away from standard R evaluation and are treated in a non-standard way. This non-standard evaluation in particular throws errors at calls to our accessor methods `q` to slot `q` of the respective distribution object. To amend this, we provide function `q.l` as alias to our accessors `q`, so that our packages also become available in these environments.

Arithmetics and unary mathematical transformations for distributions are available: For `Distribution` objects `X` and `Y` expressions like `3*X+sin(exp(-Y/4+3))` have their natural interpretation as corresponding image distributions.

## Details

Package:	distr
Version:	2.8.0
Date:	2019-03-11
Depends:	R(>= 2.14.0), methods, graphics, startupmsg, sfsmisc
Suggests:	distrEx, svUnit (>= 0.7-11)
Imports:	stats, grDevices, utils, MASS
LazyLoad:	yes
License:	LGPL-3
URL:	<a href="http://distr.r-forge.r-project.org/">http://distr.r-forge.r-project.org/</a>
VCS/SVNRevision:	1314

## Classes

Distribution classes have a slot `param` the class of which is specialized for the particular distributions. The parameter classes for the particular distributions have slots with names according to the corresponding `[rdpq]<name>` functions of package **base**. From version 1.9 on, `AbscontDistribution` and descendants have a slot `gaps` for gaps in the support. `DiscreteDistribution` and descendants have an additional slot `support`, which is again specialized to be a lattice for class `LatticeDistribution`.

For saved objects from earlier versions, we provide the methods `isOldVersion`, and `conv2NewVersion` to check whether the object was generated by an older version of this package and to convert such an object to the new format, respectively. This applies to objects of subclasses of `AbscontDistribution` lacking a `gap`-slot as well as to objects of subclasses of `LatticeDistribution` lacking a `lattice`-slot.

To enhance accuracy, from version 1.9 on, we also provide subclasses `AffLinAbscontDistribution`, `AffLinDiscreteDistribution`, and `AffLinLatticeDistribution`, as well as the class union `AffLinDistribution`, so that in particular functionals like `E` from package **distrEx** can recur to exact formula more frequently: These classes have additional slots `a`, `b`, and `X0` to reflect the fact, that a distribution object of these classes has the same distribution as  $a \cdot X_0 + b$ .

For all particular distributions, as well as for classes `AbscontDistribution`, `DiscreteDistribution`, `LatticeDistribution`, `UnivarDistrList` and `DistrList` generating functions are provided, e.g. `X <- Norm(mean = 3, sd = 2)`. The same goes for the space classes. All slots should be inspected / modified by means of corresponding accessor- / replacement functions; e.g. `mean(X) <- 3`

Again to enhance accuracy, from version 2.0 on, we also provide subclasses `UnivarMixingDistribution` to support mixing distributions, `UnivarLebDecDistribution`, to support Lebesgue decomposed distributions (with a discrete and an a.c. part) as well as `AffLinUnivarLebDecDistribution`, for corresponding affine linear transformations. Class `UnivarLebDecDistribution` is closed under arithmetical operations `+`, `/`, `*`, `^` for pairs of independent variables `+`, `-` for pairs of independent variables `+` affine linear transformations `+` truncation, huberization, min/max which are all now

available analytically.

(see Parameter classes).

[\*]: there is a generating function with the same name

```
#####
Distribution classes
#####
slots: [<name>(<class>)]
img(rSpace), param(OptionalParameter),
r(function), d(OptionalFunction), p(OptionalFunction), q(OptionalFunction),
.withSim(logical), .withArith(logical), .logExact(logical), .lowerExact(logical),
Symmetry(DistributionSymmetry)
```

```
"Distribution"
|>"UnivariateDistribution"
|>|>"UnivarMixingDistribution"          [*]
|>|>|>"UnivarLebDecDistribution"        [*]
|>|>|>|>"AffLinUnivarLebDecDistribution"
|>|>|>"CompoundDistribution"           [*]
|>|>"AbscontDistribution"              [*]
|>|>|>"AffLinAbscontDistribution"
|>|>|>"Arcsine"                        [*]
|>|>|>"Beta"                          [*]
|>|>|>"Cauchy"                        [*]
|>|>|>"ExpOrGammaOrChisq" (VIRTUAL)
|>|>|>|>"Exp"                          [*]
|>|>|>|>"Gammad"                      [*]
|>|>|>|>"Chisq"                       [*]
|>|>|>"Fd"                            [*]
|>|>|>"Lnorm"                         [*]
|>|>|>"Logis"                         [*]
|>|>|>"Norm"                          [*]
|>|>|>"Td"                            [*]
|>|>|>"Unif"                          [*]
|>|>|>"Weibull"                       [*]
|>|>|"DiscreteDistribution"            [*]
|>|>|>"AffLinDiscreteDistribution"
|>|>|>"LatticeDistribution"            [*]
|>|>|>|>"AffLinLatticeDistribution"
|>|>|>|>"Binom"                       [*]
|>|>|>|>"Dirac"                       [*]
|>|>|>|>"Hyper"                       [*]
|>|>|>|>"NBinom"                      [*]
|>|>|>|>|>"Geom"                      [*]
|>|>|>|>|>"Pois"                      [*]
```

```
"AffLinDistribution" = union ( "AffLinAbscontDistribution",
                              "AffLinDiscreteDistribution",
                              "AffLinUnivarLebDecDistribution" )
```

```
"DistrList"
|>"UnivarDistrList"          [*]
```

```
"AcDcLc" = union ( "AbscontDistribution",
                   "DiscreteDistribution",
                   "UnivarLebDecDistribution" )
```

```
#####
Parameter classes
#####
```

```
"OptionalParameter"
|>"Parameter"
|>|>"BetaParameter"
|>|>"BinomParameter"
|>|>"CauchyParameter"
|>|>"ChisqParameter"
|>|>"DiracParameter"
|>|>"ExpParameter"
|>|>"FParameter"
|>|>"GammaParameter"
|>|>"GeomParameter"
|>|>"HyperParameter"
|>|>"LnormParameter"
|>|>"LogisParameter"
|>|>"NbinomParameter"
|>|>"NormParameter"
|>|>"UniNormParameter"
|>|>|>"PoisParameter"
|>|>"TParameter"
|>|>"UnifParameter"
|>|>"WeibullParameter"
```

```
#####
Space classes
#####
```

```
"rSpace"
|>"EuclideanSpace"
|>|>"Reals"
|>"Lattice"
|>"Naturals"
```



```
#####
Symmetry classes
#####
slots:
type(character), SymmCenter(ANY)

"Symmetry"
|>"NoSymmetry"          [*]
|>"EllipticalSymmetry"  [*]
|>|>"SphericalSymmetry" [*]
|>"DistributionSymmetry"
|>"FunctionSymmetry"
|>|>"NonSymmetric"      [*]
|>|>"EvenSymmetric"     [*]
|>|>"OddSymmetric"      [*]

list thereof
"DistrSymmList"          [*]
"FunSymmList"            [*]

#####
Matrix classes
#####
slots:
none
"PosSemDefSymmMatrix" [*] is subclass of class "matrix" of package "base".
|>"PosDefSymmMatrix"  [*]

#####
Class unions
#####
"OptionalNumeric" = union("numeric", "NULL")
"OptionalMatrix" = union("matrix", "NULL")
```

## Methods

The group Math of unary (see [Math](#)) as well as convolution are made available for distributions, see [operators-methods](#); in particular for convolution powers, we have method [convpow](#). Besides, there are plot and print-methods for distributions. For the space classes, we have `liesIn`, for the `DicreteDistribution` class, we have `liesInSupport`, as well as a generating function. The "history" of distributions obtained by chaining operations may be shortened using `simplifyr`.

## Functions

RtoDPQ	Default procedure to fill slots d,p,q given r for a.c. distributions
RtoDPQ.d	Default procedure to fill slots d,p,q given r for discrete distributions

RtoDPQ.LC	Default procedure to fill slots d,p,q given r for Lebesgue decomposed distributions
decomposePM	decomposes a distribution into positive and negative part and, if discrete, into part '0'
simplifyD	tries to reduce/simplify mixing distribution using that certain weights are 0
flat.LCD	makes a single UnivarLebDecDistribution out of a list of UnivarLebDecDistribution with corresp. weights
flat.mix	makes a single UnivarLebDecDistribution out of a list of a UnivarMixingDistribution
distroptions	Functions to change the global variables of the package 'distr'
standardMethods	Utility to automatically generate accessor and replacement functions

### Extension Packages in distrXXX family

Please note that there are extension packages of this packages available on CRAN,

**distrDoc** a documentation package providing joint documentation for all packages of the distrXXX family of packages in the form of vignette 'distr'; try `require(distrDoc); vignette("distr")`.

**distrEx** provides functionals (like E, sd, mad) operating on distributions, as well as distances between distributions and basic support for multivariate and conditional distributions.

**distrSim** for the standardized treatment of simulations, also under contaminations.

**distrTEst** with classes and methods for evaluations of statistical procedures on simulations generated by **distrSim**.

**distrTeach** embodies illustrations for basic stats courses using our distribution classes.

**distrMod** provides classes for parametric models and hence covers, in an object orientated way, estimation in statistical models.

**distrEllipse** provides classes for elliptically symmetric distributions.

### 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.

### Acknowledgement

We thank Martin Maechler, Josef Leydold, John Chambers, Duncan Murdoch, Gregory Warnes, Paul Gilbert, Kurt Hornik, Uwe Ligges, Torsten Hothorn, and Seth Falcon for their help in preparing this package.

### 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.

## Demos

Demos are available — see demo(package="distr")

## Note

Arithmetics on distribution objects are understood as operations on corresponding (independent) r.v.'s and **not** on distribution functions or densities.

See also distrARITH().

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

Accuracy of these arithmetics is controlled by global options which may be inspected / set by distroptions() and getdistrOption(), confer [distroptions](#).

## Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

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

## References

P. Ruckdeschel, M. Kohl, T. Stabla, F. Camphausen (2006): S4 Classes for Distributions, *R News*, 6(2), 2-6. [https://CRAN.R-project.org/doc/Rnews/Rnews\\_2006-2.pdf](https://CRAN.R-project.org/doc/Rnews/Rnews_2006-2.pdf)

P. Ruckdeschel and M. Kohl (2014): General purpose convolution algorithm for distributions in S4-Classes by means of FFT. *J. Statist. Softw.* **59**(4): 1-25.

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/>

**Examples**

```

X <- Unif(2,3)
Y <- Pois(lambda = 3)
Z <- X+Y # generates Law of corresponding independent variables
p(Z)(0.2)
r(Z)(1000)
plot(Z+sin(Norm()))

```

---

AbscontDistribution      *Generating function "AbscontDistribution"*

---

**Description**

Generates an object of class "AbscontDistribution"

**Usage**

```

AbscontDistribution(r = NULL, d = NULL, p = NULL, q = NULL,
  gaps = NULL, param = NULL, img = new("Reals"),
  .withSim = FALSE, .withArith = FALSE,
  .lowerExact = FALSE, .logExact = FALSE,
  withgaps = getdistrOption("withgaps"),
  low1 = NULL, up1 = NULL, low = -Inf, up = Inf,
  withStand = FALSE,
  ngrid = getdistrOption("DefaultNrGridPoints"),
  ep = getdistrOption("TruncQuantile"),
  e = getdistrOption("RtoDPQ.e"),
  Symmetry = NoSymmetry())

```

**Arguments**

r	slot r to be filled
d	slot d to be filled
p	slot p to be filled
q	slot q to be filled
gaps	slot gaps (of class "matrix" with two columns) to be filled (i.e. t(gaps) must be ordered if read as vector)
param	parameter (of class "OptionalParameter")
img	image range of the distribution (of class "rSpace")
low1	lower bound (to be the lower TruncQuantile-quantile of the distribution)
up1	upper bound (to be the upper TruncQuantile-quantile of the distribution)
low	lower bound (to be the 100-percent-quantile of the distribution)
up	upper bound (to be the 100-percent-quantile of the distribution)

withStand	logical: shall we standardize argument function d to integrate to 1 — default is no resp. FALSE
ngrid	number of gridpoints
ep	tolerance epsilon
e	exponent to base 10 to be used for simulations
withgaps	logical; shall gaps be reconstructed empirically?
.withArith	normally not set by the user, but if determining the entries supp, prob distributonal arithmetics was involved, you may set this to TRUE.
.withSim	normally not set by the user, but if determining the entries supp, prob simulations were involved, you may set this to TRUE.
.lowerExact	normally not set by the user: whether the lower .tail=FALSE part is calculated exactly, avoing a “1-.”.
.logExact	normally not set by the user: whether in determining slots d,p,q, we make particular use of a logarithmic representation to enhance accuracy.
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; in this case use Symmetry=SphericalSymmetry(center).

## Details

Typical usages are

```
AbscontDistribution(r)
AbscontDistribution(r = NULL, d)
AbscontDistribution(r = NULL, d = NULL, p)
AbscontDistribution(r = NULL, d = NULL, p = NULL, d)
AbscontDistribution(r, d, p, q)
```

Minimally, only one of the slots r, d, p or q needs to be given as argument. The other non-given slots are then reconstructed according to the following scheme:

r	d	p	q	proceeding
-	-	-	-	excluded
-	+	-	-	p by .D2P, q by .P2Q, r by q(runif(n))
-	-	+	-	d by .P2D, q by .P2Q, r by q(runif(n))
-	+	+	-	q by .P2Q, r by q(runif(n))
-	-	-	+	p by .Q2P, d by .P2D, r by q(runif(n))
-	+	-	+	p by .Q2P, r by q(runif(n))
-	-	+	+	d by .P2D, r by q(runif(n))
-	+	+	+	r by q(runif(n))
+	-	-	-	call to <a href="#">RtoDPQ</a>
+	+	-	-	p by .D2P, q by .P2Q
+	-	+	-	d by .P2D, q by .P2Q
+	+	+	-	q by .P2Q
+	-	-	+	p by .Q2P, d by .P2D

+	+	-	+	p by .Q2P
+	-	+	+	d by .P2D
+	+	+	+	nothing

For this purpose, one may alternatively give arguments low1 and up1 (NULL each by default, and determined through slot q, resp. p, resp. d, resp. r in this order according to availability), for the (finite) range of values in the support of this distribution, as well as the possibly infinite theoretical range given by arguments low and up with default values  $-\text{Inf}$ ,  $\text{Inf}$ , respectively. Of course all other slots may be specified as arguments.

### Value

Object of class "AbscontDistribution"

### Author(s)

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

### See Also

[AbscontDistribution-class](#), [DiscreteDistribution-class](#), [RtoDPQ](#)

### Examples

```
plot(Norm())
plot(AbscontDistribution(r = rnorm))
plot(AbscontDistribution(d = dnorm))
plot(AbscontDistribution(p = pnorm))
plot(AbscontDistribution(q = qnorm))
plot(Ac <- AbscontDistribution(d = function(x, log = FALSE){
  d <- exp(-abs(x^3))
  ## unstandardized!!
  if(log) d <- log(d)
  return(d)},
  withStand = TRUE))
```

---

AbscontDistribution-class

*Class "AbscontDistribution"*

---

### Description

The AbscontDistribution-class is the mother-class of the classes Beta, Cauchy, Chisq, Exp, F, Gammad, Lnorm, Logis, Norm, T, Unif and Weibull. Further absolutely continuous distributions can be defined either by declaration of own random number generator, density, cumulative distribution and quantile functions, or as result of a convolution of two absolutely continuous distributions or by application of a mathematical operator to an absolutely continuous distribution.

## Objects from the Class

Objects can be created by calls of the form `new("AbscontDistribution", r, d, p, q)`. More comfortably, you may use the generating function `AbscontDistribution`. The result of these calls is an absolutely continuous distribution.

## Slots

`img` Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

`param` Object of class "Parameter": the parameter of this distribution, having only the slot name "Parameter of an absolutely continuous distribution"

`r` Object of class "function": generates random numbers

`d` Object of class "function": density function

`p` Object of class "function": cumulative distribution function

`q` Object of class "function": quantile function

`gaps` [from version 1.9 on] Object of class "OptionalMatrix", i.e.; an object which may either be NULL or a matrix. This slot, if non-NULL, contains left and right endpoints of intervals where the density of the object is 0. This slot may be inspected by the accessor `gaps()` and modified by a corresponding replacement method. It may also be filled automatically by `setgaps()`. For saved objects from earlier versions, we provide functions `isOldVersion` and `conv2NewVersion`.

`.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

**initialize** signature(.Object = "AbscontDistribution"): initialize method

**Math** signature(x = "AbscontDistribution"): application of a mathematical function, e.g. `sin` or `exp` (does not work with `log`, `sign`!), to this absolutely continuous distribution

- `abs`: signature(x = "AbscontDistribution"): exact image distribution of `abs(x)`.
- `exp`: signature(x = "AbscontDistribution"): exact image distribution of `exp(x)`.
- `sign`: signature(x = "AbscontDistribution"): exact image distribution of `sign(x)`.
- `sqrt`: signature(x = "AbscontDistribution"): exact image distribution of `sqrt(x)`.

- `log`: `signature(x = "AbscontDistribution")`: (with optional further argument `base`, defaulting to `exp(1)`) exact image distribution of `log(x)`.
- `log10`: `signature(x = "AbscontDistribution")`: exact image distribution of `log10(x)`.
- `gamma`: `signature(x = "AbscontDistribution")`: exact image distribution of `gamma(x)`.
- `lgamma`: `signature(x = "AbscontDistribution")`: exact image distribution of `lgamma(x)`.
- `digamma`: `signature(x = "AbscontDistribution")`: exact image distribution of `digamma(x)`.
- `sqrt`: `signature(x = "AbscontDistribution")`: exact image distribution of `sqrt(x)`.
- `signature(e1 = "AbscontDistribution")`: application of `'-'` to this absolutely continuous distribution.
- \* `signature(e1 = "AbscontDistribution", e2 = "numeric")`: multiplication of this absolutely continuous distribution by an object of class `"numeric"`
- / `signature(e1 = "AbscontDistribution", e2 = "numeric")`: division of this absolutely continuous distribution by an object of class `"numeric"`
- + `signature(e1 = "AbscontDistribution", e2 = "numeric")`: addition of this absolutely continuous distribution to an object of class `"numeric"`.
- `signature(e1 = "AbscontDistribution", e2 = "numeric")`: subtraction of an object of class `"numeric"` from this absolutely continuous distribution.
- \* `signature(e1 = "numeric", e2 = "AbscontDistribution")`: multiplication of this absolutely continuous distribution by an object of class `"numeric"`.
- + `signature(e1 = "numeric", e2 = "AbscontDistribution")`: addition of this absolutely continuous distribution to an object of class `"numeric"`.
- `signature(e1 = "numeric", e2 = "AbscontDistribution")`: subtraction of this absolutely continuous distribution from an object of class `"numeric"`.
- + `signature(e1 = "AbscontDistribution", e2 = "AbscontDistribution")`: Convolution of two absolutely continuous distributions. The slots `p`, `d` and `q` are approximated by grids.
- `signature(e1 = "AbscontDistribution", e2 = "AbscontDistribution")`: Convolution of two absolutely continuous distributions. The slots `p`, `d` and `q` are approximated by grids.
- plot** `signature(object = "AbscontDistribution")`: plots density, cumulative distribution and quantile function.

### Internal subclass "AffLinAbscontDistribution"

To enhance accuracy of several functionals on distributions, mainly from package **distrEx**, from version 1.9 of this package on, there is an internally used (but exported) subclass `"AffLinAbscontDistribution"` which has extra slots `a`, `b` (both of class `"numeric"`), and `X0` (of class `"AbscontDistribution"`), to capture the fact that the object has the same distribution as  $a * X0 + b$ . This is the class of the return value of methods

- `signature(e1 = "AbscontDistribution")`
- \* `signature(e1 = "AbscontDistribution", e2 = "numeric")`
- / `signature(e1 = "AbscontDistribution", e2 = "numeric")`
- + `signature(e1 = "AbscontDistribution", e2 = "numeric")`
- `signature(e1 = "AbscontDistribution", e2 = "numeric")`



```

* signature(e1 = "numeric", e2 = "AbscontDistribution")
+ signature(e1 = "numeric", e2 = "AbscontDistribution")
- signature(e1 = "numeric", e2 = "AbscontDistribution")
- signature(e1 = "AffLinAbscontDistribution")
* signature(e1 = "AffLinAbscontDistribution", e2 = "numeric")
/ signature(e1 = "AffLinAbscontDistribution", e2 = "numeric")
+ signature(e1 = "AffLinAbscontDistribution", e2 = "numeric")
- signature(e1 = "AffLinAbscontDistribution", e2 = "numeric")
* signature(e1 = "numeric", e2 = "AffLinAbscontDistribution")
+ signature(e1 = "numeric", e2 = "AffLinAbscontDistribution")
- signature(e1 = "numeric", e2 = "AffLinAbscontDistribution")

```

There also is a class union of "AffLinAbscontDistribution", "AffLinDiscreteDistribution", "AffLinUnivarLebDecDistribution" and called "AffLinDistribution" which is used for functionals.

### Internal virtual superclass "AcDcLcDistribution"

As many operations should be valid no matter whether the operands are of class "AbscontDistribution", "DiscreteDistribution", or "UnivarLebDecDistribution", there is a class union of these classes called "AcDcLcDistribution"; in particular methods for "\*", "/", "^" (see [operators-methods](#)) and methods [Minimum](#), [Maximum](#), [Truncate](#), and [Huberize](#), and [convpow](#) are defined for this class union.

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[AbscontDistribution](#) [Parameter-class](#) [UnivariateDistribution-class](#) [Beta-class](#) [Cauchy-class](#) [Chisq-class](#) [Exp-class](#) [Fd-class](#) [Gammad-class](#) [Lnorm-class](#) [Logis-class](#) [Norm-class](#) [Td-class](#) [Unif-class](#) [Weibull-class](#) [DiscreteDistribution-class](#) [Reals-class](#) [RtoDPQ](#)

### Examples

```

N <- Norm() # N is a normal distribution with mean=0 and sd=1.
E <- Exp() # E is an exponential distribution with rate=1.
A1 <- E+1 # a new absolutely continuous distributions with exact slots d, p, q
A2 <- A1*3 # a new absolutely continuous distributions with exact slots d, p, q
A3 <- N*0.9 + E*0.1 # a new absolutely continuous distribution with approximated slots d, p, q
r(A3)(1) # one random number generated from this distribution, e.g. -0.7150937
d(A3)(0) # The (approximated) density for x=0 is 0.43799.
p(A3)(0) # The (approximated) probability that x <= 0 is 0.45620.
q(A3)(.1) # The (approximated) 10 percent quantile is -1.06015.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)

```

Arcsine-class

Class "Arcsine"

**Description**

The Arcsine distribution has density

$$f(x) = \frac{1}{\pi\sqrt{1-x^2}}$$

for  $-1 < x < 1$ .

**Objects from the Class**

Objects can be created by calls of the form `Arcsine()`. This object is an Arcsine distribution.

**Slots**

`img` Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

`r` Object of class "function": generates random numbers (calls function `rArcsine`)

`d` Object of class "function": density function (calls function `dArcsine`)

`p` Object of class "function": cumulative function (calls function `pArcsine`)

`q` Object of class "function": inverse of the cumulative function (calls function `qArcsine`)

`.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

**Methods**

**initialize** signature(.Object = "Arcsine"): initialize method

**Author(s)**

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

**See Also**

[AbscontDistribution-class Reals-class](#)

**Examples**

```
A <- Arcsine()
# A is a Arcsine distribution with shape1 = 1 and shape2 = 1.
r(A)(3) # three random number generated from this distribution, e.g. 0.6979795
d(A)(c(-2,-1,-0.2,0,0.2,1,2)) # Density at x=c(-1,-0.2,0,0.2,1).
p(A)(c(-2,-1,-0.2,0,0.2,1,2)) # cdf at q=c(-1,-0.2,0,0.2,1).
q(A)(c(0,0.2,1,2)) # quantile function at at x=c(0,0.2,1).
## in RStudio or Jupyter IRKernel, use q.l(A)(c(0,0.2,1,2)) instead
```

---

Beta-class

Class "Beta"

---

**Description**

The Beta distribution with parameters  $\text{shape1} = a$  and  $\text{shape2} = b$  has density

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1-x)^{b-1}$$

for  $a > 0$ ,  $b > 0$  and  $0 \leq x \leq 1$  where the boundary values at  $x = 0$  or  $x = 1$  are defined as by continuity (as limits).

**Ad hoc methods**

For R Version <2.3.0 ad hoc methods are provided for slots `q`, `r` if `ncp!=0`; for R Version >=2.3.0 the methods from package **stats** are used.

**Objects from the Class**

Objects can be created by calls of the form `Beta(shape1, shape2)`. This object is a beta distribution.

**Slots**

`img` Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "BetaParameter": the parameter of this distribution (`shape1` and `shape2`), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `rbeta`)

`d` Object of class "function": density function (calls function `dbeta`)

`p` Object of class "function": cumulative function (calls function `pbeta`)

`q` Object of class "function": inverse of the cumulative function (calls function `qbeta`)

.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

### Methods

**initialize** signature(.Object = "Beta"): initialize method

**shape1** signature(object = "Beta"): returns the slot shape1 of the parameter of the distribution

**shape1<-** signature(object = "Beta"): modifies the slot shape1 of the parameter of the distribution

**shape2** signature(object = "Beta"): returns the slot shape2 of the parameter of the distribution

**shape2<-** signature(object = "Beta"): modifies the slot shape2 of the parameter of the distribution

- signature(e1 = "numeric", e2 = "Beta") if ncp(e2)==0 and e1 == 1, an exact (central) Beta(shape1 = shape2(e2), shape2 = shape1(e2)) is returned, else the default method is used; exact

### Note

The non-central Beta distribution is defined (Johnson et al, 1995, pp. 502) as the distribution of  $X/(X + Y)$  where  $X \sim \chi^2_{2a}(\lambda)$  and  $Y \sim \chi^2_{2b}$ . C.f. [rbeta](#)

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[BetaParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rbeta](#)

**Examples**

```

B <- Beta(shape1 = 1, shape2 = 1)
# B is a beta distribution with shape1 = 1 and shape2 = 1.
r(B)(1) # one random number generated from this distribution, e.g. 0.6979795
d(B)(1) # Density of this distribution is 1 for x=1.
p(B)(1) # Probability that x < 1 is 1.
q(B)(.1) # Probability that x < 0.1 is 0.1.
shape1(B) # shape1 of this distribution is 1.
shape1(B) <- 2 # shape1 of this distribution is now 2.
Bn <- Beta(shape1 = 1, shape2 = 3, ncp = 5)
# Bn is a beta distribution with shape1 = 1 and shape2 = 3 and ncp = 5.
B0 <- Bn; ncp(B0) <- 0;
# B0 is just the same beta distribution as Bn but with ncp = 0
q(B0)(0.1) ##
q(Bn)(0.1) ## => from R 2.3.0 on ncp no longer ignored...
## in RStudio or Jupyter IRKernel, use q.l(.)(.) instead of q(.)(.)

```

---

BetaParameter-class	Class "BetaParameter"
---------------------	-----------------------

---

**Description**

The parameter of a beta distribution, used by Beta-class

**Objects from the Class**

Objects can be created by calls of the form `new("BetaParameter", shape1, shape2, ncp)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Beta` is instantiated.

**Slots**

**shape1** Object of class "numeric": the shape1 of a beta distribution  
**shape2** Object of class "numeric": the shape2 of a beta distribution  
**ncp** Object of class "numeric": the noncentrality parameter of a beta distribution  
**name** Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "BetaParameter"): initialize method  
**shape1** signature(object = "BetaParameter"): returns the slot shape1 of the parameter of the distribution

**shape1**`<-` signature(object = "BetaParameter"): modifies the slot shape1 of the parameter of the distribution

**shape2** signature(object = "BetaParameter"): returns the slot shape2 of the parameter of the distribution

**shape2**`<-` signature(object = "BetaParameter"): modifies the slot shape2 of the parameter of the distribution

**ncp** signature(object = "BetaParameter"): returns the slot ncp of the parameter of the distribution

**ncp**`<-` signature(object = "BetaParameter"): modifies the slot ncp of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Beta-class](#) [Parameter-class](#)

### Examples

```
W <- new("BetaParameter", shape1 = 1, shape2 = 1, ncp = 0)
shape2(W) # shape2 of this distribution is 1.
shape2(W) <- 2 # shape2 of this distribution is now 2.
```

---

Binom-class

*Class "Binom"*

---

### Description

The binomial distribution with size =  $n$ , by default = 1, and prob =  $p$ , by default = 0.5, has density

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

for  $x = 0, \dots, n$ .

C.f. [rbinom](#)

### Objects from the Class

Objects can be created by calls of the form Binom(prob, size). This object is a binomial distribution.

**Slots**

**img** Object of class "Naturals": The space of the image of this distribution has got dimension 1 and the name "Natural Space".

**param** Object of class "BinomParameter": the parameter of this distribution (prob, size), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function rbinom)

**d** Object of class "function": density function (calls function dbinom)

**p** Object of class "function": cumulative function (calls function pbinom)

**q** Object of class "function": inverse of the cumulative function (calls function qbinom). The quantile is defined as the smallest value  $x$  such that  $F(x) \geq p$ , where  $F$  is the cumulative function.

**support** Object of class "numeric": a (sorted) vector containing the support of the discrete density 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 "DiscreteDistribution", directly.

Class "UnivariateDistribution", by class "DiscreteDistribution".

Class "Distribution", by class "DiscreteDistribution".

**Methods**

**+** signature(e1 = "Binom", e2 = "Binom"): For two binomial distributions with equal probabilities the exact convolution formula is implemented thereby improving the general numerical accuracy.

**initialize** signature(.Object = "Binom"): initialize method

**prob** signature(object = "Binom"): returns the slot prob of the parameter of the distribution

**prob<-** signature(object = "Binom"): modifies the slot prob of the parameter of the distribution

**size** signature(object = "Binom"): returns the slot size of the parameter of the distribution

**size<-** signature(object = "Binom"): modifies the slot size of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[BinomParameter-class](#) [DiscreteDistribution-class](#) [Naturals-class](#) [rbinom](#)

**Examples**

```
B <- Binom(prob=0.5,size=1) # B is a binomial distribution with prob=0.5 and size=1.
r(B)(1) # # one random number generated from this distribution, e.g. 1
d(B)(1) # Density of this distribution is 0.5 for x=1.
p(B)(0.4) # Probability that x<0.4 is 0.5.
q(B)(.1) # x=0 is the smallest value x such that p(B)(x)>=0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
size(B) # size of this distribution is 1.
size(B) <- 2 # size of this distribution is now 2.
C <- Binom(prob = 0.5, size = 1) # C is a binomial distribution with prob=0.5 and size=1.
D <- Binom(prob = 0.6, size = 1) # D is a binomial distribution with prob=0.6 and size=1.
E <- B + C # E is a binomial distribution with prob=0.5 and size=3.
F <- B + D # F is an object of class LatticeDistribution.
G <- B + as(D,"DiscreteDistribution") ## DiscreteDistribution
```

---

BinomParameter-class    *Class "BinomParameter"*

---

**Description**

The parameter of a binomial distribution, used by Binom-class

**Objects from the Class**

Objects can be created by calls of the form `new("BinomParameter",prob,size)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Binom` is instantiated.

**Slots**

`prob` Object of class "numeric": the probability of a binomial distribution  
`size` Object of class "numeric": the size of a binomial distribution  
`name` Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.



**Methods**

**initialize** signature(.Object = "BinomParameter"): initialize method

**prob** signature(object = "BinomParameter"): returns the slot prob of the parameter of the distribution

**prob<-** signature(object = "BinomParameter"): modifies the slot prob of the parameter of the distribution

**size** signature(object = "BinomParameter"): returns the slot size of the parameter of the distribution

**size<-** signature(object = "BinomParameter"): modifies the slot size of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Binom-class Parameter-class](#)

**Examples**

```
W <- new("BinomParameter",prob=0.5,size=1)
size(W) # size of this distribution is 1.
size(W) <- 2 # size of this distribution is now 2.
```

---

Cauchy-class

---

Class "Cauchy"

---

**Description**

The Cauchy distribution with location  $l$ , by default = 0, and scale  $s$ , by default = 1, has density

$$f(x) = \frac{1}{\pi s} \left( 1 + \left( \frac{x-l}{s} \right)^2 \right)^{-1}$$

for all  $x$ . C.f. [rcauchy](#)

**Objects from the Class**

Objects can be created by calls of the form `Cauchy(location, scale)`. This object is a Cauchy distribution.

**Slots**

**img** Object of class "Reals": The domain of this distribution has got dimension 1 and the name "Real Space".

**param** Object of class "CauchyParameter": the parameter of this distribution (location and scale), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function rcauchy)

**d** Object of class "function": density function (calls function dcauchy)

**p** Object of class "function": cumulative function (calls function pcauchy)

**q** Object of class "function": inverse of the cumulative function (calls function qcauchy)

**.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

**Is-Relations**

By means of setIs, R "knows" that a distribution object obj of class "Cauchy" with location 0 and scale 1 also is a T distribution with parameters  $df = 1$ ,  $n_{cp} = 0$ .

**Methods**

**initialize** signature(.Object = "Cauchy"): initialize method

**location** signature(object = "Cauchy"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "Cauchy"): modifies the slot location of the parameter of the distribution

**scale** signature(object = "Cauchy"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Cauchy"): modifies the slot scale of the parameter of the distribution

**+** signature(e1 = "Cauchy", e2 = "Cauchy"): For the Cauchy distribution the exact convolution formula is implemented thereby improving the general numerical approximation.

**\*** signature(e1 = "Cauchy", e2 = "numeric")

**+** signature(e1 = "Cauchy", e2 = "numeric"): For the Cauchy location scale family we use its closedness under affine linear transformations.

further arithmetic methods see [operators-methods](#)

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[CauchyParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rcauchy](#)

**Examples**

```
C <- Cauchy(location = 1, scale = 1) # C is a Cauchy distribution with location=1 and scale=1.
r(C)(1) # one random number generated from this distribution, e.g. 4.104603
d(C)(1) # Density of this distribution is 0.3183099 for x=1.
p(C)(1) # Probability that x<1 is 0.5.
q(C)(.1) # Probability that x<-2.077684 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
location(C) # location of this distribution is 1.
location(C) <- 2 # location of this distribution is now 2.
is(C,"Td") # no
C0 <- Cauchy() # standard, i.e. location = 0, scale = 1
is(C0,"Td") # yes
as(C0,"Td")
```

---

CauchyParameter-class *Class "CauchyParameter"*

---

**Description**

The parameter of a Cauchy distribution, used by Cauchy-class

**Objects from the Class**

Objects can be created by calls of the form `new("CauchyParameter", location, scale)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Cauchy` is instantiated.

**Slots**

**location:** Object of class "numeric": the location of a Cauchy distribution  
**scale** Object of class "numeric": the scale of a Cauchy distribution  
**name** Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "CauchyParameter"): initialize method

**scale** signature(object = "CauchyParameter"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "CauchyParameter"): modifies the slot scale of the parameter of the distribution

**location** signature(object = "CauchyParameter"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "CauchyParameter"): modifies the slot location of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Cauchy-class Parameter-class](#)

**Examples**

```
W <- new("CauchyParameter",location=1,scale=1)
location(W) # location of this distribution is 1.
location(W) <- 2 # location of this distribution is now 2.
```

---

Chisq-class

Class "Chisq"

---

**Description**

The chi-squared distribution with  $df = n$  degrees of freedom has density

$$f_n(x) = \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2}$$

for  $x > 0$ . The mean and variance are  $n$  and  $2n$ .

The non-central chi-squared distribution with  $df = n$  degrees of freedom and non-centrality parameter  $n\mu = \lambda$  has density

$$f(x) = e^{-\lambda/2} \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{r!} f_{n+2r}(x)$$

for  $x \geq 0$ . For integer  $n$ , this is the distribution of the sum of squares of  $n$  normals each with variance one,  $\lambda$  being the sum of squares of the normal means.

C.f. [rchisq](#)

## Objects from the Class

Objects can be created by calls of the form `Chisq(df, ncp)`. This object is a chi-squared distribution.

## Slots

**img** Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

**param** Object of class "ChisqParameter": the parameter of this distribution (df and ncp), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function `rchisq`)

**d** Object of class "function": density function (calls function `dchisq`)

**p** Object of class "function": cumulative function (calls function `pchisq`)

**q** Object of class "function": inverse of the cumulative function (calls function `qchisq`)

**.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 "ExpOrGammaOrChisq", directly.

Class "AbscontDistribution", by class "ExpOrGammaOrChisq".

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "UnivariateDistribution".

## Is-Relations

By means of `setIs`, R "knows" that a distribution object `obj` of class "Chisq" with non-centrality 0 also is a Gamma distribution with parameters `shape = df(obj)/2, scale = 2`.

## Methods

**initialize** signature(.Object = "Chisq"): initialize method

**df** signature(object = "Chisq"): returns the slot `df` of the parameter of the distribution

**df<-** signature(object = "Chisq"): modifies the slot `df` of the parameter of the distribution

**ncp** signature(object = "Chisq"): returns the slot `ncp` of the parameter of the distribution

**ncp<-** signature(object = "Chisq"): modifies the slot `ncp` of the parameter of the distribution

**+** signature(e1 = "Chisq", e2 = "Chisq"): For the chi-squared distribution we use its closedness under convolutions.

**Note**

Warning: The code for pchisq and qchisq is unreliable for values of ncp above approximately 290.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[ChisqParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rchisq](#)

**Examples**

```
C <- Chisq(df = 1, ncp = 1) # C is a chi-squared distribution with df=1 and ncp=1.
r(C)(1) # one random number generated from this distribution, e.g. 0.2557184
d(C)(1) # Density of this distribution is 0.2264666 for x = 1.
p(C)(1) # Probability that x < 1 is 0.4772499.
q(C)(.1) # Probability that x < 0.04270125 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
df(C) # df of this distribution is 1.
df(C) <- 2 # df of this distribution is now 2.
is(C, "Gammad") # no
C0 <- Chisq() # default: Chisq(df=1,ncp=0)
is(C0, "Gammad") # yes
as(C0, "Gammad")
```

---

ChisqParameter-class    *Class "ChisqParameter"*

---

**Description**

The parameter of a chi-squared distribution, used by Chisq-class

**Objects from the Class**

Objects can be created by calls of the form `new("ChisqParameter",ncp,df)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Chisq` is instantiated.

**Slots**

`ncp` Object of class "numeric": the ncp of a chi-squared distribution  
`df` Object of class "numeric": the df of a chi-squared distribution  
`name` Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "ChisqParameter"): initialize method

**df** signature(object = "ChisqParameter"): returns the slot df of the parameter of the distribution

**df<-** signature(object = "ChisqParameter"): modifies the slot df of the parameter of the distribution

**ncp** signature(object = "ChisqParameter"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "ChisqParameter"): modifies the slot ncp of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Chisq-class Parameter-class](#)

**Examples**

```
W <- new("ChisqParameter",df=1,ncp=1)
ncp(W) # ncp of this distribution is 1.
ncp(W) <- 2 # ncp of this distribution is now 2.
```

---

CompoundDistribution    *Generating function for Class "CompoundDistribution"*

---

**Description**

Generates an object of class "CompoundDistribution".

**Usage**

```
CompoundDistribution(NumbOfSummandsDistr, SummandsDistr, .withSim = FALSE,
                    withSimplify = FALSE)
```

**Arguments**

NumbOfSummandsDistr	Object of class "DiscreteDistribution", the frequency distribution; it is checked that support is contained in 0,1,2,...
SummandsDistr	Object of class "UnivDistrListOrDistribution", that is, either of class "UnivarDistrList" (non i.i.d. case) or of class "UnivariateDistribution" (i.i.d. case); the summand distribution(s).
.withSim	logical; value of the corresponding slot.
withSimplify	"logical": shall the return value be piped through a call to simplifyD?

**Value**

Object of class "CompoundDistribution", or if argument withSimplify is TRUE the result of [simplifyD](#) applied to the compound distribution, i.e. an object of class "UnivarLebDecDistribution", or if degenerate, of class "AbscontDistribution" or "DiscreteDistribution".

**Author(s)**

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

**See Also**

[CompoundDistribution-class](#), [simplifyD](#)

**Examples**

```
CP0 <- CompoundDistribution(Pois(), Norm())
CP0
CP1 <- CompoundDistribution(DiscreteDistribution(supp = c(1,5,9,11),
  prob = dbinom(0:3, size = 3, prob = 0.3)), Norm())
CP1
UL <- UnivarDistrList(Norm(), Binom(10,0.3), Chisq(df=4), Norm(),
  Binom(10,0.3), Chisq(df=4), Norm(), Binom(10,0.3),
  Chisq(df=4), Td(5), Td(10))
CP2 <- CompoundDistribution(DiscreteDistribution(supp = c(1,5,9,11),
  prob = dbinom(0:3, size = 3, prob = 0.3)), UL)
plot(CP2)
```

---

CompoundDistribution-class

*Class "CompoundDistribution"*

---

**Description**

CompoundDistribution-class is a class to formalize compound distributions; it is a subclass to class UnivarMixingDistribution.



## Objects from the Class

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

## Slots

**NumbOfSummandsDistr** Object of class "DiscreteDistribution", the frequency distribution.

**SummandsDistr** Object of class "UnivDistrListOrDistribution", that is, either of class "UnivarDistrList" (non i.i.d. case) or of class "UnivariateDistribution" (i.i.d. case); the summand distribution(s).

**mixCoeff** Object of class "numeric": a vector of probabilities for the mixing components.

**mixDistr** Object of class "UnivarDistrList": a list of univariate distributions containing the mixing components; must be of same length as **mixCoeff**.

**img** Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

**param** Object of class "Parameter": the parameter of this distribution, having only the slot name "Parameter of a discrete distribution"

**r** Object of class "function": generates random numbers

**d** fixed to NULL

**p** Object of class "function": cumulative distribution function

**q** Object of class "function": 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 "UnivarMixingDistribution" class "UnivarDistribution" by class "UnivarMixingDistribution", class "Distribution" by class "UnivariateDistribution".

## Methods

**show** `signature(object = "CompoundDistribution")` prints the object

**SummandsDistr** `signature(object = "CompoundDistribution")` returns the corresponding slot

**NumbOfSummandsDistr** `signature(object = "CompoundDistribution")` returns the corresponding slot

**setAs relations**

There is a coerce method to coerce objects of class "CompoundDistribution" to class UnivarLebDecDistribution; this is done by a simple call to simplifyD.

**Author(s)**

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

**See Also**

[Parameter-class](#), [UnivariateDistribution-class](#), [LatticeDistribution-class](#), [AbscontDistribution-class](#), [simplifyD](#), [flat.mix](#)

**Examples**

```
CP <- CompoundDistribution(Pois(),Norm())
CP
p(CP)(0.3)
plot(CP)
```

---

convpow-methods

*Distribution of the sum of univariate i.i.d r.v's*


---

**Description**

Method convpow determines the distribution of the sum of N univariate i.i.d r.v's by means of DFT

**Usage**

```
convpow(D1,...)
## S4 method for signature 'AbscontDistribution'
convpow(D1,N)
## S4 method for signature 'LatticeDistribution'
convpow(D1,N,
        ep = getdistrOption("TruncQuantile"))
## S4 method for signature 'DiscreteDistribution'
convpow(D1,N)
## S4 method for signature 'AcDcLcDistribution'
convpow(D1,N,
        ep = getdistrOption("TruncQuantile"))
```

**Arguments**

D1	an object of (a sub)class (of) "AbscontDistribution" or "LatticeDistribution" or of "UnivarLebDecDistribution"
...	not yet used; meanwhile takes up N
N	an integer or 0 (for 0 returns Dirac(0), for 1 D1)

`ep` numeric of length 1 in (0,1) — for "LatticeDistribution": support points will be cancelled if their probability is less than `ep`; for "UnivarLebDecDistribution": if `(acWeight(object)<ep)` we work with the discrete parts only, and, similarly, if `(discreteWeight(object)<ep)` we with the absolutely continuous parts only.

### Details

in the methods implemented a second argument `N` is obligatory; the general methods use a general purpose convolution algorithm for distributions by means of D/FFT. In case of an argument of class "UnivarLebDecDistribution", the result will in generally be again of class "UnivarLebDecDistribution". However, if `acWeight(D1)` is positive, `discreteWeight(convpow(D1,N))` will decay exponentially in `N`, hence from some (small)  $N_0$  on, the result will be of class "AbscontDistribution". This is used algorithmically, too, as then only the a.c. part needs to be convolved. In case of an argument `D1` of class "DiscreteDistribution", for `N` equal to 0,1 we return the obvious solutions, and for `N==2` the return value is `D1+D1`. For `N>2`, we split up `N` into `N=N1+N2`, `N1=floor(N/2)` and recursively return `convpow(D1,N1)+convpow(D1,N2)`.

### Value

Object of class "AbscontDistribution", "DiscreteDistribution", "LatticeDistribution" resp. "AcDcLcDistribution"

### further S4-Methods

There are particular methods for the following classes, using explicit convolution formulae:

`signature(D1="Norm")` returns class "Norm"  
`signature(D1="Nbinom")` returns class "Nbinom"  
`signature(D1="Binom")` returns class "Binom"  
`signature(D1="Cauchy")` returns class "Cauchy"  
`signature(D1="ExpOrGammaOrChisq")` returns class "Gammad" —if `D1` may be coerced to `Gammad`  
`signature(D1="Pois")` returns class "Pois"  
`signature(D1="Dirac")` returns class "Dirac"

### Author(s)

Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>  
 Matthias Kohl <matthias.kohl@stamats.de> Thomas Stabla <statho3@web.de>

### References

Kohl, M., Ruckdeschel, P., (2014): General purpose convolution algorithm for distributions in S4-Classes by means of FFT. *J. Statist. Softw.* **59**(4): 1-25.

### See Also

[operators](#), `distrARITH()`

Examples

```
convpow(Exp()+Pois(),4)
```

---

d-methods	<i>Methods for Function d in Package ‘distr’</i>
-----------	--

---

Description

```
d-methods
```

Methods

```
d signature(object = "Distribution"): returns the density function
```

See Also

```
Distribution-class
```

---

decomposePM-methods	<i>Methods for function decomposePM in Package ‘distr’</i>
---------------------	--

---

Description

```
decomposePM-methods
```

Usage

```
decomposePM(object)
```

Arguments

```
object          Abscont-/Discrete-/UnivarLebDec-Distribution object
```

Details

There are particular return types for the following classes

**"AbscontDistribution"** a list with components "neg" and "pos" for the respective negative and positive part; each of these parts in its turn is a list with components D for the distribution (in this case of class "AbscontDistribution" again) and w for the weight of the respective part; if the weight of the negative part is 0, the corresponding distribution is set to `-abs(Norm())`, and respectively, if the weight of the positive part is 0, the corresponding distribution is set to `abs(Norm())`.

**"DiscreteDistribution"** a list with components "neg", "pos" and "0" for the respective negative, positive and zero part; each of these parts in its turn is a list with components D for the distribution (in this case of class "DiscreteDistribution" again) and w for the weight of the respective part; while the distribution of the zero part is always  $\text{Dirac}(0)$ , if the weight of the negative part is 0, the corresponding distribution is set to  $\text{Dirac}(-1)$ , and respectively, if the weight of the positive part is 0, the corresponding distribution is set to  $\text{Dirac}(1)$ .

**"UnivarLebDecDistribution"** a list with components "neg", "pos" and "0" for the respective negative, positive and zero part; each of these parts in its turn is a list with components D for the distribution (in case of components "neg", "pos" of class "UnivarLebDecDistribution" again, while the distribution of the zero part is always  $\text{Dirac}(0)$ ) and w for the weight of the respective part; it is build up by calling `decomposePM` for `acPart(object)` and `discretePart(object)` separately, hence if weights of some parts are zero the corresponding procedure mentioned for these methods applies.

Method `decomposePM` is used by our multiplication, division and exponentiation ("\*", "/" "^") - methods.

### Value

the positive and negative part of the distribution together with corresponding weights as a list.

### See Also

[AbscontDistribution-class](#), [DiscreteDistribution-class](#), [UnivarLebDecDistribution-class](#), [operators-methods](#)

### Examples

```
decomposePM(Norm())
decomposePM(Binom(2,0.3)-Binom(5,.4))
decomposePM(UnivarLebDecDistribution(Norm(),Binom(2,0.3)-Binom(5,.4),
                                     acWeight = 0.3))
```

---

DExp-class

Class "DExp"

---

### Description

The double exponential or Laplace distribution with rate  $\lambda$  has density

$$f(x) = \frac{1}{2} \lambda e^{-\lambda|x|}$$

C.f. [Exp-class](#), [rexp](#)

### Objects from the Class

Objects can be created by calls of the form `DExp(rate)`. This object is a double exponential (or Laplace) distribution.

**Slots**

**img** Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

**param** Object of class "ExpParameter": the parameter of this distribution (rate), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function `rexp`)

**d** Object of class "function": density function (calls function `dexp`)

**p** Object of class "function": cumulative function (calls function `pexp`)

**q** Object of class "function": inverse of the cumulative function (calls function `qexp`)

**.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 "AbscontDistribution", directly.

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

**Methods**

**initialize** signature(.Object = "DExp"): initialize method

**rate** signature(object = "DExp"): returns the slot rate of the parameter of the distribution

**rate<-** signature(object = "DExp"): modifies the slot rate of the parameter of the distribution

**\*** signature(e1 = "DExp", e2 = "numeric"): For the Laplace distribution we use its closedness under scaling transformations.

**Author(s)**

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

**See Also**

[Exp-class](#) [ExpParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rexp](#)

**Examples**

```

D <- DExp(rate = 1) # D is a Laplace distribution with rate = 1.
r(D)(1) # one random number generated from this distribution, e.g. 0.4190765
d(D)(1) # Density of this distribution is 0.1839397 for x = 1.
p(D)(1) # Probability that x < 1 is 0.8160603.
q(D)(.1) # Probability that x < -1.609438 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
rate(D) # rate of this distribution is 1.
rate(D) <- 2 # rate of this distribution is now 2.
3*D ### still a DExp -distribution

```

df-methods

*Methods for Function df in Package 'distr'***Description**

df-methods

**Methods**

```

df signature(object = "TParameter"): returns the slot df of the parameter of the distribution
df<- signature(object = "TParameter"): modifies the slot df of the parameter of the distribution
df signature(object = "Td"): returns the slot df of the parameter of the distribution
df<- signature(object = "Td"): modifies the slot df of the parameter of the distribution
df signature(object = "ChisqParameter"): returns the slot df of the parameter of the distribution
df<- signature(object = "ChisqParameter"): modifies the slot df of the parameter of the distribution
df signature(object = "Chisq"): returns the slot df of the parameter of the distribution
df<- signature(object = "Chisq"): modifies the slot df of the parameter of the distribution

```

df1-methods

*Methods for Function df1 in Package 'distr'***Description**

df-methods

**Methods**

```

df1 signature(object = "FParameter"): returns the slot df1 of the parameter of an F-distribution
df1<- signature(object = "FParameter"): modifies the slot df1 of the parameter of an F-distribution
df1 signature(object = "Fd"): returns the slot df1 of the slot param of the distribution
df1<- signature(object = "Fd"): modifies the slot df1 of the slot param of the distribution

```

---

df2-methods	<i>Methods for Function df2 in Package ‘distr’</i>
-------------	--

---

**Description**

df-methods

**Methods**

- df2** signature(object = "FParameter"): returns the slot df2 of the parameter of an F-distribution
- df2<-** signature(object = "FParameter"): modifies the slot df2 of the parameter of an F-distribution
- df2** signature(object = "Fd"): returns the slot df2 of the slot param of the distribution
- df2<-** signature(object = "Fd"): modifies the slot df2 of the slot param of the distribution

---

dim-methods	<i>Methods for Function dim in Package ‘distr’</i>
-------------	--

---

**Description**

dim-methods

**Methods**

- dim** signature(object = "UnivariateDistribution"): returns the dimension of the distribution

**See Also**

[UnivariateDistribution-class](#)

---

dimension-methods	<i>Methods for Function dimension in Package ‘distr’</i>
-------------------	--

---

**Description**

dimension-methods

**Methods**

- dimension** signature(object = "EuclideanSpace"): returns the dimension of the space
- dimension<-** signature(object = "EuclideanSpace"): modifies the dimension of the space



Dirac-class

Class "Dirac"

**Description**

The Dirac distribution with location  $l$ , by default  $= 0$ , has density  $d(x) = 1$  for  $x = l$ , 0 else.

**Objects from the Class**

Objects can be created by calls of the form `Dirac(location)`. This object is a Dirac distribution.

**Slots**

`img` Object of class "Naturals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "DiracParameter": the parameter of this distribution (location), declared at its instantiation

`r` Object of class "function": generates random numbers

`d` Object of class "function": density function

`p` Object of class "function": cumulative function

`q` Object of class "function": inverse of the cumulative function

`support` Object of class "numeric": a (sorted) vector containing the support of the discrete density 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 "DiscreteDistribution", directly.

Class "UnivariateDistribution", by class "DiscreteDistribution".

Class "Distribution", by class "DiscreteDistribution".

**Methods**

- signature(e1 = "Dirac", e2 = "Dirac")

+ signature(e1 = "Dirac", e2 = "Dirac")

\* signature(e1 = "Dirac", e2 = "Dirac")

/ signature(e1 = "Dirac", e2 = "Dirac"): For the Dirac distribution these operations are trivial.

**initialize** signature(.Object = "Dirac"): initialize method

**location** signature(object = "Dirac"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "Dirac"): modifies the slot location of the parameter of the distribution

**log** signature(object = "Dirac"): returns an object of class "Dirac" distribution with log-transformed location parameter.

**Math** signature(object = "Dirac"): given a "Math" group generic fun an object of class "Dirac" distribution with fun-transformed location parameter is returned.

further arithmetic methods see [operators-methods](#)

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[DiracParameter-class](#) [DiscreteDistribution-class](#) [Naturals-class](#)

### Examples

```
D <- Dirac(location = 0) # D is a Dirac distribution with location=0.
r(D)(1)
# r(D)(1) generates a pseudo-random-number according to a Dirac
# distribution with location = 0,
# which of course will take 0 as value almost surely.
d(D)(0) # Density of this distribution is 1 for x = 0.
p(D)(1) # Probability that x < 1 is 1.
q(D)(.1) # q(D)(x) is always 0 (= location).
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
location(D) # location of this distribution is 0.
location(D) <- 2 # location of this distribution is now 2.
```

---

DiracParameter-class    *Class "DiracParameter"*

---

### Description

The parameter of a Dirac distribution, used by Dirac-class

## Objects from the Class

Objects can be created by calls of the form `new("DiracParameter", location)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Dirac` is instantiated.

## Slots

`location` Object of class "numeric": the location of a Dirac distribution

`name` Object of class "character": a name / comment for the parameters

## Extends

Class "Parameter", directly.

## Methods

**initialize** signature(.Object = "DiracParameter"): initialize method

**location** signature(object = "DiracParameter"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "DiracParameter"): modifies the slot location of the parameter of the distribution

## Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

## See Also

[Dirac-class](#) [Parameter-class](#)

## Examples

```
W <- new("DiracParameter", location=1)
location(W) # location of this distribution is 1.
location(W) <- 2 # location of this distribution is now 2.
```

---

DiscreteDistribution    *Generating function "DiscreteDistribution"*


---

## Description

Generates an object of class "DiscreteDistribution"

## Usage

```
DiscreteDistribution(supp, prob, .withArith=FALSE, .withSim=FALSE,
                    .lowerExact = TRUE, .logExact = FALSE,
                    .DistrCollapse = getdistrOption("DistrCollapse"),
                    .DistrCollapse.Unique.Warn =
                      getdistrOption("DistrCollapse.Unique.Warn"),
                    .DistrResolution = getdistrOption("DistrResolution"),
                    Symmetry = NoSymmetry())
```

## Arguments

supp	numeric vector which forms the support of the discrete distribution.
prob	vector of probability weights for the elements of supp.
.withArith	normally not set by the user, but if determining the entries supp, prob distributional arithmetics was involved, you may set this to TRUE.
.withSim	normally not set by the user, but if determining the entries supp, prob simulations were involved, you may set this to TRUE.
.lowerExact	normally not set by the user: whether the lower .tail=FALSE part is calculated exactly, avoing a "1-.".
.logExact	normally not set by the user: whether in determining slots d,p,q, we make particular use of a logarithmic representation to enhance accuracy.
.DistrCollapse	controls whether in generating a new discrete distribution, support points closer together than .DistrResolution are collapsed.
.DistrCollapse.Unique.Warn	controls whether there is a warning whenever collapsing occurs or when two points are collapsed by a call to unique() (default behaviour if .DistrCollapse is FALSE)
.DistrResolution	minimal spacing between two mass points in a discrete distribution
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; in this case use Symmetry=SphericalSymmetry(center).

**Details**

If prob is missing, all elements in supp are equally weighted.

Typical usages are

```
DiscreteDistribution(supp, prob)
DiscreteDistribution(supp)
```

**Value**

Object of class "DiscreteDistribution"

**Note**

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$ .

Also, we require that support points have distance at least `.DistrResolution`, if this condition fails, upon a suggestion by Jacob van Etten, <jacobvanetten@yahoo.com>, we use the global option `.DistrCollapse` to decide whether we use collapsing or not. If we do so, we collapse support points if they are too close to each other, taking the (left most) median among them as new support point which accumulates all the mass of the collapsed points. With `.DistrCollapse==FALSE`, we at least collapse points according to the result of `unique()`, and if after this collapsing, the minimal distance is less than `.DistrResolution`, we throw an error. By `.DistrCollapse.Unique.Warn`, we control, whether we throw a warning upon collapsing or not.

**Author(s)**

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

**See Also**

[DiscreteDistribution-class](#) [AbscontDistribution-class](#) [RtoDPQ.d](#)

**Examples**

```
# Dirac-measure at 0
D1 <- DiscreteDistribution(supp = 0)
D1
# simple discrete distribution
D2 <- DiscreteDistribution(supp = c(1:5), prob = c(0.1, 0.2, 0.3, 0.2, 0.2))
D2

plot(D2)
```

---

DiscreteDistribution-class  
*Class "DiscreteDistribution"*

---

## Description

The DiscreteDistribution-class is the mother-class of the class LatticeDistribution.

## Objects from the Class

Objects can be created by calls to `new("DiscreteDistribution", ...)`, but more easily is the use of the generating function "DiscreteDistribution". This generating function, from version 1.9 on, has been moved to this package from package **distrEx**.

## Slots

`img` Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

`param` Object of class "Parameter": the parameter of this distribution, having only the slot name "Parameter of a discrete distribution"

`r` Object of class "function": generates random numbers

`d` Object of class "function": density/probability function

`p` Object of class "function": cumulative distribution function

`q` Object of class "function": 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

`.finSupport` logical: used internally to check whether the true support is finite; in case `img` is one-dimensional, it is of length 2 (left and right end).

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

## Extends

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

**Methods**

**initialize** signature(.Object = "DiscreteDistribution"): initialize method

**coerce** signature(from = "DiscreteDistribution", to = "LatticeDistribution"): coerce method to class "LatticeDistribution" (checks if support is a lattice)

**Math** signature(x = "DiscreteDistribution"): application of a mathematical function, e.g. sin or tan to this discrete distribution

- abs: signature(x = "DiscreteDistribution"): exact image distribution of abs(x).
- exp: signature(x = "DiscreteDistribution"): exact image distribution of exp(x).
- sign: signature(x = "DiscreteDistribution"): exact image distribution of sign(x).
- sqrt: signature(x = "DiscreteDistribution"): exact image distribution of sqrt(x).
- log: signature(x = "DiscreteDistribution"): (with optional further argument base, defaulting to exp(1)) exact image distribution of log(x).
- log10: signature(x = "DiscreteDistribution"): exact image distribution of log10(x).
- gamma: signature(x = "DiscreteDistribution"): exact image distribution of gamma(x).
- lgamma: signature(x = "DiscreteDistribution"): exact image distribution of lgamma(x).
- digamma: signature(x = "DiscreteDistribution"): exact image distribution of digamma(x).

- signature(e1 = "DiscreteDistribution"): application of '-' to this discrete distribution

\* signature(e1 = "DiscreteDistribution", e2 = "numeric"): multiplication of this discrete distribution by an object of class 'numeric'

/ signature(e1 = "DiscreteDistribution", e2 = "numeric"): division of this discrete distribution by an object of class 'numeric'

+ signature(e1 = "DiscreteDistribution", e2 = "numeric"): addition of this discrete distribution to an object of class 'numeric'

- signature(e1 = "DiscreteDistribution", e2 = "numeric"): subtraction of an object of class 'numeric' from this discrete distribution

\* signature(e1 = "numeric", e2 = "DiscreteDistribution"): multiplication of this discrete distribution by an object of class 'numeric'

+ signature(e1 = "numeric", e2 = "DiscreteDistribution"): addition of this discrete distribution to an object of class 'numeric'

- signature(e1 = "numeric", e2 = "DiscreteDistribution"): subtraction of this discrete distribution from an object of class 'numeric'

+ signature(e1 = "DiscreteDistribution", e2 = "DiscreteDistribution"): Convolution of two discrete distributions. The slots p, d and q are approximated on a common grid.

- signature(e1 = "DiscreteDistribution", e2 = "DiscreteDistribution"): Convolution of two discrete distributions. The slots p, d and q are approximated on a common grid.

**support** signature(object = "DiscreteDistribution"): returns the support

**p.l** signature(object = "DiscreteDistribution"): returns the left continuous cumulative distribution function, i.e.;  $p.l(t) = P(\text{object} < t)$

**q.r** signature(object = "DiscreteDistribution"): returns the right-continuous quantile function, i.e.;  $q.r(s) = \sup\{t \mid P(\text{object} \geq t) \leq s\}$

**plot** signature(object = "DiscreteDistribution"): plots density, cumulative distribution and quantile function

### Internal subclass "AffLinDiscreteDistribution"

To enhance accuracy of several functionals on distributions, mainly from package **distrEx**, from version 1.9 of this package on, there is an internally used (but exported) subclass "AffLinDiscreteDistribution" which has extra slots `a`, `b` (both of class "numeric"), and `X0` (of class "DiscreteDistribution"), to capture the fact that the object has the same distribution as  $a * X0 + b$ . This is the class of the return value of methods

```
- signature(e1 = "DiscreteDistribution")
* signature(e1 = "DiscreteDistribution", e2 = "numeric")
/ signature(e1 = "DiscreteDistribution", e2 = "numeric")
+ signature(e1 = "DiscreteDistribution", e2 = "numeric")
- signature(e1 = "DiscreteDistribution", e2 = "numeric")
* signature(e1 = "numeric", e2 = "DiscreteDistribution")
+ signature(e1 = "numeric", e2 = "DiscreteDistribution")
- signature(e1 = "numeric", e2 = "DiscreteDistribution")
- signature(e1 = "AffLinDiscreteDistribution")
* signature(e1 = "AffLinDiscreteDistribution", e2 = "numeric")
/ signature(e1 = "AffLinDiscreteDistribution", e2 = "numeric")
+ signature(e1 = "AffLinDiscreteDistribution", e2 = "numeric")
- signature(e1 = "AffLinDiscreteDistribution", e2 = "numeric")
* signature(e1 = "numeric", e2 = "AffLinDiscreteDistribution")
+ signature(e1 = "numeric", e2 = "AffLinDiscreteDistribution")
- signature(e1 = "numeric", e2 = "AffLinDiscreteDistribution")
```

There also is a class union of "AffLinAbscontDistribution", "AffLinDiscreteDistribution", "AffLinUnivarLebDecDistribution" and called "AffLinDistribution" which is used for functionals.

### Internal virtual superclass "AcDcLcDistribution"

As many operations should be valid no matter whether the operands are of class "AbscontDistribution", "DiscreteDistribution", or "UnivarLebDecDistribution", there is a class union of these classes called "AcDcLcDistribution"; in particular methods for `"*"`, `"/"`, `"^"` (see [operators-methods](#)) and methods [Minimum](#), [Maximum](#), [Truncate](#), and [Huberize](#), and [convpow](#) are defined for this class union.

### Note

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$ .

Also, we require that support points have distance at least  $\text{getdistrOption}(\text{"DistrResolution"})$ , if this condition fails, upon a suggestion by Jacob van Etten, <jacobvanetten@yahoo.com>, we use the global option `getdistrOption("DistrCollapse")` to decide whether we use collapsing or not. If we do so, we collapse support points if they are too close to each other, taking



the (left most) median among them as new support point which accumulates all the mass of the collapsed points. With `getdistrOption("DistrCollapse")==FALSE`, we at least collapse points according to the result of `unique()`, and if after this collapsing, the minimal distance is less than `getdistrOption("DistrResoltion")`, we throw an error. By `getdistrOption("DistrCollapse.Unique.Warn")`, we control, whether we throw a warning upon collapsing or not.

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Parameter-class](#) [UnivariateDistribution-class](#) [LatticeDistribution-class](#) [AbscontDistribution-class](#)  
[Reals-class](#) [RtoDPQ.d](#)

### Examples

```
# Dirac-measure at 0
D1 <- DiscreteDistribution(supp = 0)
support(D1)

# simple discrete distribution
D2 <- DiscreteDistribution(supp = c(1:5), prob = c(0.1, 0.2, 0.3, 0.2, 0.2))
plot(D2)
(pp <- p(D2)(support(D2)))
p(D2)(support(D2)-1e-5)
p(D2)(support(D2)+1e-5)
p.l(D2)(support(D2))
p.l(D2)(support(D2)-1e-5)
p.l(D2)(support(D2)+1e-5)
q(D2)(pp)
q(D2)(pp-1e-5)
q(D2)(pp+1e-5)
## in RStudio or Jupyter IRKernel, use q.l(.)(.) instead of q(.)(.)
q.r(D2)(pp)
q.r(D2)(pp-1e-5)
q.r(D2)(pp+1e-5)
```

---

distr-defunct

Class "GeomParameter"

---

### Description

The parameter of a geometric distribution, used by Geom-class

### Objects from the Class

Objects were created by calls of the form `new("GeomParameter", prob)`. Usually an object of this class was not needed on its own, it was generated automatically when an object of the class `Geom` is instantiated.

### Slots

`prob` Object of class "numeric": the probability of a geometric distribution  
`name` Object of class "character": a name / comment for the parameters

### Extends

Class "Parameter", directly.

### Methods

**initialize** signature(.Object = "GeomParameter"): initialize method  
**prob** signature(object = "GeomParameter"): returns the slot prob of the parameter of the distribution  
**prob<-** signature(object = "GeomParameter"): modifies the slot prob of the parameter of the distribution

### Defunct

The use of class `GeomParameter` is defunct as of version 2.8.0; it is to be replaced by a corresponding use of class `NbinomParameter` with slot `size = 1` which may be generated, e.g. by `new("NbinomParameter", prob, size = 1, name = "Parameter of a Geometric distribution")`

### Author(s)

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

### See Also

[Defunct](#)

---

distrARITH

*Arithmetics on Distributions*


---

### Description

Provides information on the interpretation of arithmetics operating on Distributions in package **distr**

### Usage

```
distrARITH(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**

```
distrARITH()
```

---

Distribution-class	<i>Class "Distribution"</i>
--------------------	-----------------------------

---

**Description**

The Distribution-class is the mother-class of class UnivariateDistribution.

**Objects from the Class**

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

**Slots**

`img` Object of class "rSpace": the space of the image  
`param` Object of class "OptionalParameter": the parameter  
`r` Object of class "function": generates random numbers  
`d` Object of class "OptionalFunction": density function  
`p` Object of class "OptionalFunction": cumulative distribution function  
`q` Object of class "OptionalFunction": 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.

**Methods**

**img** signature(object = "Distribution"): returns the space of the image  
**param** signature(object = "Distribution"): returns the parameter  
**r** signature(object = "Distribution"): returns the random number generator  
**d** signature(object = "Distribution"): returns the density function  
**p** signature(object = "Distribution"): returns the cumulative distribution function  
**q** signature(object = "Distribution"): returns the quantile function  
**.logExact** signature(object = "Distribution"): returns slot .logExact if existing; else tries to convert the object to a newer version of its class by [conv2NewVersion](#) and returns the corresponding slot of the converted object.  
**.lowerExact** signature(object = "Distribution"): returns slot .lowerExact if existing; else tries to convert the object to a newer version of its class by [conv2NewVersion](#) and returns the corresponding slot of the converted object.  
**Symmetry:** returns slot Symmetry if existing; else tries to convert the object to a newer version of its class by [conv2NewVersion](#) and returns the corresponding slot of the converted object.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[UnivariateDistribution-class](#) [Parameter-class](#)

---

DistributionSymmetry-class

*Class of Symmetries for Distributions*

---

**Description**

Class of symmetries for distributions.

**Objects from the Class**

A virtual Class: No objects may be created from it.

**Slots**

type Object of class "character": describes type of symmetry.  
 SymmCenter Object of class "OptionalNumeric": center of symmetry.

**Extends**

Class "Symmetry", directly.

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Symmetry-class](#), [Distribution-class](#), [OptionalNumeric-class](#)

---

DistrList

*Generating function for DistrList-class*

---

**Description**

Generates an object of class "DistrList".

**Usage**

```
DistrList(..., Dlist)
```

**Arguments**

...	Objects of class "Distribution" (or subclasses)
Dlist	an optional list or object of class "DistrList"; if not missing it is appended to argument ...; this way DistrList may also be called with a list (or "DistrList"-object) as argument as suggested in an e-mail by Krunoslav Sever (thank you!)

**Value**

Object of class "DistrList"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[DistrList-class](#), [UnivarDistrList-class](#), [UnivarDistrList](#)

**Examples**

```
(DL <- DistrList(Norm(), Exp(), Pois()))
plot(DL)
as(Norm(), "DistrList")

## The function is currently defined as
function(...){
  new("DistrList", list(...))
}
```

---

DistrList-class	<i>List of distributions</i>
-----------------	------------------------------

---

**Description**

Create a list of distributions

**Objects from the Class**

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

**Slots**

`.Data` Object of class "list". A list of distributions.

**Extends**

Class "list", from data part.  
Class "vector", by class "list".

**Methods**

**show** signature(object = "DistrList")  
**plot** signature(object = "DistrList")  
**coerce** signature(from = "Distribution", to = "DistrList"): create a "DistrList" object from a "Distribution" object

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[DistrList](#), [Distribution-class](#)

### Examples

```
(DL <- new("DistrList", list(Norm(), Exp())))  
plot(DL)  
as(Norm(), "DistrList")
```

---

distrMASK

*Masking of/by other functions in package "distr"*

---

### Description

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

### Usage

```
distrMASK(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

```
distrMASK()
```

---

distroptions

*functions to change the global variables of the package ‘distr’*


---

## Description

With `distroptions` and `getdistrOption` you may inspect and change the global variables used by package **distr**.

## Usage

```
distroptions(...)
getdistrOption(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.

## Details

Invoking `distroptions()` with no arguments returns a list with the current values of the options. To access the value of a single option, one should use `getdistrOption("WarningSim")`, e.g., rather than `distroptions("WarningSim")` which is a *list* of length one.

## Value

`distroptions()` returns a list of the global options of **distr**.

`distroptions("RtoDPQ.e")` returns the global option `RtoDPQ.e` as a list of length 1.

`distroptions("RtoDPQ.e" = 3)` sets the value of the global option `RtoDPQ.e` to 3. `getdistrOption("RtoDPQ.e")` the current value set for option `RtoDPQ.e`.

## Currently available options

`DefaultNrGridPoints` default number of grid points in integration, default value:  $2^{12}$

`DistrResolution` minimal spacing between two mass points in a discrete distribution, default value:  $1e-6$

`DistrCollapse` logical; in discrete distributions, shall support points with distance smaller than `DistrResolution` be collapsed; default value: TRUE

`TruncQuantile` argument for q-slot at which to truncate; also, for discrete distributions, support is restricted to  $[q(\text{TruncQuantile}), q(1-\text{TruncQuantile})]$ , default value:  $1e-5$

`DefaultNrFFTGridPointsExponent` by default, for  $e = \text{DefaultNrFFTGridPointsExponent}$ , FFT uses  $2^e$  gridpoints; default value: 12

`RtoDPQ.e` by default, for reconstructing the d-,p-,q-slots out of simulations by slot `r`, `RtoDPQ` resp. `RtoDPQ.d` use  $10^e$  simulations, where  $e = \text{RtoDPQ.e}$ , default value: 5



**WarningSim** if `WarningSim==TRUE`, print/show issue a warning as to the precision of d-,p-,q-slots when these are obtained by `RtoDPQ` resp. `RtoDPQ.d`, default value: `TRUE`

**WarningArith** if `WarningArith==TRUE`, print/show issue a warning as to the interpretation of arithmetics operating on distributions, when the corresponding distribution to be plotted/shown is obtained by such an operation; keep in mind that arithmetics in fact operate on random variables distributed according to the given distributions and **not** on corresponding cdf's or densities; default value: `TRUE`

**withSweave** is code run in Sweave (then no new graphic devices are opened), default value: `FALSE`

**withgaps** controls whether in the return value of arithmetic operations the slot gaps of an the `AbscontDistribution` part is filled automatically based on empirical evaluations via `setgaps` —default `TRUE`

**simplifyD** controls whether in the return value of arithmetic operations there is a call to `simplifyD` or not —default `TRUE`

**use.generalized.inverse.by.default** logical; decides whether by default (i.e., if argument `generalized` of `solve` is not explicitly set), `solve` is to use generalized inverses if the original `solve`-method from package **base** fails; if the option is `FALSE`, in case of failure, and unless argument `generalized` is not explicitly set to `TRUE`, `solve` will throw an error as is the **base**-method behavior. The default value is `TRUE`.

**DistrCollapse.Unique.Warn** controls whether there is a warning whenever collapsing occurs or when two points are collapsed by a call to `unique()` (default behaviour if `DistrCollapse` is `FALSE`); —default `FALSE`

**warn.makeDNew** controls whether a warning is issued once in internal utility `.makeDNew` standard integration with `integrate` throws an error—default `TRUE`

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[options](#), [getOption](#)

### Examples

```
distributions("RtoDPQ.e") # returns the value of RtoDPQ.e, by default = 5
currentDistrOptions <- distributions()
distributions(RtoDPQ.e = 6)
distributions("RtoDPQ.e")
getdistrOption("RtoDPQ.e")
distributions(c("WarningSim", "WarningArith"))
getdistrOption("WarningSim")
distributions("WarningSim" = FALSE)
  # switches off warnings as to (In)accuracy due to simulations
distributions("WarningArith" = FALSE)
  # switches off warnings as to arithmetics
distributions(currentDistrOptions)
```

---

DistrSymmList	<i>Generating function for DistrSymmList-class</i>
---------------	--

---

**Description**

Generates an object of class "DistrSymmList".

**Usage**

```
DistrSymmList(...)
```

**Arguments**

...                    Objects of class "DistributionSymmetry" which shall form the list of symmetry types.

**Value**

Object of class "DistrSymmList"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[DistrSymmList-class](#)

**Examples**

```
DistrSymmList(NoSymmetry(), SphericalSymmetry(SymmCenter = 1),  
              EllipticalSymmetry(SymmCenter = 2))  
  
## The function is currently defined as  
function (...){  
  new("DistrSymmList", list(...))  
}
```

---

DistrSymmList-class      *List of Symmetries for a List of Distributions*


---

**Description**

Create a list of symmetries for a list of distributions

**Objects from the Class**

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

**Slots**

.Data Object of class "list". A list of objects of class "DistributionSymmetry".

**Extends**

Class "list", from data part.  
Class "vector", by class "list".

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[DistributionSymmetry-class](#)

**Examples**

```
new("DistrSymmList", list(NoSymmetry(), SphericalSymmetry(SymmCenter = 1),
                        EllipticalSymmetry(SymmCenter = 2)))
```

---

EllipticalSymmetry      *Generating function for EllipticalSymmetry-class*


---

**Description**

Generates an object of class "EllipticalSymmetry".

**Usage**

```
EllipticalSymmetry(SymmCenter = 0)
```

**Arguments**

SymmCenter      numeric: center of symmetry

**Value**

Object of class "EllipticalSymmetry"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[EllipticalSymmetry-class](#), [DistributionSymmetry-class](#)

**Examples**

```
EllipticalSymmetry()

## The function is currently defined as
function(SymmCenter = 0){
  new("EllipticalSymmetry", SymmCenter = SymmCenter)
}
```

---

EllipticalSymmetry-class

*Class for Elliptically Symmetric Distributions*

---

**Description**

Class for elliptically symmetric distributions.

**Objects from the Class**

Objects can be created by calls of the form `new("EllipticalSymmetry")`. More frequently they are created via the generating function `EllipticalSymmetry`. Elliptical symmetry for instance leads to a simplification for the computation of optimally robust influence curves.

**Slots**

type    Object of class "character": contains "elliptical symmetric distribution"

SymmCenter    Object of class "numeric": center of symmetry

**Extends**

Class "DistributionSymmetry", directly.

Class "Symmetry", by class "DistributionSymmetry".

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[EllipticalSymmetry](#), [DistributionSymmetry-class](#)

**Examples**

```
new("EllipticalSymmetry")
```

---

EmpiricalDistribution *Generating function "EmpiricalDistribution"*

---

**Description**

Generates an object of class "DiscreteDistribution"

**Usage**

```
EmpiricalDistribution(data, .withArith=FALSE, .withSim=FALSE,
                      .lowerExact = TRUE, .logExact = FALSE,
                      .DistrCollapse = getdistrOption("DistrCollapse"),
                      .DistrCollapse.Unique.Warn =
                        getdistrOption("DistrCollapse.Unique.Warn"),
                      .DistrResolution = getdistrOption("DistrResolution"),
                      Symmetry = NoSymmetry())
```

**Arguments**

<code>data</code>	numeric vector with data.
<code>.withArith</code>	normally not set by the user, but if determining the entries supp, probab distributonal arithmetics was involved, you may set this to TRUE.
<code>.withSim</code>	normally not set by the user, but if determining the entries supp, probab simulations were involved, you may set this to TRUE.
<code>.lowerExact</code>	normally not set by the user: whether the lower .tail=FALSE part is calculated exactly, avoing a "1-.".
<code>.logExact</code>	normally not set by the user: whether in determining slots d,p,q, we make particular use of a logarithmic representation to enhance accuracy.
<code>.DistrCollapse</code>	controls whether in generating a new discrete distribution, support points closer together than .DistrResolution are collapsed.
<code>.DistrCollapse.Unique.Warn</code>	controls whether there is a warning whenever collapsing occurs or when two points are collapsed by a call to unique() (default behaviour if .DistrCollapse is FALSE)

<code>.DistrResolution</code>	minimal spacing between two mass points in a discrete distribution
<code>Symmetry</code>	you may help <b>R</b> in calculations if you tell it whether the distribution is non-symmetric (default) or symmetric with respect to a center; in this case use <code>Symmetry=SphericalSymmetry(center)</code> .

### Details

The function is a simple utility function providing a wrapper to the generating function [DiscreteDistribution](#). Typical usage is

```
EmpiricalDistribution(data)
```

### Value

Object of class "DiscreteDistribution"

### Author(s)

Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[DiscreteDistribution](#) [DiscreteDistribution-class](#)

### Examples

```
x <- rnorm(20)
D1 <- EmpiricalDistribution(data = x)
D1

plot(D1)
```

---

EuclideanSpace-class    *Class "EuclideanSpace"*

---

### Description

The distribution-classes contain a slot where the sample space is stored. One typical sample space is the Euclidean Space in dimension  $k$ .

### Usage

```
EuclideanSpace(dimension = 1)
```

### Arguments

`dimension`      positive integer: dimension of the Euclidean space (default =1)

## Objects from the Class

Objects could theoretically be created by calls of the form `new("EuclideanSpace", dimension, name)`. Usually an object of this class is not needed on its own. `EuclideanSpace` is the mother-class of the class `Reals`, which is generated automatically when a univariate absolutely continuous distribution is instantiated.

## Slots

`dimension` Object of class "numeric": the dimension of the space, by default = 1

`name` Object of class "character": the name of the space, by default = "Euclidean Space"

## Extends

Class "rSpace", directly.

## Methods

**initialize** signature(.Object = "EuclideanSpace"): initialize method

**liesIn** signature(object = "EuclideanSpace", x = "numeric"): Does a particular vector lie in this space or not?

**dimension** signature(object = "EuclideanSpace"): returns the dimension of the space

**dimension<-** signature(object = "EuclideanSpace"): modifies the dimension of the space

## Author(s)

Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

## See Also

[rSpace-class](#) [Reals-class](#) [Distribution-class](#) [liesIn-methods](#)

## Examples

```
E <- EuclideanSpace(dimension = 2)
dimension(E) # The dimension of this space is 2.
dimension(E) <- 3 # The dimension of this space is now 3.
liesIn(E,c(0,0,0)) # TRUE
liesIn(E,c(0,0)) # FALSE
```

Exp-class

Class "Exp"

**Description**

The exponential distribution with rate  $\lambda$  has density

$$f(x) = \lambda e^{-\lambda x}$$

for  $x \geq 0$ .

C.f. [rexp](#)

**Objects from the Class**

Objects can be created by calls of the form `Exp(rate)`. This object is an exponential distribution.

**Slots**

`img` Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "ExpParameter": the parameter of this distribution (rate), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `rexp`)

`d` Object of class "function": density function (calls function `dexp`)

`p` Object of class "function": cumulative function (calls function `pexp`)

`q` Object of class "function": inverse of the cumulative function (calls function `qexp`)

`.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 "ExpOrGammaOrChisq", directly.

Class "AbscontDistribution", by class "ExpOrGammaOrChisq".

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



**Is-Relations**

By means of `setIs`, R “knows” that a distribution object `obj` of class “Exp” also is a Gamma distribution with parameters `shape = 1, scale = 1/rate(obj)` and a Weibull distribution with parameters `shape = 1, scale = 1/rate(obj)`

**Methods**

**initialize** signature(.Object = "Exp"): initialize method  
**rate** signature(object = "Exp"): returns the slot rate of the parameter of the distribution  
**rate<-** signature(object = "Exp"): modifies the slot rate of the parameter of the distribution  
 \* signature(e1 = "Exp", e2 = "numeric"): For the exponential distribution we use its closedness under positive scaling transformations.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[ExpParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rexp](#)

**Examples**

```
E <- Exp(rate = 1) # E is a exp distribution with rate = 1.
r(E)(1) # one random number generated from this distribution, e.g. 0.4190765
d(E)(1) # Density of this distribution is 0.3678794 for x = 1.
p(E)(1) # Probability that x < 1 is 0.6321206.
q(E)(.1) # Probability that x < 0.1053605 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
rate(E) # rate of this distribution is 1.
rate(E) <- 2 # rate of this distribution is now 2.
is(E, "Gammad") # yes
as(E, "Gammad")
is(E, "Weibull")
E+E+E ### a Gammad -distribution
2*E+Gammad(scale=1)
```

---

ExpParameter-class	Class "ExpParameter"
--------------------	----------------------

---

**Description**

The parameter of an exponential distribution, used by `Exp`-class and `DExp`-class

### Objects from the Class

Objects can be created by calls of the form `new("ExpParameter",rate)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Exp` is instantiated.

### Slots

`rate` Object of class "numeric": the rate of an exponential distribution

`name` Object of class "character": a name / comment for the parameters

### Extends

Class "Parameter", directly.

### Methods

**initialize** signature(.Object = "ExpParameter"): initialize method

**rate** signature(object = "ExpParameter"): returns the slot rate of the parameter of the distribution

**rate<-** signature(object = "ExpParameter"): modifies the slot rate of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Exp-class](#) [DExp-class](#) [Parameter-class](#)

### Examples

```
W <- new("ExpParameter", rate = 1)
rate(W) # rate of this distribution is 1.
rate(W) <- 2 # rate of this distribution is now 2.
```

Fd-class

Class "Fd"

**Description**

The F distribution with  $df1 = n_1$ , by default = 1, and  $df2 = n_2$ , by default = 1, degrees of freedom has density

$$d(x) = \frac{\Gamma(n_1/2 + n_2/2)}{\Gamma(n_1/2)\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{n_1/2} x^{n_1/2-1} \left(1 + \frac{n_1 x}{n_2}\right)^{-(n_1+n_2)/2}$$

for  $x > 0$ .

C.f. [rf](#)

**Objects from the Class**

Objects can be created by calls of the form `Fd(df1, df2)`. This object is a F distribution.

**Slots**

`img` Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "FParameter": the parameter of this distribution (`df1` and `df2`), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `rf`)

`d` Object of class "function": density function (calls function `df`)

`p` Object of class "function": cumulative function (calls function `pf`)

`q` Object of class "function": inverse of the cumulative function (calls function `qf`)

`.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

## Methods

**initialize** signature(.Object = "Fd"): initialize method  
**df1** signature(object = "Fd"): returns the slot df1 of the parameter of the distribution  
**df1<-** signature(object = "Fd"): modifies the slot df1 of the parameter of the distribution  
**df2** signature(object = "Fd"): returns the slot df2 of the parameter of the distribution  
**df2<-** signature(object = "Fd"): modifies the slot df2 of the parameter of the distribution

## Ad hoc methods

- An ad hoc method is provided for slot d if ncp!=0.
- For R Version <2.3.0 ad hoc methods are provided for slots q, r if ncp!=0; for R Version >=2.3.0 the methods from package **stats** are used.

## Note

It is the distribution of the ratio of the mean squares of n1 and n2 independent standard normals, and hence of the ratio of two independent chi-squared variates each divided by its degrees of freedom. Since the ratio of a normal and the root mean-square of m independent normals has a Student's  $t_m$  distribution, the square of a  $t_m$  variate has a F distribution on 1 and m degrees of freedom.

The non-central F distribution is again the ratio of mean squares of independent normals of unit variance, but those in the numerator are allowed to have non-zero means and ncp is the sum of squares of the means.

## Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

## See Also

[FParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rf](#)

## Examples

```
F <- Fd(df1 = 1, df2 = 1) # F is a F distribution with df=1 and df2=1.
r(F)(1) # one random number generated from this distribution, e.g. 29.37863
d(F)(1) # Density of this distribution is 0.1591549 for x=1 .
p(F)(1) # Probability that x<1 is 0.5.
q(F)(.1) # Probability that x<0.02508563 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
df1(F) # df1 of this distribution is 1.
df1(F) <- 2 # df1 of this distribution is now 2.
Fn <- Fd(df1 = 1, df2 = 1, ncp = 0.5)
# Fn is a F distribution with df=1, df2=1 and ncp =0.5.
d(Fn)(1) ## from R 2.3.0 on ncp no longer ignored...
```

flat.LCD

*Flattening a list of Lebesgue decomposed distributions***Description**

flattens a list of Lebesgue decomposed distributions endowed with weights to give one Lebesgue decomposed distribution

**Usage**

```
flat.LCD(..., mixCoeff = NULL, withgaps = getdistrOption("withgaps"))
```

**Arguments**

...	list of Lebesgue decomposed distributions
mixCoeff	Object of class "numeric" of the same length as ...: a vector of probabilities for the mixing components.
withgaps	logical; shall gaps be detected empirically?

**Details**

flat.LCD flattens a list of Lebesgue decomposed distributions given through ..., i.e., it takes all list elements and mixing coefficients and builds up the mixed distribution (forgetting about the components); the result will be one distribution of class `UnivarLebDecDistribution`. If `mixCoeff` is missing, all list elements are equally weighted. It is used internally in our methods for `"*"`, `"/"`, `"^"` (see [operators-methods](#)), [Minimum](#), and [convpow](#), as well in method [flat.mix](#).

**Value**

flat.LCD returns an object of class `UnivarLebDecDistribution`.

**Author(s)**

Peter Ruckdeschel <[peter.ruckdeschel@uni-oldenburg.de](mailto:peter.ruckdeschel@uni-oldenburg.de)>

**See Also**

[UnivarLebDecDistribution-class](#), [operators-methods](#)

**Examples**

```
D1 <- as(Norm(),"UnivarLebDecDistribution")
D2 <- as(Pois(1),"UnivarLebDecDistribution")
D3 <- as(Binom(1,.4),"UnivarLebDecDistribution")
flat.LCD(D1,D2,D3, mixCoeff = c(0.4,0.5,0.1))
```

---

flat.mix	<i>Default procedure to fill slots d,p,q given r for Lebesgue decomposed distributions</i>
----------	--

---

**Description**

function to do get empirical density, cumulative distribution and quantile function from random numbers

**Usage**

```
flat.mix(object)
```

**Arguments**

object                      object of class UnivariateMixingDistribution

**Details**

flat.mix generates  $10^e$  random numbers, by default

$$e = RtoDPQ.e$$

. Replicates are assumed to be part of the discrete part, unique values to be part of the a.c. part of the distribution. For the replicated ones, we generate a discrete distribution by a call to [DiscreteDistribution](#). The a.c. density is formed on the basis of  $n$  points using approxfun and density (applied to the unique values), by default

$$n = DefaultNrGridPoints$$

. The cumulative distribution function is based on all random variables, and, as well as the quantile function, is also created on the basis of  $n$  points using approxfun and ecdf. Of course, the results are usually not exact as they rely on random numbers.

**Value**

flat.mix returns an object of class UnivarLebDecDistribution.

**Note**

Use RtoDPQ for absolutely continuous and RtoDPQ.d for discrete distributions.

**Author(s)**

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

**See Also**

[UnivariateDistribution-class](#), [density](#), [approxfun](#), [ecdf](#)

**Examples**

```

D1 <- Norm()
D2 <- Pois(1)
D3 <- Binom(1,.4)
D4 <- UnivarMixingDistribution(D1,D2,D3, mixCoeff = c(0.4,0.5,0.1),
  withSimplify = FALSE)
D <- UnivarMixingDistribution(D1,D4,D1,D2, mixCoeff = c(0.4,0.3,0.1,0.2),
  withSimplify = FALSE)
D
D0<-flat.mix(D)
D0
plot(D0)

```

---

FParameter-class

Class "FParameter"

---

**Description**

The parameter of a F distribution, used by Fd-class

**Objects from the Class**

Objects can be created by calls of the form `new("FParameter", df1, df2, ncp)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Fd` is instantiated.

**Slots**

`df1` Object of class "numeric": the degrees of freedom of the nominator of an F distribution  
`df2` Object of class "numeric": the degrees of freedom of the denominator of an F distribution  
`ncp` Object of class "numeric": the noncentrality parameter of an F distribution  
`name` Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "FParameter"): initialize method  
**df1** signature(object = "FParameter"): returns the slot `df1` of the parameter of the distribution  
**df1<-** signature(object = "FParameter"): modifies the slot `df1` of the parameter of the distribution  
**df2** signature(object = "FParameter"): returns the slot `df2` of the parameter of the distribution  
**df2<-** signature(object = "FParameter"): modifies the slot `df2` of the parameter of the distribution

**ncp** signature(object = "FParameter"): returns the slot ncp of the parameter of the distribution  
**ncp<-** signature(object = "FParameter"): modifies the slot ncp of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Fd-class Parameter-class](#)

### Examples

```
W <- new("FParameter", df1 = 1, df2 = 1, ncp = 0)
df2(W) # df2 of this distribution is 1.
df2(W) <- 2 # df2 of this distribution is now 2.
```

---

Gammad-class

*Class "Gammad"*

---

### Description

The Gammad distribution with parameters shape =  $\alpha$ , by default = 1, and scale =  $\sigma$ , by default = 1, has density

$$d(x) = \frac{1}{\sigma^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\sigma}$$

for  $x > 0$ ,  $\alpha > 0$  and  $\sigma > 0$ . The mean and variance are  $E(X) = \alpha\sigma$  and  $Var(X) = \alpha\sigma^2$ . C.f. [rgamma](#)

### Objects from the Class

Objects can be created by calls of the form `Gammad(scale, shape)`. This object is a gamma distribution.

### Slots

**img** Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".  
**param** Object of class "GammaParameter": the parameter of this distribution (scale and shape), declared at its instantiation  
**r** Object of class "function": generates random numbers (calls function `rgamma`)  
**d** Object of class "function": density function (calls function `dgamma`)



p Object of class "function": cumulative function (calls function pgamma)

q Object of class "function": inverse of the cumulative function (calls function qgamma)

.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 "ExpOrGammaOrChisq", directly.

Class "AbscontDistribution", by class "ExpOrGammaOrChisq".

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "UnivariateDistribution".

### Methods

**initialize** signature(.Object = "Gammad"): initialize method

**scale** signature(object = "Gammad"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Gammad"): modifies the slot scale of the parameter of the distribution

**shape** signature(object = "Gammad"): returns the slot shape of the parameter of the distribution

**shape<-** signature(object = "Gammad"): modifies the slot shape of the parameter of the distribution

**+** signature(e1 = "Gammad", e2 = "Gammad"): For the Gamma distribution we use its closedness under convolutions.

**\*** signature(e1 = "Gammad", e2 = "numeric"): For the Gamma distribution we use its closedness under positive scaling transformations.

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@statmats.de>

### See Also

[GammaParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rgamma](#)

**Examples**

```
G <- Gammad(scale=1,shape=1) # G is a gamma distribution with scale=1 and shape=1.
r(G)(1) # one random number generated from this distribution, e.g. 0.1304441
d(G)(1) # Density of this distribution is 0.3678794 for x=1.
p(G)(1) # Probability that x<1 is 0.6321206.
q(G)(.1) # Probability that x<0.1053605 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
scale(G) # scale of this distribution is 1.
scale(G) <- 2 # scale of this distribution is now 2.
```

---

GammaParameter-class    *Class "GammaParameter"*

---

**Description**

The parameter of a gamma distribution, used by Gammad-class

**Objects from the Class**

Objects can be created by calls of the form `new("GammaParameter", shape, scale)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Gammad` is instantiated.

**Slots**

`shape` Object of class "numeric": the shape of a Gamma distribution  
`scale` Object of class "numeric": the scale of a Gamma distribution  
`name` Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "GammaParameter"): initialize method  
**scale** signature(object = "GammaParameter"): returns the slot scale of a parameter of a Gamma distribution  
**scale<-** signature(object = "GammaParameter"): modifies the slot scale of a parameter of a Gamma distribution  
**shape** signature(object = "GammaParameter"): returns the slot shape of a parameter of a Gamma distribution  
**shape<-** signature(object = "GammaParameter"): modifies the slot shape of a parameter of a Gamma distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Gammad-class Parameter-class](#)

**Examples**

```
W <- new("GammaParameter",scale=1,shape=1)
shape(W) # shape of this distribution is 1.
shape(W) <- 2 # shape of this distribution is now 2.
```

---

gaps-methods

---

*Methods for Functions gaps and setgaps in Package ‘distr’*


---

**Description**

[set]gaps-methods

**Usage**

```
gaps(object)
gaps(object)
gaps(object) <- value
setgaps(object, ...)
## S4 method for signature 'AbscontDistribution'
gaps(object)

## S4 method for signature 'AbscontDistribution'
setgaps(object, exactq = 6,
        ngrid = 50000, ...)
```

**Arguments**

object	object of class "AbscontDistribution" (or subclasses)
...	further arguments to be passed to setgaps; not yet used.
value	$n \times 2$ matrix $m$ of numerics where $c(t(m))$ is an ordered vector; value to be assigned to slot gaps
exactq	density values smaller than $10^{-\text{exactq}}$ are considered as 0.
ngrid	number of gridpoints at which the density is evaluated.

**Methods**

- gaps** signature(object = "AbscontDistribution"): returns slot gaps of an absolutely continuous distribution
- setgaps** signature(object = "AbscontDistribution"): tries to find out the gaps (where  $d(\text{object})$  is approximately 0) and fills slot gaps of object correspondingly
- setgaps** signature(object = "UnivarMixingDistribution"): for each mixing component, if it has a slot gaps, tries to find out the gaps and fills slot gaps of the component correspondingly, and, subsequently merges all found gap-slots of the components to a gap-slot for the object, using internal function [.mergegaps2](#).
- gaps<-** signature(object = "AbscontDistribution"): modifies slot gaps of an absolutely continuous distribution

Geom-class

Class "Geom"

**Description**

The geometric distribution with  $\text{prob} = p$  has density

$$p(x) = p(1 - p)^x$$

for  $x = 0, 1, 2, \dots$

C.f. [rgeom](#)

**Objects from the Class**

Objects can be created by calls of the form `Geom(prob)`. This object is a geometric distribution.

**Slots**

- img** Object of class "Naturals": The space of the image of this distribution has got dimension 1 and the name "Natural Space".
- param** Object of class "NbinomParameter": the parameter of this distribution (`prob`), declared at its instantiation (`size=1`)
- r** Object of class "function": generates random numbers (calls function `rgeom`)
- d** Object of class "function": density function (calls function `dgeom`)
- p** Object of class "function": cumulative function (calls function `pgeom`)
- q** Object of class "function": inverse of the cumulative function (calls function `qgeom`). The quantile is defined as the smallest value  $x$  such that  $F(x) \geq p$ , where  $F$  is the distribution function.
- support** Object of class "numeric": a (sorted) vector containing the support of the discrete density 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 "DiscreteDistribution", directly.

Class "Nbinom", directly.

Class "UnivariateDistribution", by class "DiscreteDistribution".

Class "Distribution", by class "DiscreteDistribution".

### Contains-Relations

By means of a contains argument in the class declaration, R “knows” that a distribution object obj of class "Geom" also is a negative Binomial distribution with parameters size = 1, prob = prob(obj)

### Methods

**initialize** signature(.Object = "Geom"): initialize method

**prob** signature(object = "Geom"): returns the slot prob of the parameter of the distribution

**prob<-** signature(object = "Geom"): modifies the slot prob of the parameter of the distribution

### Note

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Nbinom-class](#) [GeomParameter-class](#) [DiscreteDistribution-class](#) [Naturals-class](#) [rgeom](#)

### Examples

```
G <- Geom(prob = 0.5) # G is a geometric distribution with prob = 0.5.
r(G)(1) # one random number generated from this distribution, e.g. 0
d(G)(1) # Density of this distribution is 0.25 for x = 1.
p(G)(1) # Probability that x<1 is 0.75.
q(G)(.1) # x = 0 is the smallest value x such that p(G)(x) >= 0.1.
```

```
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
prob(G) # prob of this distribution is 0.5.
prob(G) <- 0.6 # prob of this distribution is now 0.6.
as(G, "Nbinom")
G+G+G
```

---

getLabel

*Labels for distribution objects*


---

## Description

a help function to get reasonable labels for distribution objects

## Usage

```
getLabel(x, withnames = TRUE)
```

## Arguments

x	a distribution object
withnames	logical: are the parameters (if any) of x to be displayed with names?

## Remark

The need for this helper function (external to our plot methods) was brought to our attention in a mail by Kouros Owzar <owzar001@mc.duke.edu>.

## Author(s)

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

## See Also

[plot-methods](#)

## Examples

```
## example due to Kouros Owzar:
foo<- function(law,n, withnames = TRUE)
{
  data.frame(muhat=mean(r(law)(n)),n=n,law= getLabel(law,withnames))
}
### a function that groups certain informations on
## created with distribution objects
do.call("rbind",lapply(list(Exp(1),Norm(0,1),Weibull(1,1)),foo,n=100))
do.call("rbind",lapply(list(Exp(1),Norm(0,1),Weibull(1,1)),foo,n=100,FALSE))
```

getLow, getUp

*getLow, getUp functions of package distr***Description**

getLow, getUp return lower and upper endpoint of a distribution — truncated to lower/upper TruncQuantile if infinite; in case of an object of class "LatticeDistribution" with infinite lattice length, we search for the smallest/largest point in the lattice which is returned by successive halving of  $x=0.5$  in `q(object)(x, lower.tail)` for `lower.tail` TRUE resp. FALSE.

**Usage**

```
## S4 method for signature 'AbscontDistribution'
getUp(object,
        eps = getdistrOption("TruncQuantile"))
## S4 method for signature 'DiscreteDistribution'
getUp(object, ...)
## S4 method for signature 'LatticeDistribution'
getUp(object, ...)
## S4 method for signature 'UnivarLebDecDistribution'
getUp(object,
        eps = getdistrOption("TruncQuantile"))
## S4 method for signature 'UnivarMixingDistribution'
getUp(object,
        eps = getdistrOption("TruncQuantile"))
## S4 method for signature 'AbscontDistribution'
getLow(object,
        eps = getdistrOption("TruncQuantile"))
## S4 method for signature 'DiscreteDistribution'
getLow(object, ...)
## S4 method for signature 'LatticeDistribution'
getLow(object, ...)
## S4 method for signature 'UnivarLebDecDistribution'
getLow(object,
        eps = getdistrOption("TruncQuantile"))
## S4 method for signature 'UnivarMixingDistribution'
getLow(object,
        eps = getdistrOption("TruncQuantile"))
```

**Arguments**

<code>object</code>	a distribution object
<code>eps</code>	truncation point (numeric)
<code>...</code>	for convenience only; makes it possible to call <code>getLow</code> , <code>getUp</code> with argument <code>eps</code> no matter of the class of <code>object</code> ; is ignored in these functions.

**Value**

getLow, getUp     a numeric of length 1

**Author(s)**

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

---

Huberize-methods

*Methods for function Huberize in Package 'distr'*

---

**Description**

Huberize-methods

**Usage**

```
Huberize(object, ...)
## S4 method for signature 'AcDcLcDistribution'
Huberize(object, lower, upper,
          withSimplify = getdistrOption("simplifyD"))
```

**Arguments**

object	distribution object
...	further arguments for Huberize; takes up lower, upper, withSimplify.
lower	numeric; lower truncation point
upper	numeric; upper truncation point
withSimplify	logical; is result to be piped through a call to <a href="#">simplifyD</a> ?

**Value**

the corresponding distribution of the truncated random variable

**Methods**

**Huberize** signature(object = "AcDcLcDistribution"): returns the unconditioned distribution of  $\min(\text{upper}, \max(X, \text{lower}))$ , if  $X$  is distributed according to object; the result is of class "UnivarLebDecDistribution" in general.

**See Also**

[Truncate](#)

**Examples**

```
Hub <- Huberize(Norm(), lower=-1, upper=2)
Hub
plot(Hub)
```



Hyper-class

Class "Hyper"

**Description**

The hypergeometric distribution is used for sampling *without* replacement. The density of this distribution with parameters  $m$ ,  $n$  and  $k$  (named  $Np$ ,  $N - Np$ , and  $n$ , respectively in the reference below) is given by

$$p(x) = \binom{m}{x} \binom{n}{k-x} / \binom{m+n}{k}$$

for  $x = 0, \dots, k$ . C.f. [rhyper](#)

**Objects from the Class**

Objects can be created by calls of the form `Hyper(m,n,k)`. This object is a hypergeometric distribution.

**Slots**

`img` Object of class "Naturals": The space of the image of this distribution has got dimension 1 and the name "Natural Space".

`param` Object of class "HyperParameter": the parameter of this distribution ( $m$ ,  $n$ ,  $k$ ), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `rhyper`)

`d` Object of class "function": density function (calls function `dhyper`)

`p` Object of class "function": cumulative function (calls function `phyper`)

`q` Object of class "function": inverse of the cumulative function (calls function `qhyper`). The  $\alpha$ -quantile is defined as the smallest value  $x$  such that  $p(x) \geq \alpha$ , where  $p$  is the cumulative function.

`support`: Object of class "numeric": a (sorted) vector containing the support of the discrete density 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 "DiscreteDistribution", directly.

Class "UnivariateDistribution", by class "DiscreteDistribution".

Class "Distribution", by class "DiscreteDistribution".

**Methods**

**initialize** signature(.Object = "Hyper"): initialize method

**m** signature(object = "Hyper"): returns the slot m of the parameter of the distribution

**m<-** signature(object = "Hyper"): modifies the slot m of the parameter of the distribution

**n** signature(object = "Hyper"): returns the slot n of the parameter of the distribution

**n<-** signature(object = "Hyper"): modifies the slot n of the parameter of the distribution

**k** signature(object = "Hyper"): returns the slot k of the parameter of the distribution

**k<-** signature(object = "Hyper"): modifies the slot k of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[HyperParameter-class](#) [DiscreteDistribution-class](#) [Naturals-class](#) [rhyper](#)

**Examples**

```
H <- Hyper(m=3,n=3,k=3) # H is a hypergeometric distribution with m=3,n=3,k=3.
r(H)(1) # one random number generated from this distribution, e.g. 2
d(H)(1) # Density of this distribution is 0.45 for x=1.
p(H)(1) # Probability that x<1 is 0.5.
q(H)(.1) # x=1 is the smallest value x such that p(H)(x)>=0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
m(H) # m of this distribution is 3.
m(H) <- 2 # m of this distribution is now 2.
```

---

HyperParameter-class    *Class "HyperParameter"*

---

**Description**

The parameter of a hypergeometric distribution, used by Hyper-class

**Objects from the Class**

Objects can be created by calls of the form `new("HyperParameter", k, m, n)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Hyper` is instantiated.

**Slots**

**k** Object of class "numeric": k of a hypergeometric distribution  
**m** Object of class "numeric": m of a hypergeometric distribution  
**n** Object of class "numeric": n of a hypergeometric distribution  
**name** Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "HyperParameter"): initialize method  
**k** signature(object = "HyperParameter"): returns the slot k of the parameter of the distribution  
**k<-** signature(object = "HyperParameter"): modifies the slot k of the parameter of the distribution  
**m** signature(object = "HyperParameter"): returns the slot m of the parameter of the distribution  
**m<-** signature(object = "HyperParameter"): modifies the slot m of the parameter of the distribution  
**n** signature(object = "HyperParameter"): returns the slot n of the parameter of the distribution  
**n<-** signature(object = "HyperParameter"): modifies the slot n of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Hyper-class Parameter-class](#)

**Examples**

```
W <- new("HyperParameter",k=3, m=3, n=3)
m(W) # m of this distribution is 3.
m(W) <- 2 # m of this distribution is now 2.
```

---

igamma	<i>Inverse of the digamma function</i>
--------	--

---

**Description**

Function `igamma` is a numerical inverse of `digamma`.

**Usage**

```
igamma(v)
```

**Arguments**

`v` a numeric in the range `[-100000,18]`

**Details**

`igamma` is vectorized; it is won by spline inversion of a grid; it works well for range `[digamma(1e-5);digamma(1e8)]` or `[-100000,18]`.

**Value**

`igamma(x)` is a value `u` such that `digamma(u)` is approximately `x`.

**Author(s)**

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

**See Also**

`digamma`

**Examples**

```
igamma(digamma(c(1e-4,1,20,1e8)))
```

---

img-methods	<i>Methods for Function img in Package 'distr'</i>
-------------	--

---

**Description**

`img-methods`

**Methods**

**img** `signature(object = "Distribution")`: returns the image space / domain of the distribution

k-methods

*Methods for Function k in Package 'distr'***Description**

k-methods

**Methods****k** signature(object = "HyperParameter"): returns the slot k of the parameter of the distribution**k<-** signature(object = "HyperParameter"): modifies the slot k of the parameter of the distribution**k** signature(object = "Hyper"): returns the slot k of the parameter of the distribution**k<-** signature(object = "Hyper"): modifies the slot k of the parameter of the distribution

lambda-methods

*Methods for Function lambda in Package 'distr'***Description**

lambda-methods

**Methods****lambda** signature(object = "PoisParameter"): returns the slot lambda of the parameter of the distribution**lambda<-** signature(object = "PoisParameter"): modifies the slot lambda of the parameter of the distribution**lambda** signature(object = "Pois"): returns the slot lambda of the parameter of the distribution**lambda<-** signature(object = "Pois"): modifies the slot lambda of the parameter of the distribution

---

Lattice-class	Class "Lattice"
---------------	-----------------

---

### Description

Class Lattice formalizes an affine linearly generated grid of (support) points  $\text{pivot} + (0:(\text{Length}-1)) * \text{width}$ ; this is used for subclass LatticeDistribution of class DiscreteDistribution which in addition to the latter contains a slot lattice of class Lattice.

### Usage

```
Lattice(pivot = 0, width = 1, Length = 2, name = "a lattice")
```

### Arguments

pivot	the (finite) utmost left or right value of the lattice
width	the (finite) grid-width; if negative the lattice is expanded to the left, else to the right
Length	the (possibly infinite) length of the lattice
name	the (possibly empty) name of the lattice (inherited from class rSpace)

### Objects from the Class

Objects may be generated by calling the generating function Lattice.

### Slots

pivot	Object of class "numeric": — the pivot of the lattice; must be of length 1
width	Object of class "numeric": — the width of the lattice; must be of length 1 and must not be 0
Length	Object of class "numeric": — the width of the lattice; must be an integer > 0 of length 1
name	Object of class "character": the name of the space, by default = "a lattice"

### Extends

Class "rSpace", directly.

### Methods

```
pivot signature(.Object = "Lattice"): returns the 'pivot' slot
pivot<- signature(.Object = "Lattice"): modifies the 'pivot' slot
width signature(.Object = "Lattice"): returns the 'width' slot
width<- signature(.Object = "Lattice"): modifies the 'width' slot
Length signature(.Object = "Lattice"): returns the 'Length' slot
Length<- signature(.Object = "Lattice"): modifies the 'Length' slot
```

**Author(s)**

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

**See Also**

[rSpace-class LatticeDistribution-class](#)

**Examples**

```
L <- Lattice(pivot = 0, width = 1, Length = Inf, name = "the Naturals")
name(L)
pivot(L) <- 1 ### now starting from 1
```

---

LatticeDistribution	Class "LatticeDistribution"
---------------------	-----------------------------

---

**Description**

The LatticeDistribution-class is the mother-class of the classes Binom, Dirac, Geom, Hyper, Nbinom and Poisson. It formalizes a distribution on a regular affine linear lattice.

**Usage**

```
LatticeDistribution(lattice = NULL, supp = NULL, prob = NULL,
  .withArith = FALSE, .withSim = FALSE,
  DiscreteDistribution = NULL, check = TRUE,
  Symmetry = NoSymmetry())
```

**Arguments**

DiscreteDistribution	an object of class DiscreteDistribution or AffLinDiscreteDistribution to be coerced to LatticeDistribution or AffLinLatticeDistribution, respectively
lattice	lattice (of class Lattice) which determines the support of the discrete distribution.
supp	numeric vector which forms the support of the discrete distribution.
prob	vector of probability weights for the elements of supp.
.withArith	normally not set by the user, but if determining the entries supp, prob distributional arithmetics was involved, you may set this to TRUE.
.withSim	normally not set by the user, but if determining the entries supp, prob simulations were involved, you may set this to TRUE.
check	logical: if TRUE, LatticeDistribution() throws an error if argument lattice and other arguments are inconsistent or if there is no way to automatically generate a lattice argument. If check == FALSE, LatticeDistribution() returns an object of DiscreteDistribution, ignoring argument lattice

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; in this case use `Symmetry=SphericalSymmetry(center)`.

## Details

Typical usages are

```
LatticeDistribution(DiscreteDistribution)
LatticeDistribution(lattice, DiscreteDistribution)
LatticeDistribution(lattice, supp, prob, .withArith, .withSim, check = FALSE)
LatticeDistribution(lattice, supp, prob)
LatticeDistribution(supp)
```

For the generating function `LatticeDistribution()`, the arguments are processed in the following order:

Arguments `.withSim` and `.withArith` are used in any case.

If there is an argument `DiscreteDistribution` (of the respective class), all its slots (except for `.withSim` and `.withArith`) will be used for filling the slots of the object of class `LatticeDistribution()/AffLinLatticeDistribution()`. If in addition, there is an argument `lattice` of class `Lattice`, it will be checked for consistency with argument `DiscreteDistribution` and if oK will be used for slot `lattice` of the object of class `LatticeDistribution()/AffLinLatticeDistribution()`. In case there is no `lattice` argument, slot `lattice` will be constructed from slot `support` from argument `DiscreteDistribution`.

If there is no argument `DiscreteDistribution`, but there are arguments `supp` and `lattice` (the latter of class `Lattice`) then these are checked for consistency and if oK, generating function `DiscreteDistribution()` is called with arguments `supp`, `prob`, `.withArith`, and `.withSim` to produce an object of class `DiscreteDistribution` the slots of which will be used for the filling the slots of the object of class `LatticeDistribution()/AffLinLatticeDistribution()`. If in this case, argument `prob` is not given explicitly, all elements in `supp` are equally weighted.

If there is no argument `DiscreteDistribution`, but there is an argument `lattice` of class `Lattice` (but no argument `slot`) then if `Length(lattice)` is finite, a corresponding support vector `supp` is generated from argument `lattice` and generating function `DiscreteDistribution()` is called with arguments `supp`, `prob`, `.withArith`, and `.withSim` to produce an object of class `DiscreteDistribution` the slots of which will be used for the filling the slots of the object of class `LatticeDistribution()`. If in the same situation `Length(lattice)` is not finite, a finite length for the support vector is extracted from argument `prob` and after generating `supp` one proceeds as in the finite `Length(lattice)` case.

If there is no argument `DiscreteDistribution` and no argument `lattice` of class `Lattice` but an argument `supp` then it will be checked if `supp` makes for a lattice, and if so, `DiscreteDistribution()` is called with arguments `supp`, `prob`, `.withArith`, and `.withSim` to produce an object of class `DiscreteDistribution` the slots of which will be used for the filling the slots of the object of class



LatticeDistribution(). The corresponding lattice-slot will be filled with information from argument supp.

The price for this flexibility of arguments, LatticeDistribution() may be called with, is that you should call LatticeDistribution() with *named arguments* only.

Note that internally we suppress lattice points from the support where the probability is 0.

### Objects from the Class

The usual way to generate objects of class LatticeDistribution is to call the generating function LatticeDistribution() (see details).

Somewhat more flexible, but also proner to inconsistencies is a call to new("LatticeDistribution"), where you may explicitly specify random number generator, (counting) density, cumulative distribution and quantile functions. For convenience, in this call to new("LatticeDistribution"), an additional possibility is to only specify the random number generator. The function RtoDPQ.d then approximates the three remaining slots d, p and q by random sampling.

### Note

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$ .

### Author(s)

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

### See Also

[Parameter-class Lattice-class LatticeDistribution-class Reals-class RtoDPQ.d](#)

### Examples

```
LatticeDistribution(DiscreteDistribution = DiscreteDistribution(supp =
  c(4,3,2), prob=c(0.3,0.1,0.6)))
LatticeDistribution(supp = c(4,3,2))
```

---

LatticeDistribution-class

*Class "LatticeDistribution"*

---

### Description

The LatticeDistribution-class is the mother-class of the classes Binom, Dirac, Geom, Hyper, Nbinom and Poisson. It formalizes a distribution on a regular affine linear lattice.

## Objects from the Class

The usual way to generate objects of class `LatticeDistribution` is to call the generating function `LatticeDistribution`.

Somewhat more flexible, but also prone to inconsistencies is a call to `new("LatticeDistribution")`, where you may explicitly specify random number generator, (counting) density, cumulative distribution and quantile functions. For convenience, in this call to `new("LatticeDistribution")`, an additional possibility is to only specify the random number generator. The function `RtoDPQ.d` then approximates the three remaining slots `d`, `p` and `q` by random sampling.

## Slots

`img` Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

`param` Object of class "Parameter": the parameter of this distribution, having only the slot name "Parameter of a discrete distribution"

`r` Object of class "function": generates random numbers

`d` Object of class "function": (counting) density/probability function

`p` Object of class "function": cumulative distribution function

`q` Object of class "function": quantile function

`support` Object of class "numeric": a (sorted) vector containing the support of the discrete density function

`lattice` Object of class "Lattice": the lattice generating the support.

`.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

`initialize` signature(`.Object` = "LatticeDistribution"): initialize method

`-` signature(`e1` = "LatticeDistribution"): application of '-' to this lattice distribution

`*` signature(`e1` = "LatticeDistribution", `e2` = "numeric"): multiplication of this lattice distribution by an object of class 'numeric'

`/` signature(`e1` = "LatticeDistribution", `e2` = "numeric"): division of this lattice distribution by an object of class 'numeric'

```

+ signature(e1 = "LatticeDistribution", e2 = "numeric"): addition of this lattice distribu-
  tion to an object of class 'numeric'
- signature(e1 = "LatticeDistribution", e2 = "numeric"): subtraction of an object of class
  'numeric' from this lattice distribution
* signature(e1 = "numeric", e2 = "LatticeDistribution"): multiplication of this lattice dis-
  tribution by an object of class 'numeric'
+ signature(e1 = "numeric", e2 = "LatticeDistribution"): addition of this lattice distribu-
  tion to an object of class 'numeric'
- signature(e1 = "numeric", e2 = "LatticeDistribution"): subtraction of this lattice distri-
  bution from an object of class 'numeric'
+ signature(e1 = "LatticeDistribution", e2 = "LatticeDistribution"): Convolution of two
  lattice distributions. Slots p, d and q are approximated by grids.
- signature(e1 = "LatticeDistribution", e2 = "LatticeDistribution"): Convolution of two
  lattice distributions. The slots p, d and q are approximated by grids.
sqrt signature(x = "LatticeDistribution"): exact image distribution of sqrt(x).
lattice accessor method to the corresponding slot.
coerce signature(from = "LatticeDistribution", to = "DiscreteDistribution"): coerces
  an object from "LatticeDistribution" to "DiscreteDistribution" thereby cancelling
  out support points with probability 0.

```

#### Internal subclass "AffLinLatticeDistribution"

To enhance accuracy of several functionals on distributions, mainly from package **distrEx**, there is an internally used (but exported) subclass "AffLinLatticeDistribution" which has extra slots `a`, `b` (both of class "numeric"), and `X0` (of class "LatticeDistribution"), to capture the fact that the object has the same distribution as  $a * X0 + b$ . This is the class of the return value of methods

```

- signature(e1 = "LatticeDistribution")
* signature(e1 = "LatticeDistribution", e2 = "numeric")
/ signature(e1 = "LatticeDistribution", e2 = "numeric")
+ signature(e1 = "LatticeDistribution", e2 = "numeric")
- signature(e1 = "LatticeDistribution", e2 = "numeric")
* signature(e1 = "numeric", e2 = "LatticeDistribution")
+ signature(e1 = "numeric", e2 = "LatticeDistribution")
- signature(e1 = "numeric", e2 = "LatticeDistribution")
- signature(e1 = "AffLinLatticeDistribution")
* signature(e1 = "AffLinLatticeDistribution", e2 = "numeric")
/ signature(e1 = "AffLinLatticeDistribution", e2 = "numeric")
+ signature(e1 = "AffLinLatticeDistribution", e2 = "numeric")
- signature(e1 = "AffLinLatticeDistribution", e2 = "numeric")
* signature(e1 = "numeric", e2 = "AffLinLatticeDistribution")
+ signature(e1 = "numeric", e2 = "AffLinLatticeDistribution")
- signature(e1 = "numeric", e2 = "AffLinLatticeDistribution")

```

There is also an explicit coerce-method from class "AffLinLatticeDistribution" to class "AffLinDiscreteDistribution" which cancels out support points with probability 0.

**Note**

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$

**Author(s)**

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

**See Also**

[LatticeDistribution](#) [Parameter-class Lattice](#) [class UnivariateDistribution](#) [class DiscreteDistribution](#) [class Binom](#) [class Dirac](#) [class Geom](#) [class Hyper](#) [class Nbinom](#) [class Pois](#) [class AbscontDistribution](#) [class Reals](#) [class RtoDPQ.d](#)

**Examples**

```
B <- Binom(prob = 0.1, size = 10) # B is a Binomial distribution w/ prob=0.1 and size=10.
P <- Pois(lambda = 1) # P is a Poisson distribution with lambda = 1.
D1 <- B+1 # a new Lattice distributions with exact slots d, p, q
D2 <- D1*3 # a new Lattice distributions with exact slots d, p, q
D3 <- B+P # a new Lattice distributions with approximated slots d, p, q
D4 <- D1+P # a new Lattice distributions with approximated slots d, p, q
support(D4) # the (approximated) support of this distribution is 1, 2, ..., 21
r(D4)(1) # one random number generated from this distribution, e.g. 4
d(D4)(1) # The (approximated) density for x=1 is 0.1282716.
p(D4)(1) # The (approximated) probability that x<=1 is 0.1282716.
q(D4)(.5) # The (approximated) 50 percent quantile is 3.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
```

---

Length-methods

---

*Methods for Function Length in Package 'distr'*


---

**Description**

Length-methods

**Methods**

**Length** signature(object = "Lattice"): returns the slot Length of the lattice

**Length<-** signature(object = "Lattice"): modifies the slot Length of the lattice

**Length** signature(object = "LatticeDistribution"): returns the slot Length of the lattice slot of the distribution

**Length<-** signature(object = "LatticeDistribution"): modifies the slot Length of the lattice slot of the distribution

liesIn-methods

*Methods for Function liesIn in Package 'distr'***Description**

liesIn-methods

**Methods****liesIn** signature(object = "EuclideanSpace", x = "numeric"):

Does a particular vector lie in this space or not?

**liesIn** signature(object = "Naturals", x = "numeric"):

Does a particular vector only contain naturals?

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**

```

liesInSupport(object, x, ...)
## S4 method for signature 'UnivarLebDecDistribution,numeric'
liesInSupport(object,x, checkFin = FALSE)
## S4 method for signature 'UnivarMixingDistribution,numeric'
liesInSupport(object,x, checkFin = FALSE)
## S4 method for signature 'LatticeDistribution,numeric'
liesInSupport(object,x, checkFin = FALSE)
## S4 method for signature 'DiscreteDistribution,numeric'
liesInSupport(object,x, checkFin = FALSE)
## S4 method for signature 'AbscontDistribution,numeric'
liesInSupport(object,x, checkFin = FALSE)
## S4 method for signature 'Distribution,matrix'
liesInSupport(object,x, checkFin = FALSE)
## S4 method for signature 'ExpOrGammaOrChisq,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Lnorm,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Fd,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Norm,numeric'
liesInSupport(object,x, checkFin = TRUE)

```

```
## S4 method for signature 'DExp,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Cauchy,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Td,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Logis,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Weibull,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Unif,numeric'
liesInSupport(object,x, checkFin = TRUE)
## S4 method for signature 'Beta,numeric'
liesInSupport(object,x, checkFin = TRUE)
```

### Arguments

object	object of class "Distribution"
x	numeric vector or matrix
checkFin	logical: in case FALSE, we simply check whether x lies in the <i>numerical</i> (i.e., possibly cut to relevant quantile range) support; in case TRUE we try to check this by more exact techniques (e.g. in case of lattice distributions) and by using slot .finSupport / the return values of q.l(object) in 0 and 1. This is only used on discrete (parts of) distributions).
...	used for specific arguments to particular methods.

### Value

logical vector

### Methods

**object = "DiscreteDistribution", x = "numeric":** We return a logical vector of the same length as x with TRUE when x lies in the support of object. As support we use the value of support(object), so this is possibly cut to relevant quantile ranges. In case checkFin is TRUE, in addition, we flag those coordinates to TRUE where  $x < \min(\text{support}(\text{object}))$  if `is.na(object@.finSupport[1])` or `object@.finSupport[1]==FALSE` or `q.l(object)(0)==-Inf`, and similarly, where  $x > \max(\text{support}(\text{object}))$  if `is.na(object@.finSupport[2])` or `object@.finSupport[2]==Inf` or `q.l(object)(1)==Inf`. In addition we flag those coordinates to TRUE where  $q.l(\text{object})(0) \leq x < \min(\text{support}(\text{object}))$  if `object@.finSupport[1]==TRUE` and, similarly, where  $q.l(\text{object})(1) \geq x > \max(\text{support}(\text{object}))$  if `object@.finSupport[2]==TRUE`.

**object = "Distribution", x = "matrix":** Argument x is cast to vector and then the respective liesInSupport method for vectors is called. The method throws an error when the dispatch mechanism does not find a suitable, applicable respective vector-method.

**object = "AbscontDistribution", x = "numeric":** We return a logical vector of the same length as x with TRUE where  $q.l(\text{object})(0) \leq x \leq q.l(\text{object})(1)$  (and replace the boundary values by `q.l(object)(10*.Machine$double.eps)` resp. `q.l(object)(1-10*.Machine$double.eps)` once the return values for 0 or 1 return are NaN.

**object = "LatticeDistribution", x = "numeric":** We return a logical vector of the same length as x with TRUE when x lies in the support of object. As support we use the value of support(object), so this is possibly cut to relevant quantile ranges. In case checkFin is TRUE, we instead use the lattice information: We check whether all values (x-pivot(lattice(object))/width(lattice(object))) are non-negative integers and are non larger than Length(lattice(object))-1. In addition, we flag those coordinates to TRUE where  $x < \min(\text{support}(\text{object}))$  if is.na(object@.finSupport[1]) or object@.finSupport[1]==FALSE, and similarly, where  $x > \max(\text{support}(\text{object}))$  if is.na(object@.finSupport[2]) or object@.finSupport[2]==FALSE.

**object = "UnivarLebDecDistribution", x = "numeric":** We split up object into discrete and absolutely continuous part and for each of them apply liesInSupport separately; the two return values are combined by a coponentwise logical |.

**object = "UnivarMixingDistribution", x = "numeric":** We first cast object to UnivarLebDecDistribution by flat.mix and then apply the respective method.

### Author(s)

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

### See Also

[Distribution-class](#)

### Examples

```
liesInSupport(Exp(1), rnorm(10))

# note
x <- rpois(10, lambda = 10)
liesInSupport(Pois(1), x)
# better
liesInSupport(Pois(1), x, checkFin = TRUE)
liesInSupport(Pois(1), 1000*x, checkFin = TRUE)
liesInSupport(-10*Pois(1), -10*x+1, checkFin = TRUE)

xs = c(1000*x, runif(10))
D <- UnivarMixingDistribution(Pois(1), Unif())
liesInSupport(D, xs)
```

---

Lnorm-class

Class "Lnorm"

---

### Description

The log normal distribution has density

$$d(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2}$$

where  $\mu$ , by default = 0, and  $\sigma$ , by default = 1, are the mean and standard deviation of the logarithm. C.f. [rlnorm](#)

## Objects from the Class

Objects can be created by calls of the form `Lnorm(meanlog, sdlog)`. This object is a log normal distribution.

## Slots

**img** Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

**param** Object of class "LnormParameter": the parameter of this distribution (meanlog and sdlog), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function `rlnorm`)

**d** Object of class "function": density function (calls function `dlnorm`)

**p** Object of class "function": cumulative function (calls function `plnorm`)

**q** Object of class "function": inverse of the cumulative function (calls function `qlnorm`)

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

## Methods

**initialize** signature(.Object = "Lnorm"): initialize method

**meanlog** signature(object = "Lnorm"): returns the slot meanlog of the parameter of the distribution

**meanlog<-** signature(object = "Lnorm"): modifies the slot meanlog of the parameter of the distribution

**sdlog** signature(object = "Lnorm"): returns the slot sdlog of the parameter of the distribution

**sdlog<-** signature(object = "Lnorm"): modifies the slot sdlog of the parameter of the distribution

**\*** signature(e1 = "Lnorm", e2 = "numeric"): For the Lognormal distribution we use its closedness under positive scaling transformations.



**Note**

The mean is  $E(X) = \exp(\mu + 1/2\sigma^2)$ , and the variance  $Var(X) = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$  and hence the coefficient of variation is  $\sqrt{\exp(\sigma^2) - 1}$  which is approximately  $\sigma$  when that is small (e.g.,  $\sigma < 1/2$ ).

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[LnormParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rlnorm](#)

**Examples**

```
L <- Lnorm(meanlog=1,sdlog=1) # L is a lnorm distribution with mean=1 and sd=1.
r(L)(1) # one random number generated from this distribution, e.g. 3.608011
d(L)(1) # Density of this distribution is 0.2419707 for x=1.
p(L)(1) # Probability that x<1 is 0.1586553.
q(L)(.1) # Probability that x<0.754612 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
meanlog(L) # meanlog of this distribution is 1.
meanlog(L) <- 2 # meanlog of this distribution is now 2.
```

---

LnormParameter-class    *Class "LnormParameter"*

---

**Description**

The parameter of a log normal distribution, used by Lnorm-class

**Objects from the Class**

Objects can be created by calls of the form `new("LnormParameter", meanlog, sdlog)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class Lnorm is instantiated.

**Slots**

meanlog   Object of class "numeric": the mean of a log normal distribution  
 sdlog   Object of class "numeric": the sd of a log normal distribution  
 name   Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "LnormParameter"): initialize method

**sdlog** signature(object = "LnormParameter"): returns the slot sdlog of the parameter of the distribution

**sdlog<-** signature(object = "LnormParameter"): modifies the slot sdlog of the parameter of the distribution

**meanlog** signature(object = "LnormParameter"): returns the slot meanlog of the parameter of the distribution

**meanlog<-** signature(object = "LnormParameter"): modifies the slot meanlog of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Lnorm-class Parameter-class](#)

**Examples**

```
W <- new("LnormParameter",sdlog=1,meanlog=0)
meanlog(W) # meanlog of this distribution is 0.
meanlog(W) <- 2 # meanlog of this distribution is now 2.
```

---

location-methods

---

*Methods for Function location in Package 'distr'*


---

**Description**

location-methods

**Methods**

**location** signature(object = "LogisParameter"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "LogisParameter"): modifies the slot location of the parameter of the distribution

**location** signature(object = "Logis"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "Logis"): modifies the slot location of the parameter of the distribution

**location** signature(object = "CauchyParameter"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "CauchyParameter"): modifies the slot location of the parameter of the distribution

**location** signature(object = "Cauchy"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "Cauchy"): modifies the slot location of the parameter of the distribution

**location** signature(object = "DiracParameter"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "DiracParameter"): modifies the slot location of the parameter of the distribution

**location** signature(object = "Dirac"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "Dirac"): modifies the slot location of the parameter of the distribution

---

Logis-class

Class "Logis"

---

## Description

The Logistic distribution with location =  $\mu$ , by default = 0, and scale =  $\sigma$ , by default = 1, has distribution function

$$p(x) = \frac{1}{1 + e^{-(x-\mu)/\sigma}}$$

and density

$$d(x) = \frac{1}{\sigma} \frac{e^{(x-\mu)/\sigma}}{(1 + e^{(x-\mu)/\sigma})^2}$$

It is a long-tailed distribution with mean  $\mu$  and variance  $\pi^2/3\sigma^2$ . C.f. [rlogis](#)

## Objects from the Class

Objects can be created by calls of the form `Logis(location,scale)`. This object is a logistic distribution.

**Slots**

**img** Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

**param** Object of class "LogisParameter": the parameter of this distribution (location and scale), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function rlogis)

**d** Object of class "function": density function (calls function dlogis)

**p** Object of class "function": cumulative function (calls function plogis)

**q** Object of class "function": inverse of the cumulative function (calls function qlogis)

**.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

**Methods**

**initialize** signature(.Object = "Logis"): initialize method

**location** signature(object = "Logis"): returns the slot location of the parameter of the distribution

**location<-** signature(object = "Logis"): modifies the slot location of the parameter of the distribution

**scale** signature(object = "Logis"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Logis"): modifies the slot scale of the parameter of the distribution

**\*** signature(e1 = "Logis", e2 = "numeric")

**+** signature(e1 = "Logis", e2 = "numeric"): For the logistic location scale family we use its closedness under affine linear transformations.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[LogisParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rlogis](#)

**Examples**

```
L <- Logis(location = 1,scale = 1)
# L is a logistic distribution with location = 1 and scale = 1.
r(L)(1) # one random number generated from this distribution, e.g. 5.87557
d(L)(1) # Density of this distribution is 0.25 for x = 1.
p(L)(1) # Probability that x < 1 is 0.5.
q(L)(.1) # Probability that x < -1.197225 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
location(L) # location of this distribution is 1.
location(L) <- 2 # location of this distribution is now 2.
```

---

LogisParameter-class    *Class "LogisParameter"*

---

**Description**

The parameter of a logistic distribution, used by Logis-class

**Objects from the Class**

Objects can be created by calls of the form `new("LogisParameter",scale,location)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Logis` is instantiated.

**Slots**

`scale` Object of class "numeric": the scale of a logistic distribution  
`location` Object of class "numeric": the location of a logistic distribution  
`name` Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "LogisParameter"): initialize method  
**location** signature(object = "LogisParameter"): returns the slot location of the parameter of the distribution  
**location<-** signature(object = "LogisParameter"): modifies the slot location of the parameter of the distribution  
**scale** signature(object = "LogisParameter"): returns the slot scale of the parameter of the distribution  
**scale<-** signature(object = "LogisParameter"): modifies the slot scale of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Logis-class](#) [Parameter-class](#)

**Examples**

```
W <- new("LogisParameter",location=0,scale=1)
scale(W) # scale of this distribution is 1.
scale(W) <- 2 # scale of this distribution is now 2.
```

---

m-methods

*Methods for Function m in Package ‘distr’*

---

**Description**

m-methods

**Methods**

**m** signature(object = "HyperParameter"): returns the slot m of the parameter of the distribution

**m<-** signature(object = "HyperParameter"): modifies the slot m of the parameter of the distribution

**m** signature(object = "Hyper"): returns the slot m of the parameter of the distribution

**m<-** signature(object = "Hyper"): modifies the slot m of the parameter of the distribution

---

makeAbscontDistribution

*"makeAbscontDistribution"*

---

**Description**

Transforms an object of "UnivariateDistribution" to an object of class "makeAbscontDistribution".

**Usage**

```
makeAbscontDistribution(object, gaps = NULL,
  param = NULL, img = NULL,
  withgaps = getdistrOption("withgaps"),
  ngrid = getdistrOption("DefaultNrGridPoints"),
  ep = getdistrOption("TruncQuantile"))
```

**Arguments**

object	Objects of class "UnivariateDistribution" (or subclasses)
gaps	slot gaps (of class "matrix" with two columns) to be filled (i.e. <code>t(gaps)</code> must be ordered if read as vector)
param	parameter (of class "OptionalParameter")
img	image range of the distribution (of class "rSpace")
withgaps	logical; shall gaps be reconstructed empirically?
ngrid	number of gridpoints
ep	tolerance epsilon

**Details**

takes slot `p` of object and then generates an "AbscontDistribution" object using generating function [AbscontDistribution](#).

**Author(s)**

Peter Ruckdeschel <[peter.ruckdeschel@uni-oldenburg.de](mailto:peter.ruckdeschel@uni-oldenburg.de)>

**Examples**

```
Hu <- Huberize(Norm(), -2,1)
Hu
plot(Hu)
Hu0 <- makeAbscontDistribution(Hu)
Hu0
plot(Hu0)
```

---

Math-methods

*Methods for Functions from group 'Math' in Package 'distr'*


---

**Description**

Math-methods provide automatical generation of image distributions for random variables transformed by functions from group [Math](#)

**Methods**

**Math** signature(`x = "AbscontDistribution"`): application of a mathematical function from group [Math](#), e.g. `sin` or `exp` (including `log`, `log10`, `gamma`, `lgamma`, `digamma`), to this absolutely continuous distribution

**Math** signature(`x = "DiscreteDistribution"`): application of a mathematical function, e.g. `sin` or `exp` (including `log`, `log10`, `gamma`, `lgamma`, `digamma`), to this discrete distribution

**Math** signature(`x = "UnivarLebDecDistribution"`): application of a mathematical function from group [Math](#), e.g. `sin` or `exp` (including `log`, `log10`, `gamma`, `lgamma`), to this Lebesgue decomposed distribution

**Math** signature(x = "UnivarLebDecDistribution"): application of a mathematical function from group **Math**, e.g. sin or exp (including log, log10, gamma, lgamma), to this distribution of class "AcDcLcDistribution"

**abs** signature(x = "AbscontDistribution"): application of function abs to this absolutely continuous distribution; (exactly)

**abs** signature(x = "DiscreteDistribution"): application of function abs to this discrete distribution; (exactly)

**sign** signature(x = "AbscontDistribution"): application of function abs to this absolutely continuous distribution; (exactly)

**sign** signature(x = "DiscreteDistribution"): application of function abs to this discrete continuous distribution; (exactly)

**exp** signature(x = "AbscontDistribution"): application of function exp to this absolutely continuous distribution; (exactly)

**exp** signature(x = "DiscreteDistribution"): application of function exp to this discrete distribution; (exactly)

**log** signature(x = "AbscontDistribution"): application of function log to this absolutely continuous distribution; (exactly for R-version >2.5.1)

**log** signature(x = "DiscreteDistribution"): application of function log to this discrete distribution; (exactly for R-version >2.5.1)

---

Max-methods

---

*Methods for Function Max in Package 'distr'*


---

## Description

Max-methods

## Methods

**Max** signature(object = "UnifParameter"): returns the slot Max of the parameter of the distribution

**Max<-** signature(object = "UnifParameter"): modifies the slot Max of the parameter of the distribution

**Max** signature(object = "Unif"): returns the slot Max of the parameter of the distribution

**Max<-** signature(object = "Unif"): modifies the slot Max of the parameter of the distribution



---

mean-methods	<i>Methods for Function mean in Package ‘distr’</i>
--------------	---

---

**Description**

mean-methods

**Methods**

- mean** signature(object = "NormParameter"): returns the slot mean of the parameter of the distribution
- mean<-** signature(object = "NormParameter"): modifies the slot mean of the parameter of the distribution
- mean** signature(object = "Norm"): returns the slot mean of the parameter of the distribution
- mean<-** signature(object = "Norm"): modifies the slot mean of the parameter of the distribution

---

meanlog-methods	<i>Methods for Function meanlog in Package ‘distr’</i>
-----------------	--

---

**Description**

meanlog-methods

**Methods**

- meanlog** signature(object = "LnormParameter"): returns the slot meanlog of the parameter of the distribution
- meanlog<-** signature(object = "LnormParameter"): modifies the slot meanlog of the parameter of the distribution
- meanlog** signature(object = "Lnorm"): returns the slot meanlog of the parameter of the distribution
- meanlog<-** signature(object = "Lnorm"): modifies the slot meanlog of the parameter of the distribution

---

Min-methods

---

*Methods for Function Min in Package ‘distr’*


---

## Description

Min-methods

## Methods

**Min** signature(object = "UnifParameter"): returns the slot Min of the parameter of the distribution

**Min<-** signature(object = "UnifParameter"): modifies the slot Min of the parameter of the distribution

**Min** signature(object = "Unif"): returns the slot Min of the parameter of the distribution

**Min<-** signature(object = "Unif"): modifies the slot Min of the parameter of the distribution

---

Minimum-methods

---

*Methods for functions Minimum and Maximum in Package ‘distr’*


---

## Description

Minimum and Maximum-methods

## Usage

```
Minimum(e1, e2, ...)
Maximum(e1, e2, ...)
## S4 method for signature 'AbscontDistribution,AbscontDistribution'
Minimum(e1,e2, ...)
## S4 method for signature 'DiscreteDistribution,DiscreteDistribution'
Minimum(e1,e2, ...)
## S4 method for signature 'AbscontDistribution,Dirac'
Minimum(e1,e2,
        withSimplify = getdistrOption("simplifyD"))
## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
Minimum(e1,e2,
        withSimplify = getdistrOption("simplifyD"))
## S4 method for signature 'AcDcLcDistribution,AcDcLcDistribution'
Maximum(e1,e2,
        withSimplify = getdistrOption("simplifyD"))
## S4 method for signature 'AbscontDistribution,numeric'
Minimum(e1,e2, ...)
## S4 method for signature 'DiscreteDistribution,numeric'
Minimum(e1,e2, ...)
```

```
## S4 method for signature 'AcDcLcDistribution,numeric'
Minimum(e1,e2,
        withSimplify = getdistrOption("simplifyD"))
## S4 method for signature 'AcDcLcDistribution,numeric'
Maximum(e1,e2,
        withSimplify = getdistrOption("simplifyD"))
```

### Arguments

e1	distribution object
e2	distribution object or numeric
...	further arguments (to be able to call various methods with the same arguments)
withSimplify	logical; is result to be piped through a call to <a href="#">simplifyD</a> ?

### Value

the corresponding distribution of the minimum / maximum

### Methods

**Minimum** signature(e1 = "AbscontDistribution", e2 = "AbscontDistribution"): returns the distribution of  $\min(X1, X2)$ , if  $X1, X2$  are independent and distributed according to e1 and e2 respectively; the result is again of class "AbscontDistribution"

**Minimum** signature(e1 = "DiscreteDistribution", e2 = "DiscreteDistribution"): returns the distribution of  $\min(X1, X2)$ , if  $X1, X2$  are independent and distributed according to e1 and e2 respectively; the result is again of class "DiscreteDistribution"

**Minimum** signature(e1 = "AbscontDistribution", e2 = "Dirac"): returns the distribution of  $\min(X1, X2)$ , if  $X1, X2$  are distributed according to e1 and e2 respectively; the result is of class "UnivarLebDecDistribution"

**Minimum** signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"): returns the distribution of  $\min(X1, X2)$ , if  $X1, X2$  are distributed according to e1 and e2 respectively; the result is of class "UnivarLebDecDistribution"

**Minimum** signature(e1 = "AcDcLcDistribution", e2 = "numeric"): if  $e2 = n$ , returns the distribution of  $\min(X1, X2, \dots, Xn)$ , if  $X1, X2, \dots, Xn$  are i.i.d. according to e1; the result is of class "UnivarLebDecDistribution"

**Maximum** signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"): returns the distribution of  $\max(X1, X2)$ , if  $X1, X2$  are distributed according to e1 and e2 respectively; translates into  $-\text{Minimum}(-e1, -e2)$ ; the result is of class "UnivarLebDecDistribution"

**Maximum** signature(e1 = "AcDcLcDistribution", e2 = "numeric"): if  $e2 = n$ , returns the distribution of  $\max(X1, X2, \dots, Xn)$ , if  $X1, X2, \dots, Xn$  are i.i.d. according to e1; translates into  $-\text{Minimum}(-e1, e2)$ ; the result is of class "UnivarLebDecDistribution"

### See Also

[Huberize](#), [Truncate](#)

Examples

```
plot(Maximum(Unif(0,1), Minimum(Unif(0,1), Unif(0,1))))
plot(Minimum(Exp(4),4))
## a sometimes lengthy example...
plot(Minimum(Norm(),Pois()))
```

---

n-methods	<i>Methods for Function n in Package ‘distr’</i>
-----------	--

---

Description

n-methods

Methods

- n** signature(object = "HyperParameter"): returns the slot n of the parameter of the distribution
- n<-** signature(object = "HyperParameter"): modifies the slot n of the parameter of the distribution
- n** signature(object = "Hyper"): returns the slot n of the parameter of the distribution
- n<-** signature(object = "Hyper"): modifies the slot n of the parameter of the distribution

---

name-methods	<i>Methods for Function name in Package ‘distr’</i>
--------------	---

---

Description

name-methods

Methods

- name** signature(object = "Parameter"): returns the slot name of the parameter
- name<-** signature(object = "Parameter"): modifies the slot name of the parameter
- name** signature(object = "rSpace"): returns the slot name of the space
- name<-** signature(object = "rSpace"): modifies the slot name of the space

---

Naturals-class	Class "Naturals"
----------------	------------------

---

### Description

The distribution-classes contain a slot where the sample space is stored. Typically, discrete random variables take naturals as values.

### Usage

```
Naturals()
```

### Objects from the Class

Objects could theoretically be created by calls of the form `new("Naturals",dimension,name)`. Usually an object of this class is not needed on its own. It is generated automatically when a univariate discrete distribution is instantiated.

### Slots

`dimension` Object of class "character": the dimension of the space, by default = 1

`name` Object of class "character": the name of the space, by default = "Natural Space"

### Extends

Class "Reals", directly.

Class "EuclideanSpace", by class "Reals".

Class "rSpace", by class "Reals".

### Methods

**initialize** signature(.Object = "Naturals"): initialize method

**liesIn** signature(object = "Naturals",x = "numeric"): Does a particular vector only contain naturals?

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Reals-class](#) [DiscreteDistribution-class](#)

**Examples**

```

N <- Naturals()
liesIn(N,1) # TRUE
liesIn(N,c(0,1)) # FALSE
liesIn(N,0.1) # FALSE

```

---

Nbinom-class

---

Class "Nbinom"

---

**Description**

The negative binomial distribution with size =  $n$ , by default = 1, and prob =  $p$ , by default = 0.5, has density

$$d(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

for  $x = 0, 1, 2, \dots$

This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached. C.f. [rnbino](#)

**Objects from the Class**

Objects can be created by calls of the form `Nbinom(prob, size)`. This object is a negative binomial distribution.

**Slots**

**img** Object of class "Naturals": The space of the image of this distribution has got dimension 1 and the name "Natural Space".

**param** Object of class "NbinomParameter": the parameter of this distribution (prob, size), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function `rnbino`)

**d** Object of class "function": density function (calls function `dnbinom`)

**p** Object of class "function": cumulative function (calls function `pnbinom`)

**q** Object of class "function": inverse of the cumulative function (calls function `qnbinom`). The quantile is defined as the smallest value  $x$  such that  $F(x) \geq p$ , where  $F$  is the distribution function.

**support** Object of class "numeric": a (sorted) vector containing the support of the discrete density 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 "DiscreteDistribution", directly.  
 Class "UnivariateDistribution", by class "DiscreteDistribution".  
 Class "Distribution", by class "DiscreteDistribution".

**Methods**

**initialize** signature(.Object = "Nbinom"): initialize method

**prob** signature(object = "Nbinom"): returns the slot prob of the parameter of the distribution

**prob<-** signature(object = "Nbinom"): modifies the slot prob of the parameter of the distribution

**size** signature(object = "Nbinom"): returns the slot size of the parameter of the distribution

**size<-** signature(object = "Nbinom"): modifies the slot size of the parameter of the distribution

+ signature(e1 = "Nbinom", e2 = "Nbinom"): For the negative binomial distribution we use its closedness under convolutions.

**Note**

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$ .

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[NbinomParameter-class](#) [Geom-class](#) [DiscreteDistribution-class](#) [Naturals-class](#) [rnbino](#)

**Examples**

```
N <- Nbinom(prob = 0.5, size = 1) # N is a binomial distribution with prob=0.5 and size=1.
r(N)(1) # one random number generated from this distribution, e.g. 3
d(N)(1) # Density of this distribution is 0.25 for x=1.
p(N)(0.4) # Probability that x<0.4 is 0.5.
q(N)(.1) # x=0 is the smallest value x such that p(B)(x)>=0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
size(N) # size of this distribution is 1.
size(N) <- 2 # size of this distribution is now 2.
```

---

NbinomParameter-class    *Class "NbinomParameter"*


---

### Description

The parameter of a negative binomial distribution, used by Nbinom-class

### Objects from the Class

Objects can be created by calls of the form `new("NbinomParameter",prob,size)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class Nbinom is prepared.

### Slots

**prob** Object of class "numeric": the probability of a negative binomial distribution

**size** Object of class "numeric": the size of a negative binomial distribution

**name** Object of class "character": a name / comment for the parameters

### Extends

Class "Parameter", directly.

### Methods

**initialize** signature(.Object = "NbinomParameter"): initialize method

**prob** signature(object = "NbinomParameter"): returns the slot prob of the parameter of the distribution

**prob<-** signature(object = "NbinomParameter"): modifies the slot prob of the parameter of the distribution

**size** signature(object = "NbinomParameter"): returns the slot size of the parameter of the distribution

**size<-** signature(object = "NbinomParameter"): modifies the slot size of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Nbinom-class](#) [Parameter-class](#)



## Examples

```
W <- new("NbinomParameter",prob=0.5,size=1)
size(W) # size of this distribution is 1.
size(W) <- 2 # size of this distribution is now 2.
```

---

ncp-methods

---

*Methods for Function ncp in Package 'distr'*


---

## Description

ncp-methods

## Methods

**ncp** signature(object = "BetaParameter"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "BetaParameter"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "Beta"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "Beta"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "ChisqParameter"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "ChisqParameter"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "Chisq"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "Chisq"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "FParameter"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "FParameter"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "Fd"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "Fd"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "TParameter"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "TParameter"): modifies the slot ncp of the parameter of the distribution

**ncp** signature(object = "Td"): returns the slot ncp of the parameter of the distribution

**ncp<-** signature(object = "Td"): modifies the slot ncp of the parameter of the distribution

---

Norm-class

Class "Norm"

---

## Description

The normal distribution has density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$$

where  $\mu$  is the mean of the distribution and  $\sigma$  the standard deviation. C.f. [rnorm](#)

## Objects from the Class

Objects can be created by calls of the form `Norm(mean, sd)`. This object is a normal distribution.

## Slots

`img` Object of class "Reals": The domain of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "UniNormParameter": the parameter of this distribution (mean and sd), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `rnorm`)

`d` Object of class "function": density function (calls function `dnorm`)

`p` Object of class "function": cumulative function (calls function `pnorm`)

`q` Object of class "function": inverse of the cumulative function (calls function `qnorm`)

`.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

**Methods**

```
- signature(e1 = "Norm", e2 = "Norm")

+ signature(e1 = "Norm", e2 = "Norm"): For the normal distribution the exact convolution formulas are implemented thereby improving the general numerical approximation.

* signature(e1 = "Norm", e2 = "numeric")

+ signature(e1 = "Norm", e2 = "numeric"): For the normal distribution we use its closedness under affine linear transformations.

initialize signature(.Object = "Norm"): initialize method

mean signature(object = "Norm"): returns the slot mean of the parameter of the distribution

mean<- signature(object = "Norm"): modifies the slot mean of the parameter of the distribution

sd signature(object = "Norm"): returns the slot sd of the parameter of the distribution

sd<- signature(object = "Norm"): modifies the slot sd of the parameter of the distribution

further arithmetic methods see operators-methods
```

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[UniNormParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rnorm](#)

**Examples**

```
N <- Norm(mean=1,sd=1) # N is a normal distribution with mean=1 and sd=1.
r(N)(1) # one random number generated from this distribution, e.g. 2.257783
d(N)(1) # Density of this distribution is 0.3989423 for x=1.
p(N)(1) # Probability that x<1 is 0.5.
q(N)(.1) # Probability that x<-0.2815516 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
mean(N) # mean of this distribution is 1.
sd(N) <- 2 # sd of this distribution is now 2.
M <- Norm() # M is a normal distribution with mean=0 and sd=1.
O <- M+N # O is a normal distribution with mean=1 (=1+0) and sd=sqrt(5) (=sqrt(2^2+1^2)).
```

---

NormParameter-class      *Class "NormParameter"*

---

### Description

The parameter of a normal distribution, used by Norm-class

### Objects from the Class

Objects can be created by calls of the form `new("NormParameter", sd, mean)`. Usually an object of this class is not needed on its own. It is the mother-class of the class `UniNormParameter`, which is generated automatically when such a distribution is instantiated.

### Slots

`sd` Object of class "numeric": the sd of a normal distribution  
`mean` Object of class "numeric": the mean of a normal distribution  
`name` Object of class "character": a name / comment for the parameters

### Extends

Class "Parameter", directly.

### Methods

**initialize** signature(.Object = "NormParameter"): initialize method  
**mean** signature(object = "NormParameter"): returns the slot mean of the parameter of the distribution  
**mean<-** signature(object = "NormParameter"): modifies the slot mean of the parameter of the distribution  
**sd** signature(object = "NormParameter"): returns the slot sd of the parameter of the distribution  
**sd<-** signature(object = "NormParameter"): modifies the slot sd of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Norm-class Parameter-class](#)

**Examples**

```
W <- new("NormParameter", mean = 0, sd = 1)
sd(W) # sd of this distribution is 1.
sd(W) <- 2 # sd of this distribution is now 2.
```

---

NoSymmetry

*Generating function for NoSymmetry-class*

---

**Description**

Generates an object of class "NoSymmetry".

**Usage**

```
NoSymmetry()
```

**Value**

Object of class "NoSymmetry"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[NoSymmetry-class](#), [DistributionSymmetry-class](#)

**Examples**

```
NoSymmetry()

## The function is currently defined as
function(){ new("NoSymmetry") }
```

---

NoSymmetry-class	<i>Class for Non-symmetric Distributions</i>
------------------	--

---

**Description**

Class for non-symmetric distributions.

**Objects from the Class**

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

**Slots**

type   Object of class "character": contains "non-symmetric distribution"  
SymmCenter   Object of class "NULL"

**Extends**

Class "DistributionSymmetry", directly.  
Class "Symmetry", by class "DistributionSymmetry".

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[NoSymmetry](#), [Distribution-class](#)

**Examples**

```
new("NoSymmetry")
```

---

operators-methods	<i>Methods for operators +,-,*,/,... in Package distr</i>
-------------------	---

---

**Description**

Arithmetics and unary mathematical transformations for distributions

**Arguments**

e1,e2                   objects of class "UnivariateDistribution" (or subclasses) or "numeric"

## Details

Arithmetics as well as all functions from group Math, see [Math](#) are provided for distributions; wherever possible exact expressions are used; else random variables are generated according to this transformation and subsequently the remaining slots filled by [RtoDPQ](#), [RtoDPQ.d](#)

## Methods

- signature(e1 = "UnivariateDistribution", e2 = "missing") unary operator; result again of class "UnivariateDistribution"; exact
- signature(e1 = "Norm", e2 = "missing") unary operator; result again of "Norm"; exact
- + signature(e1 = "UnivariateDistribution", e2 = "numeric") result again of class "UnivariateDistribution"; exact
- + signature(e1 = "AbscontDistribution", e2 = "numeric") result of class "AffLinAbscontDistribution"; exact
- + signature(e1 = "DiscreteDistribution", e2 = "numeric") result of class "AffLinDiscreteDistribution"; exact
- + signature(e1 = "LatticeDistribution", e2 = "numeric") result of class "AffLinLatticeDistribution"; exact
- + signature(e1 = "UnivarLebDecDistribution", e2 = "numeric") result of class "AffLinUnivarLebDecDistribution"; exact
- + signature(e1 = "CompoundDistribution", e2 = "numeric") result of class "AffLinUnivarLebDecDistribution"; exact
- + signature(e1 = "AffLinAbscontDistribution", e2 = "numeric") result again of class "AffLinAbscontDistribution"; exact
- + signature(e1 = "AffLinDiscreteDistribution", e2 = "numeric") result again of class "AffLinDiscreteDistribution"; exact
- + signature(e1 = "AffLinLatticeDistribution", e2 = "numeric") result again of class "AffLinLatticeDistribution"; exact
- + signature(e1 = "AffLinUnivarLebDecDistribution", e2 = "numeric") result of class "AffLinUnivarLebDecDistribution"; exact
- + signature(e1 = "Cauchy", e2 = "numeric") result again of class "Cauchy"; exact
- + signature(e1 = "Dirac", e2 = "numeric") result again of class "Dirac"; exact
- + signature(e1 = "Norm", e2 = "numeric") result again of class "Norm"; exact
- + signature(e1 = "Unif", e2 = "numeric") result again of class "Unif"; exact
- + signature(e1 = "Logis", e2 = "numeric") result again of class "Logis"; exact
- + signature(e1 = "numeric", e2 = "UnivariateDistribution") is translated to signature(e1 = "UnivariateDistribution", e2 = "numeric"); exact
- signature(e1 = "UnivariateDistribution", e2 = "ANY"); exact
- signature(e1 = "UnivariateDistribution", e2 = "numeric") is translated to  $e1 + (-e2)$ ; exact
- signature(e1 = "numeric", e2 = "UnivariateDistribution") is translated to  $(-e1) + e2$ ; exact

```

- signature(e1 = "numeric", e2 = "Beta") if ncp(e2)==0 and e1 == 1, an exact (central) Beta(shape1
  = shape2(e2), shape2 = shape1(e2)) is returned, else the default method is used; exact
* signature(e1 = "UnivariateDistribution", e2 = "numeric") result again of class "UnivariateDistribution";
  exact
* signature(e1 = "AbscontDistribution", e2 = "numeric") result of class "AffLinAbscontDistribution";
  exact
* signature(e1 = "DiscreteDistribution", e2 = "numeric") result of class "AffLinDiscreteDistribution";
  exact
* signature(e1 = "LatticeDistribution", e2 = "numeric") result of class "AffLinLatticeDistribution";
  exact
* signature(e1 = "UnivarLebDecDistribution", e2 = "numeric") result of class "AffLinUnivarLebDecDistribution";
  exact
* signature(e1 = "CompoundDistribution", e2 = "numeric") result of class "AffLinUnivarLebDecDistribution";
  exact
* signature(e1 = "AffLinAbscontDistribution", e2 = "numeric") result again of class "AffLinAbscontDistribution";
  exact
* signature(e1 = "AffLinDiscreteDistribution", e2 = "numeric") result again of class "AffLinDiscreteDistribution";
  exact
* signature(e1 = "AffLinLatticeDistribution", e2 = "numeric") result again of class "AffLinLatticeDistribution";
  exact
* signature(e1 = "AffLinUnivarLebDecDistribution", e2 = "numeric") result of class "AffLinUnivarLebDecDistribution";
  exact
* signature(e1 = "DExp", e2 = "numeric") if abs(e2)>0 result again of class "DExp"; exact
* signature(e1 = "Exp", e2 = "numeric") if e2>0 result again of class "Exp"; exact
* signature(e1 = "ExpOrGammaOrChisq", e2 = "numeric") if e1 is a Gamma distribution and
  e2>0 result of class "Gammad"; exact
* signature(e1 = "Weibull", e2 = "numeric") if e2>0 result of class "Weibull"; exact
* signature(e1 = "Cauchy", e2 = "numeric") if abs(e2)>0 result again of class "Cauchy"; exact
* signature(e1 = "Dirac", e2 = "numeric") result again of class "Dirac"; exact
* signature(e1 = "Norm", e2 = "numeric") if abs(e2)>0 result again of class "Norm"; exact
* signature(e1 = "Unif", e2 = "numeric") if abs(e2)>0 result again of class "Unif"; exact
* signature(e1 = "Logis", e2 = "numeric") if e2>0 result again of class "Logis"; exact
* signature(e1 = "Lnorm", e2 = "numeric") if e2>0 result again of class "Lnorm"; exact
* signature(e1 = "numeric", e2 = "UnivariateDistribution") is translated to signature(e1
  = "UnivariateDistribution", e2 = "numeric"); exact
/ signature(e1 = "UnivariateDistribution", e2 = "numeric") is translated to e1 * (1/e2);
  exact
+ signature(e1 = "UnivariateDistribution", e2 = "UnivariateDistribution") result again
  of class "UnivariateDistribution"; is generated by simulations

```



- `signature(e1 = "UnivariateDistribution", e2 = "UnivariateDistribution")` is translated to  $(-e1) + (-e2)$ ; result again of class "UnivariateDistribution"; is generated by simulations
- `signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution")`: both operands are coerced to class "UnivarLebDecDistribution" and the corresponding method is used.
- + `signature(e1 = "AbscontDistribution", e2 = "AbscontDistribution")` assumes  $e1, e2$  independent; result again of class "AbscontDistribution"; is generated by FFT
- + `signature(e1 = "AbscontDistribution", e2 = "DiscreteDistribution")` assumes  $e1, e2$  independent; result again of class "AbscontDistribution"; is generated by FFT
- + `signature(e1 = "DiscreteDistribution", e2 = "AbscontDistribution")` assumes  $e1, e2$  independent; result again of class "AbscontDistribution"; is generated by FFT
- + `signature(e1 = "LatticeDistribution", e2 = "LatticeDistribution")` assumes  $e1, e2$  independent; if the larger lattice-width is an integer multiple of the smaller(in abs. value) one: result again of class "LatticeDistribution"; is generated by D/FFT
- + `signature(e1 = "DiscreteDistribution", e2 = "DiscreteDistribution")` assumes  $e1, e2$  independent; result again of class "DiscreteDistribution"; is generated by explicite convolution
- + `signature(e1 = "LatticeDistribution", e2 = "DiscreteDistribution")` assumes  $e1, e2$  independent; result again of class "DiscreteDistribution"; is generated by explicite convolution
- + `signature(e1 = "UnivarLebDecDistribution", e2 = "UnivarLebDecDistribution")` assumes  $e1, e2$  independent; result again of class "UnivarLebDecDistribution"; is generated by separate explicite convolution of a.c. and discrete parts of  $e1$  and  $e2$  and subsequent flattening with `flat.LCD`; if `getdistrOption("withSimplify")` is TRUE, result is piped through a call to `simplifyD`
- + `signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution")`: both operands are coerced to class "UnivarLebDecDistribution" and the corresponding method is used.
- + `signature(e1 = "Binom", e2 = "Binom")` assumes  $e1, e2$  independent; if `prob(e1) == prob(e2)`, result again of class "Binom"; uses the convolution formula for binomial distributions; exact
- + `signature(e1 = "Cauchy", e2 = "Cauchy")` assumes  $e1, e2$  independent; result again of class "Cauchy"; uses the convolution formula for Cauchy distributions; exact
- + `signature(e1 = "Chisq", e2 = "Chisq")` assumes  $e1, e2$  independent; result again of class "Chisq"; uses the convolution formula for Chisq distributions; exact
- + `signature(e1 = "Dirac", e2 = "Dirac")` result again of class "Dirac"; exact
- + `signature(e1 = "ExpOrGammaOrChisq", e2 = "ExpOrGammaOrChisq")` assumes  $e1, e2$  independent; if  $e1, e2$  are Gamma distributions, result is of class "Gammad"; uses the convolution formula for Gamma distributions; exact
- + `signature(e1 = "Pois", e2 = "Pois")` assumes  $e1, e2$  independent; result again of class "Pois"; uses the convolution formula for Poisson distributions; exact
- + `signature(e1 = "Nbinom", e2 = "Nbinom")` assumes  $e1, e2$  independent; if `prob(e1) == prob(e2)`, result again of class "Nbinom"; uses the convolution formula for negative binomial distributions; exact
- + `signature(e1 = "Norm", e2 = "Norm")` assumes  $e1, e2$  independent; result again of class "Norm"; uses the convolution formula for normal distributions; exact

```

+ signature(e1 = "UnivariateDistribution", e2 = "Dirac") translated to e1 + location(e2);
  result again of class "Dirac"; exact
+ signature(e1 = "Dirac", e2 = "UnivariateDistribution") translated to e2 + location(e1);
  result again of class "Dirac"; exact
+ signature(e1 = "Dirac", e2 = "DiscreteDistribution") translated to e2 + location(e1);
  result again of class "Dirac"; exact
- signature(e1 = "Dirac", e2 = "Dirac") result again of class "Dirac"; exact
* signature(e1 = "Dirac", e2 = "Dirac") result again of class "Dirac"; exact
* signature(e1 = "UnivariateDistribution", e2 = "Dirac") translated to e1 * location(e2);
  result again of class "Dirac"; exact
* signature(e1 = "Dirac", e2 = "UnivariateDistribution") translated to e2 * location(e1);
  result again of class "Dirac"; exact
* signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"): by means of decomposePM
  e1 and e2 are decomposed into positive and negative parts; of these, convolutions of the cor-
  responding logarithms are computed separately and finally exp is applied to them, again sepa-
  rately; the resulting mixing components are then "flattened" to one object of class UnivarLebDecDistribution
  by flat.LCD which according to getdistrOption(withSimplify) gets piped through a call
  to simplifyD.
/ signature(e1 = "Dirac", e2 = "Dirac") result again of class "Dirac"; exact
/ signature(e1 = "numeric", e2 = "Dirac") result again of class "Dirac"; exact
/ signature(e1 = "numeric", e2 = "AcDcLcDistribution"): if d.discrete(e2)(0)*discreteWeight(e2)>0
  throws an error (would give division by 0 with positive probability); else by means of decomposePM
  e2 is decomposed into positive and negative parts; then, similarly the result obtains as for
  "*" (signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution")) by the exp-
  log trick and is "flattened" to one object of class UnivarLebDecDistribution by flat.LCD
  and according to getdistrOption(withSimplify) is piped through a call to simplifyD;
  exact..
/ signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"): translated to e1 * (1/e2).
^ signature(e1 = "AcDcLcDistribution", e2 = "Integer"): if e2=0 returns Dirac(1); if e2=1
  returns e1; if e2<0 translated to (1/e1)^(-e2); exact.
^ signature(e1 = "AcDcLcDistribution", e2 = "numeric"): if e2 is integer uses preceding item;
  else if e1<0 with positive probability, throughs an error; else the result obtains similarly to
  "*" (signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution")) by the exp-
  log trick and is "flattened" to one object of class UnivarLebDecDistribution by flat.LCD
  and according to getdistrOption(withSimplify) is piped through a call to simplifyD;
  exact.
^ signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"): if e1 is negative with
  positive probability, throws an error if e2 is non-integer with positive probability; if e1 is
  0 with positive probability throws an error if e2 is non-integer with positive probability.
  if e2 is integer with probability 1 uses DiscreteDistribution(supp=e1^(Dirac(x)) for
  each x in support(e2), builds up a corresponding mixing distribution; the latter is "flat-
  tened" to one object of class UnivarLebDecDistribution by flat.LCD and according to
  getdistrOption(withSimplify) is piped through a call to simplifyD. Else the result ob-
  tains similarly to "*" (signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"))

```

by the exp-log trick and is “flattened” to one object of class `UnivarLebDecDistribution` by `flat.LCD` and according to `getdistrOption(withSimplify)` is piped through a call to `simplifyD`; exact.

```
^ signature(e1 = "numeric", e2 = "AcDcLcDistribution"): if e1 is negative, throws an error
if e2 is non-integer with positive probability; if e1 is 0 throws an error if e2 is non-integer with
positive probability. if e2 is integer with probability 1 uses DiscreteDistribution(supp=e1^support(e2), prob=di
else the result obtains similarly to "*" (signature(e1 = "AcDcLcDistribution", e2 = "AcDcLcDistribution"))
by the exp-log trick and is “flattened” to one object of class UnivarLebDecDistribution
by flat.LCD and according to getdistrOption(withSimplify) is piped through a call to
simplifyD; exact.
```

## References

Ruckdeschel, P., Kohl, M.(2014): General purpose convolution algorithm for distributions in S4-Classes by means of FFT. *J. Statist. Softw.* **59**(4): 1-25.

## See Also

`UnivariateDistribution-class` `AbscontDistribution-class`  
`DiscreteDistribution-class` `LatticeDistribution-class`  
`Norm-class` `Binom-class` `Pois-class` `Dirac-class`  
`Cauchy-class` `Gammad-class` `Logis-class` `Lnorm-class`  
`Exp-class` `Weibull-class` `Nbinom-class`

## Examples

```
N <- Norm(0,3)
P <- Pois(4)
a <- 3
N + a
N + P
N - a
a * N
a * P
N / a + sin( a * P - N)
N * P
N / N

## takes a little time
N ^ P

1.2 ^ N
abs(N) ^ 1.3
```

---

OptionalParameter-class

*Classes "OptionalParameter", "OptionalMatrix"*

---

**Description**

auxiliary classes; may contain either a `Parameter` or `NULL`, resp. a matrix or `NULL` cf. J. Chambers, "green book".

**Objects from the Class**

"OptionalParameter" is a virtual Class: No objects may be created from it; "OptionalMatrix" is a class generated by `setClassUnion()` so may contain `NULL` or any matrix

**Methods**

No methods defined with class "OptionalParameter" in the signature.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Parameter-class](#), [AbscontDistribution-class](#)

---

options

*additional options in package 'distr'*

---

**Description**

In package **distr**, we add an extra option "newDevice"; it is inspected and manipulated as usual.

**Details**

We do not change the behaviour of `options` or `getOption`; for the general documentation to these two functions, confer [options](#), [getOption](#). Here we only document added options.

**Additionally available options in package 'distr'**

"newDevice" logical; controls behaviour when generating several plots within one function; if `TRUE`, before each call to `plot.new`, a call to `devNew` is inserted; if `FALSE`, we reproduce the usual behaviour in **graphics**, i.e.; we do not call `devNew`. Defaults to `FALSE`.

**Author(s)**

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

**See Also**[options](#), [getOption](#)**Examples**

```
getOption("newDevice")
options("newDevice"=TRUE)
```

p-methods

*Methods for Function p in Package 'distr'***Description**

p-methods

**Methods**

**p** signature(object = "Distribution"): returns the cumulative distribution function (c.d.f.), i.e.;  $p(t) = P(object \leq t)$

**p.r** signature(object = "Distribution"): from distr-2.6 onwards, we provide this as a synonym for method p; this synonym more explicitly states that we are dealing with the right-continuous variant of a c.d.f.

**See Also**[Distribution-class](#), [p.l](#)**Examples**

```
require(distr)
N <- Norm()
p(N)(0.3)
p.r(N)(0.3)
```

p.l-methods

*Methods for Function p.l in Package 'distr'***Description**

p-methods

Methods

return the left continuous cumulative distribution function, i.e.;  $p.l(t) = P(object < t)$

**p.l** signature(object = "AbscontDistribution")

**p.l** signature(object = "DiscreteDistribution")

**p.l** signature(object = "UnivarLebDecDistribution")

**p.l** signature(object = "UnivarMixingDistribution")

See Also

[DiscreteDistribution-class](#) [UnivarLebDecDistribution-class](#)

---

param-methods	<i>Methods for Function param in Package 'distr'</i>
---------------	--

---

Description

param-methods

Methods

**param** signature(object = "Distribution"): returns the parameter

---

Parameter-class	<i>Class "Parameter"</i>
-----------------	--------------------------

---

Description

Parameter is the mother-class of all Parameter classes.

Objects from the Class

Objects can be created by calls of the form new("Parameter").

Slots

**name** Object of class "character": a name / comment for the parameters

Methods

**name** signature(object = "Parameter"): returns the name of the parameter

**name<-** signature(object = "Parameter"): modifies the name of the parameter

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Distribution-class](#)

---

pivot-methods

*Methods for Function pivot in Package ‘distr’*

---

**Description**

pivot-methods

**Methods**

**pivot** signature(object = "Lattice"): returns the slot pivot of the lattice  
**pivot<-** signature(object = "Lattice"): modifies the slot pivot of the lattice  
**pivot** signature(object = "LatticeDistribution"): returns the slot pivot of the lattice slot of the distribution  
**pivot<-** signature(object = "LatticeDistribution"): modifies the slot pivot of the lattice slot of the distribution

---

plot-methods

*Methods for Function plot in Package ‘distr’*

---

**Description**

plot-methods

**Usage**

```
plot(x, y, ...)
## S4 method for signature 'AbscontDistribution,missing'
plot(x, width = 10, height = 5.5,
     withSweave = getdistrOption("withSweave"), xlim = NULL, ylim = NULL,
     ngrid = 1000, verticals = TRUE, do.points = TRUE, main = FALSE,
     inner = TRUE, sub = FALSE, bmar = par("mar")[1], tmar = par("mar")[3], ...,
     cex.main = par("cex.main"), cex.inner = 1.2, cex.sub = par("cex.sub"),
     col.points = par("col"), col.vert = par("col"), col.main = par("col.main"),
     col.inner = par("col.main"), col.sub = par("col.sub"), cex.points = 2.0,
```

```

    pch.u = 21, pch.a = 16, mfColRow = TRUE,
    to.draw.arg = NULL, withSubst = TRUE)
## S4 method for signature 'DiscreteDistribution,missing'
plot(x, width = 10, height = 5.5,
     withSweave = getdistrOption("withSweave"), xlim = NULL, ylim = NULL,
     verticals = TRUE, do.points = TRUE, main = FALSE, inner = TRUE, sub = FALSE,
     bmar = par("mar")[1], tmar = par("mar")[3], ...,
     cex.main = par("cex.main"), cex.inner = 1.2, cex.sub = par("cex.sub"),
     col.points = par("col"), col.hor = par("col"), col.vert = par("col"),
     col.main = par("col.main"), col.inner = par("col.main"),
     col.sub = par("col.sub"), cex.points = 2.0, pch.u = 21, pch.a = 16,
     mfColRow = TRUE, to.draw.arg = NULL, withSubst = TRUE)
## S4 method for signature 'AffLinUnivarLebDecDistribution,missing'
plot(x, width = 10,
     height = 5.5, withSweave = getdistrOption("withSweave"), xlim = NULL,
     ylim = NULL, ngrid = 1000, verticals = TRUE, do.points = TRUE, main = FALSE,
     inner = TRUE, sub = FALSE, bmar = par("mar")[1], tmar = par("mar")[3], ...,
     cex.main = par("cex.main"), cex.inner = 1.2, cex.sub = par("cex.sub"),
     col.points = par("col"), col.hor = par("col"), col.vert = par("col"),
     col.main = par("col.main"), col.inner = par("col.main"),
     col.sub = par("col.sub"), cex.points = 2.0, pch.u = 21, pch.a = 16,
     mfColRow = TRUE, to.draw.arg = NULL, withSubst = TRUE)
## S4 method for signature 'UnivarLebDecDistribution,missing'
plot(x, width = 10,
     height = 14.5, withSweave = getdistrOption("withSweave"), xlim = NULL,
     ylim = NULL, ngrid = 1000, verticals = TRUE, do.points = TRUE, main = FALSE,
     inner = TRUE, sub = FALSE, bmar = par("mar")[1], tmar = par("mar")[3], ...,
     cex.main = par("cex.main"), cex.inner = 0.9, cex.sub = par("cex.sub"),
     col.points = par("col"), col.hor = par("col"), col.vert = par("col"),
     col.main = par("col.main"), col.inner = par("col.main"),
     col.sub = par("col.sub"), cex.points = 2.0, pch.u = 21, pch.a = 16,
     mfColRow = TRUE, to.draw.arg = NULL, withSubst = TRUE)
## S4 method for signature 'DistrList,missing'
plot(x, y, ...)
## S4 method for signature 'CompoundDistribution,missing'
plot(x, y, ...)

```

## Arguments

<code>x</code>	object of class "AffLinUnivarLebDecDistribution" or class "UnivarLebDecDistribution" or class "AbscontDistribution" or class "DiscreteDistribution" or class "DistrList": (list of) distribution(s) to be plotted
<code>y</code>	missing
<code>xlim</code>	the x limits ( <code>x1</code> , <code>x2</code> ) of the plot. Note that <code>x1 &gt; x2</code> is allowed and leads to a "reversed axis". As in <code>plot.default</code> .
<code>ylim</code>	the y limits of the plot. Either as in <code>plot.default</code> (i.e. a vector of length 2) or a vector of length 4, where the first two elements are the values for <code>ylim</code> in panel



	"d", and the last two elements are the values for ylim resp. xlim in panels "p", and "q".
width	width (in inches) of the graphics device opened
height	height (in inches) of the graphics device opened
withSweave	logical: if TRUE (for working with Sweave) no extra device is opened and height/width are not set
ngrid	integer: number of grid points used for plots of absolutely continuous distributions
main	logical: is a main title to be used? or just as argument main in <a href="#">plot.default</a> .
inner	logical: do panels for density/probability function - cdf - quantile function have their own titles? or list which is filled to length 3 (resp. 8 for class UnivarLebDecDistribution) if necessary (possibly using recycling rules): titles for density/probability function - cdf - quantile function (each of the same form as argument main in <a href="#">plot.default</a> )
sub	logical: is a sub-title to be used? or just as argument sub in <a href="#">plot.default</a> .
tmar	top margin – useful for non-standard main title sizes
bmar	bottom margin – useful for non-standard sub title sizes
verticals	logical: if TRUE, draw vertical lines at steps; as in <a href="#">plot.stepfun</a>
do.points	logical: if TRUE, draw also draw points at the (xlim restricted) knot locations; as in <a href="#">plot.stepfun</a>
cex.points	numeric; character expansion factor; as in <a href="#">plot.stepfun</a>
col.points	character or integer code; color of points; as in <a href="#">plot.stepfun</a>
col.hor	character or integer code; color of horizontal lines; as in <a href="#">plot.stepfun</a>
col.vert	character or integer code; color of vertical lines; as in <a href="#">plot.stepfun</a>
cex.main	magnification to be used for main titles relative to the current setting of cex; as in <a href="#">par</a>
cex.inner	magnification to be used for inner titles relative to the current setting of cex; as in <a href="#">par</a>
cex.sub	magnification to be used for sub titles relative to the current setting of cex; as in <a href="#">par</a>
col.main	character or integer code; color for the main title
col.inner	character or integer code; color for the inner title
col.sub	character or integer code; color for the sub title
pch.u	character or integer code; plotting characters or symbols for unattained value; see <a href="#">points</a>
pch.a	character or integer code; plotting characters or symbols for attained value; see <a href="#">points</a>
mfColRow	shall default partition in panels be used — defaults to TRUE

<code>to.draw.arg</code>	Either NULL (default; everything is plotted) or a vector of either integers (the indices of the subplots to be drawn) or characters — the names of the subplots to be drawn: in case of an object <code>x</code> of class <code>"DiscreteDistribution"</code> or <code>"AbscontDistribution"</code> <code>c("d","p","q")</code> for density, c.d.f. and quantile function; in case of <code>x</code> a proper <code>"UnivarLebDecDistribution"</code> (with pos. weights for both discrete and abs. continuous part) names are <code>c("p","q","d.c","p.c","q.c","d.d","p.d","q.d")</code> for c.d.f. and quantile function of the composed distribution and the respective three panels for the absolutely continuous and the discrete part, respectively;
<code>withSubst</code>	logical; if TRUE (default) pattern substitution for titles and labels is used; otherwise no substitution is used.
<code>...</code>	additional arguments for plot — see <a href="#">plot</a> , <a href="#">plot.default</a> , <a href="#">plot.stepfun</a>

## Details

**plot** signature(`x = "AffLinUnivarLebDecDistribution"`, `y = "missing"`): plots cumulative distribution function and the quantile function

**plot** signature(`x = "UnivarLebDecDistribution"`, `y = "missing"`): plots a set of eight plots: in the first row, it plots the cumulative distribution function and the quantile function; in the second row the absolutely continuous part (with density, cdf and quantile fct.), and in the last row the discrete part (with prob.fct., cdf and quantile fct.).

**plot** signature(`x = "CompoundDistribution"`, `y = "missing"`): coerces `x` to `"UnivarLebDecDistribution"` and uses the corresponding method.

**plot** signature(`x = "AbscontDistribution"`, `y = "missing"`): plots density, cumulative distribution function and the quantile function

**plot** signature(`x = "DiscreteDistribution"`, `y = "missing"`): plots probability function, cumulative distribution function and the quantile function

**plot** signature(`x = "DistrList"`, `y = "missing"`): plots a list of distributions

Any parameters of `plot.default` may be passed on to this particular plot method.

For main-, inner, and subtitles given as arguments `main`, `inner`, and `sub`, top and bottom margins are enlarged to 5 resp. 6 by default but may also be specified by `tmar` / `bmar` arguments. If `main` / `inner` / `sub` are logical then if the respective argument is FALSE nothing is done/plotted, but if it is TRUE, we use a default main title taking up the calling argument `x` in case of `main`, default inner titles taking up the class and (named) parameter slots of argument `x` in case of `inner`, and a "generated on <data>"-tag in case of `sub`. Of course, if `main` / `inner` / `sub` are character, this is used for the title; in case of `inner` it is then checked whether it has length 3. In all title and axis label arguments, if `withSubst` is TRUE, the following patterns are substituted:

"%C" class of argument `x`

"%P" parameters of `x` in form of a comma-separated list of <value>'s coerced to character

"%Q" parameters of `x` in form of a comma-separated list of <value>'s coerced to character and in parenthesis — unless empty; then ""

"%N" parameters of `x` in form of a comma-separated list <name> = <value> coerced to character

"%A" deparsed argument `x`

"%D" time/date-string when the plot was generated

If not explicitly set, `col.points`, `col.vert`, `col.hor`, `col.main`, `col.inner`, `col.sub` are set to `col` if this arg is given and else to `par("col")` resp. for the titles `par("col.main")`, `par("col.main")`, `par("col.sub")`.

If not explicitly set, `pch.a`, `pch.u` are set to `pch` if this arg is given and else to 16, 21, respectively.

If not explicitly set, `cex` is set to 1. If not explicitly set, `cex.points` is set to `$2.0 cex$` (if `cex` is given) and to 2.0 else.

If general plot arguments `xlab`, `ylab` are not specified, they are set to "x", "q", "p" for `xlab` and to "d(x)", "p(q)", "q(p)" for `ylab` for density, cdf and quantile function respectively. Otherwise, according to the respective content of `to.draw.arg`, it is supposed to be a list with one entry for each selected panel, i.e., in case `x` is an object of class `DiscreteDistribution` or `AbscontDistribution` a list of maximal length maximally 3, respectively, in case `x` is an object of class `UnivarLebDecDistribution`. In these label arguments, the same pattern substitutions are made as for titles. If no character substitutions and mathematical expressions are needed, character vectors of respective length instead of lists are also allowed for arguments `xlab`, `ylab`.

In addition, argument `...` may contain arguments `panel.first`, `panel.last`, i.e., hook expressions to be evaluated at the very beginning and at the very end of each panel (within the then valid coordinates). To be able to use these hooks for each panel individually, they may also be lists of expressions (of the same length as the number of panels and run through in the same order as the panels).

## Value

An S3 object of class `c("plotInfo", "DiagnInfo")`, i.e., a list containing the information needed to produce the respective plot, which at a later stage could be used by different graphic engines (like, e.g. `ggplot`) to produce the plot in a different framework. A more detailed description will follow in a subsequent version.

## See Also

[plot](#), [plot.default](#), [plot.stepfun](#), [par](#)

## Examples

```
plot(Binom(size = 4, prob = 0.3))
plot(Binom(size = 4, prob = 0.3), do.points = FALSE)
plot(Binom(size = 4, prob = 0.3), verticals = FALSE)
plot(Binom(size = 4, prob = 0.3), main = TRUE)
plot(Binom(size = 4, prob = 0.3), main = FALSE)
plot(Binom(size = 4, prob = 0.3), cex.points = 1.2, pch = 20)
plot(Binom(size = 4, prob = 0.3), xlab = list("a1", "a2", "a3"),
      ylab=list("p"="U", "q"="V", "d"="W"))
B <- Binom(size = 4, prob = 0.3)
plot(B, col = "red", col.points = "green", main = TRUE, col.main = "blue",
      col.sub = "orange", sub = TRUE, cex.sub = 0.6, col.inner = "brown")
plot(Nbinom(size = 4, prob = 0.3), cex.points = 1.2, col = "red",
      col.points = "green")
plot(Nbinom(size = 4, prob = 0.3), cex.points = 1.2, pch.u = 20, pch.a = 10)
plot(Norm(), main = TRUE, cex.main = 3, tmar = 6)
plot(Norm(), inner = FALSE, main = TRUE, cex.main = 3, tmar = 6)
```

```

plot(Norm(), lwd = 3, col = "red", ngrid = 200, lty = 3, las = 2)
plot(Norm(), main = "my Distribution: %A",
      inner = list(expression(paste(lambda,"-density of %C(%P)")), "CDF",
                    "Pseudo-inverse with param's %N"),
      sub = "this plot was correctly generated on %D",
      cex.inner = 0.9, cex.sub = 0.8)

plot(Norm(),panel.first=grid(4,4))
## does not (yet) work as desired:
plot(Norm(),panel.first=list(grid(5,5),grid(3,3),grid(4,4)))
li <- list(substitute(grid(5,5)),substitute(grid(3,3)),substitute(grid(4,4)))
plot(Norm(),panel.first=li)

plot(Cauchy())
plot(Cauchy(), xlim = c(-4,4))
plot(Chisq())
### the next ylab argument is just for illustration purposes
plot(Chisq(),mfColRow = FALSE,to.draw.arg="d",
      xlab="x",ylab=list(expression(paste(lambda,"-density of %C(%P)"))))
## substitution can be switched off
plot(Chisq(),mfColRow = FALSE,to.draw.arg="d",
      xlab="x",ylab=list(expression(paste(lambda,"-density of %C(%P)"))), withSubst=FALSE)
plot(Chisq(), log = "xy", ngrid = 100)
Ch <- Chisq(); setgaps(Ch); plot(Ch, do.points = FALSE)
setgaps(Ch, exactq = 3); plot(Ch, verticals = FALSE)
plot(Ch, cex = 1.2, pch.u = 20, pch.a = 10, col.points = "green",
      col.vert = "red")

## Not run: # to save time
## some distribution with gaps
wg <- flat.mix(UnivarMixingDistribution(Unif(0,1),Unif(4,5),
                                       withSimplify=FALSE))
# some Lebesgue decomposed distribution
mymix <- UnivarLebDecDistribution(acPart = wg, discretePart = Binom(4,.4),
                                acWeight = 0.4)
plot(mymix)
#
## selection of subpanels for plotting
N <- Norm()
par(mfrow=c(1,2))
plot(N, mfColRow = FALSE, to.draw.arg=c("d","q"))
plot(N, mfColRow = FALSE, to.draw.arg=c(2,3))
par(mfrow=c(1,1))

wg <- flat.mix(UnivarMixingDistribution(Unif(0,1),Unif(4,5),
                                       withSimplify=FALSE))
myLC <- UnivarLebDecDistribution(discretePart=Binom(3,.3), acPart = wg,
                                discreteWeight=.2)
layout(matrix(c(rep(1,6),2,2,3,3,4,4,5,5,5,6,6,6),
              nrow=3, byrow=TRUE))
plot(myLC,mfColRow = FALSE,
      to.draw.arg=c("p","d.c","p.c","q.c", "p.d","q.d"))

```

```

P <- Pois(2)
plot(as(P,"UnivarLebDecDistribution"),mfColRow = FALSE,to.draw.arg=c("d.d"))
### the next ylab argument is just for illustration purposes
plot(as(P,"UnivarLebDecDistribution"),mfColRow = FALSE,to.draw.arg=c("d.d"),
      xlab="x",ylab=list(expression(paste(lambda,"-density of %C(%P)"))))

## End(Not run)

```

Pois-class

Class "Pois"

## Description

The Poisson distribution has density

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

for  $x = 0, 1, 2, \dots$ . The mean and variance are  $E(X) = Var(X) = \lambda$ .

C.f. [rpois](#)

## Objects from the Class

Objects can be created by calls of the form `Pois(lambda)`. This object is a Poisson distribution.

## Slots

**img** Object of class "Naturals": The space of the image of this distribution has got dimension 1 and the name "Natural Space".

**param** Object of class "PoisParameter": the parameter of this distribution (lambda), declared at its instantiation

**r** Object of class "function": generates random numbers (calls function `rpois`)

**d** Object of class "function": density function (calls function `dpois`)

**p** Object of class "function": cumulative function (calls function `ppois`)

**q** Object of class "function": inverse of the cumulative function (calls function `qpois`). The quantile is defined as the smallest value  $x$  such that  $F(x) \geq p$ , where  $F$  is the distribution function.

**support** Object of class "numeric": a (sorted) vector containing the support of the discrete density 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 "DiscreteDistribution", directly. Class "UnivariateDistribution", by class "DiscreteDistribution".  
 Class "Distribution", by class "DiscreteDistribution".

**Methods**

**+** signature(e1 = "Pois", e2 = "Pois"): For the Poisson distribution the exact convolution formula is implemented thereby improving the general numerical approximation.

**initialize** signature(.Object = "Pois"): initialize method

**lambda** signature(object = "Pois"): returns the slot lambda of the parameter of the distribution

**lambda<-** signature(object = "Pois"): modifies the slot lambda of the parameter of the distribution

**Note**

Working with a computer, we use a finite interval as support which carries at least mass  $1 - \text{getdistrOption}(\text{"TruncQuantile"})$

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[PoisParameter-class](#) [DiscreteDistribution-class](#) [Naturals-class](#) [rpois](#)

**Examples**

```
P <- Pois(lambda = 1) # P is a Poisson distribution with lambda = 1.
r(P)(1) # one random number generated from this distribution, e.g. 1
d(P)(1) # Density of this distribution is 0.3678794 for x = 1.
p(P)(0.4) # Probability that x < 0.4 is 0.3678794.
q(P)(.1) # x = 0 is the smallest value x such that p(B)(x) >= 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.)(.) instead of q(.)(.)
lambda(P) # lambda of this distribution is 1.
lambda(P) <- 2 # lambda of this distribution is now 2.
R <- Pois(lambda = 3) # R is a Poisson distribution with lambda = 2.
S <- P + R # R is a Poisson distribution with lambda = 5(=2+3).
```

---

PoisParameter-class	Class "PoisParameter"
---------------------	-----------------------

---

### Description

The parameter of a Poisson distribution, used by Pois-class

### Objects from the Class

Objects can be created by calls of the form `new("PoisParameter", lambda)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Pois` is prepared.

### Slots

`lambda` Object of class "numeric": the lambda of a Poisson distribution

`name` Object of class "character": a name / comment for the parameters

### Extends

Class "Parameter", directly.

### Methods

**initialize** signature(.Object = "PoisParameter"): initialize method

**lambda** signature(object = "PoisParameter"): returns the slot lambda of the parameter of the distribution

**lambda<-** signature(object = "PoisParameter"): modifies the slot lambda of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Pois-class Parameter-class](#)

### Examples

```
W <- new("PoisParameter", lambda = 1)
lambda(W) # lambda of this distribution is 1.
lambda(W) <- 2 # lambda of this distribution is now 2.
```

---

PosDefSymmMatrix	<i>Generating functions for PosSemDefSymmMatrix-class resp. PosDefSymmMatrix-class</i>
------------------	--

---

## Description

Generates an object of class "PosSemDefSymmMatrix" resp. of class "PosDefSymmMatrix".

## Usage

```
PosSemDefSymmMatrix(mat)
PosDefSymmMatrix(mat)
```

## Arguments

mat                    A numeric positive-[semi-]definite, symmetric matrix with finite entries.

## Details

If mat is no matrix, as.matrix is applied.

## Value

Object of class "PosSemDefSymmMatrix" resp. of class "PosDefSymmMatrix"

## Author(s)

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

## See Also

[PosDefSymmMatrix-class](#)

## Examples

```
PosSemDefSymmMatrix(1)
PosSemDefSymmMatrix(diag(2))
PosDefSymmMatrix(1)
PosDefSymmMatrix(diag(2))
```



---

PosDefSymmMatrix-class

*Positive-[Semi-]definite, symmetric matrices*

---

## Description

The class of positive-[semi-]definite, symmetric matrices.

## Objects from the Class

Objects can be created by calls of the form `new("PosSemDefSymmMatrix", ...)` resp. `new("PosDefSymmMatrix", ...)`. More frequently they are created via the generating functions `PosSemDefSymmMatrix` resp. `PosDefSymmMatrix`.

## Slots

`.Data` Object of class "matrix". A numeric matrix with finite entries.

## Extends

Class "PosSemDefSymmMatrix", directly Class "matrix", from data part.

Class "structure", by class "matrix".

Class "array", by class "matrix".

Class "vector", by class "matrix", with explicit coerce.

Class "vector", by class "matrix", with explicit coerce.

## Author(s)

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

## See Also

[PosDefSymmMatrix](#), [matrix-class](#)

## Examples

```
new("PosDefSymmMatrix", diag(2))
```

---

print-methods

---

*Methods for Functions print/show in Package 'distr'*


---

## Description

print/show-methods

## Methods

**print** signature(x = "UnivariateDistribution"): returns the class of the object and its parameters

**show** signature(x = "UnivariateDistribution"): returns the class of the object and its parameters

---

prob-methods

---

*Methods for Function prob in Package 'distr'*


---

## Description

prob-methods

## Methods

**prob** signature(object = "BinomParameter"): returns the slot prop of the parameter of the distribution

**prob<-** signature(object = "BinomParameter"): modifies the slot prob of the parameter of the distribution

**prob** signature(object = "Binom"): returns the slot prop of the parameter of the distribution

**prob<-** signature(object = "Binom"): modifies the slot prob of the parameter of the distribution

**prob** signature(object = "NbinomParameter"): returns the slot prop of the parameter of the distribution

**prob<-** signature(object = "NbinomParameter"): modifies the slot prob of the parameter of the distribution

**prob** signature(object = "Nbinom"): returns the slot prop of the parameter of the distribution

**prob<-** signature(object = "Nbinom"): modifies the slot prob of the parameter of the distribution

**prob** signature(object = "GeomParameter"): returns the slot prop of the parameter of the distribution (deprecated from 1.9 on)

**prob<-** signature(object = "GeomParameter"): modifies the slot prob of the parameter of the distribution (deprecated from 1.9 on)

**prob** signature(object = "Geom"): returns the slot prop of the parameter of the distribution

- prob<-** signature(object = "Geom"): modifies the slot prob of the parameter of the distribution
- prob** signature(object = "DiscreteDistribution"): returns the (named) vector of probabilities for the support points of the distribution.
- prob<-** signature(object = "DiscreteDistribution"): generates a new object of class "DiscreteDistribution" with the same support as object as well as the same .withSim, .withArith, .lowerExact, .logExact slots.
- prob** signature(object = "UnivarLebDecDistribution"): returns a  $2 \times n$  matrix where n is the length of the support of the discrete part of the distribution; the first row named "cond" gives the vector of probabilities for the support points of the discrete part of the distribution (i.e.; conditional on being in the discrete part), the second row named "abs" is like the first one but multiplied with discreteWeight of the distribution, hence gives the absolute probabilities of the support points; the columns are named by the support values.

q-methods

*Methods for Function q in Package 'distr'*

## Description

q-methods

## Methods

- q** signature(save = "Distribution"): returns the (left-continuous) quantile function, i.e.;  $q(s) = \inf\{t \mid P(\text{object} \leq t) \geq s\}$
- q.l** signature(object = "Distribution"): from distr-2.6 onwards, we provide this as a synonym for method q; this synonym more explicitly states that we are dealing with the left-continuous variant of a quantile function. It is useful in particular when used from the console in RStudio, as RStudio catches calls to q() and treats them separately from usual R evaluation. The developers of RStudio have been asked to fix this and comply with standard R evaluation which explicitly allows overloading q() as we do it in this package, but so far have refused to do so, as they claim overloading q() was insane.

## See Also

[Distribution-class](#), [q.r](#)

## Examples

```
require(distr)
N <- Norm()
q(N)(0.3)
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
q.l(N)(0.3)
```

---

q.r-methods	<i>Methods for Function q.r in Package ‘distr’</i>
-------------	--

---

**Description**

q.r-methods

**Methods**

return the right-continuous quantile function, i.e.;  $q.r(s) = \sup\{t \mid P(\text{object} \leq t) \leq s\}$

**q.r** signature(object = "DiscreteDistribution")

**q.r** signature(object = "AbscontDistribution")

**q.r** signature(object = "UnivarLebDecDistribution")

**q.r** signature(object = "UnivarMixingDistribution")

**See Also**

[DiscreteDistribution-class](#) [UnivarLebDecDistribution-class](#)

---

qqbounds	<i>Computation of confidence intervals for qqplot</i>
----------	---

---

**Description**

We compute confidence intervals for QQ plots. These can be simultaneous (to check whether the whole data set is compatible) or pointwise (to check whether each (single) data point is compatible);

**Usage**

```
qqbounds(x,D,alpha,n,withConf.pw, withConf.sim,  
         exact.sCI=(n<100),exact.pCI=(n<100),  
         nosym.pCI = FALSE, debug = FALSE)
```

**Arguments**

x	data to be checked for compatibility with distribution D.
D	object of class "UnivariateDistribution", the assumed data distribution.
alpha	confidence level
n	sample size
withConf.pw	logical; shall pointwise confidence lines be computed?
withConf.sim	logical; shall simultaneous confidence lines be computed?

<code>exact.pCI</code>	logical; shall pointwise CIs be determined with exact Binomial distribution?
<code>exact.sCI</code>	logical; shall simultaneous CIs be determined with exact kolmogorov distribution?
<code>nosym.pCI</code>	logical; shall we use (shortest) asymmetric CIs?
<code>debug</code>	logical; if TRUE additional output to debug confidence bounds.

## Details

Both simultaneous and pointwise confidence intervals come in a finite-sample and an asymptotic version; the finite sample versions will get quite slow for large data sets `x`, so in these cases the asymptotic version will be preferable.

For simultaneous intervals, the finite sample version is based on C function "pkolmogorov2x" from package **stats**, while the asymptotic one uses R function `pkstwo` again from package **stats**, both taken from the code to [ks.test](#).

Both finite sample and asymptotic versions use the fact, that the distribution of the supremal distance between the empirical distribution  $\hat{F}_n$  and the corresponding theoretical one  $F$  (assuming data from  $F$ ) does not depend on  $F$  for continuous distribution  $F$  and leads to the Kolmogorov distribution (compare, e.g. Durbin(1973)). In case of  $F$  with jumps, the corresponding Kolmogorov distribution is used to produce conservative intervals.

For pointwise intervals, the finite sample version is based on corresponding binomial distributions, (compare e.g., Fisz(1963)), while the asymptotic one uses a CLT approximation for this binomial distribution. In fact, this approximation is only valid for distributions with strictly positive density at the evaluation quantiles.

In the finite sample version, the binomial distributions will in general not be symmetric, so that, by setting `nosym.pCI` to TRUE we may produce shortest asymmetric confidence intervals (albeit with a considerable computational effort).

The symmetric intervals returned by default will be conservative (which also applies to distributions with jumps in this case).

For distributions with jumps or with density (nearly) equal to 0 at the corresponding quantile, we use the approximation of  $(D-E(D))/sd(D)$  by the standard normal at these points; this latter approximation is only available if package **distrEx** is installed; otherwise the corresponding columns will be filled with NA.

## Value

A list with components `crit` — a matrix with the lower and upper confidence bounds, and `err` a logical vector of length 2.

Component `crit` is a matrix with `length(x)` rows and four columns `c("sim.left", "sim.right", "pw.left", "pw.right")`. Entries will be set to NA if the corresponding `x` component is not in `support(D)` or if the computation method returned an error or if the corresponding parts have not been required (if `withConf.pw` or `withConf.sim` is FALSE).

`err` has components `pw` —do we have a non-error return value for the computation of pointwise CI's (FALSE if `withConf.pw` is FALSE)— and `sim` —do we have a non-error return value for the computation of simultaneous CI's (FALSE if `withConf.sim` is FALSE).

**Author(s)**

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

**References**

- Durbin, J. (1973) *Distribution theory for tests based on the sample distribution function*. SIAM.
- Fisz, M. (1963). *Probability Theory and Mathematical Statistics*. 3rd ed. Wiley, New York.

**See Also**

`qqplot` from package **stats** – the standard QQ plot function, `ks.test` again from package **stats** for the implementation of the Kolmogorov distributions; `qqplot` from package **distr** for comparisons of distributions, and `qqplot` from package **distrMod** for comparisons of data with models, as well as `qqplot` for checking of corresponding robust estimators.

**Examples**

```
qqplot(Norm(15,sqrt(30)), Chisq(df=15))
## uses:
qqbounds(x = rnorm(30),Norm(),alpha=0.95,n=30,
          withConf.pw = TRUE, withConf.sim = TRUE,
          exact.sCI=TRUE ,exact.pCI= TRUE,
          nosym.pCI = FALSE)
qqbounds(x = rchisq(30,df=4),Chisq(df=4),alpha=0.95,n=30,
          withConf.pw = TRUE, withConf.sim = TRUE,
          exact.sCI=FALSE ,exact.pCI= FALSE,
          nosym.pCI = FALSE)
qqbounds(x = rchisq(30,df=4),Chisq(df=4),alpha=0.95,n=30,
          withConf.pw = TRUE, withConf.sim = TRUE,
          exact.sCI=TRUE ,exact.pCI= TRUE,
          nosym.pCI = TRUE)
```

---

qqplot

---

*Methods for Function qqplot in Package ‘distr’*


---

**Description**

We generalize function `qqplot` from package **stats** to be applicable to distribution objects. In this context, `qqplot` produces a QQ plot of two distributions, i.e.; argument `x` is the distribution to be checked for compatibility, and `y` is the model ( $H_0$ -)distribution. Graphical parameters may be given as arguments to `qqplot`. The **stats** function is just the method for signature `x=ANY,y=ANY`. In all title and axis label arguments, if `withSubst` is `TRUE`, the following patterns are substituted:

```
"%C" class of argument x
"%A" deparsed argument x
"%D" time/date-string when the plot was generated
```

**Usage**

```

qqplot(x, y, ...)
## S4 method for signature 'UnivariateDistribution,UnivariateDistribution'
qqplot(x, y,
  n = 30, withIdLine = TRUE, withConf = TRUE,
  withConf.pw = withConf, withConf.sim = withConf,
  plot.it = TRUE, xlab = deparse(substitute(x)),
  ylab = deparse(substitute(y)), ...,
  width = 10, height = 5.5, withSweave = getdistrOption("withSweave"),
  mfColRow = TRUE, n.CI = n, col.IdL = "red", lty.IdL = 2, lwd.IdL = 2,
  alpha.CI = .95, exact.pCI = (n<100), exact.sCI = (n<100), nosym.pCI = FALSE,
  col.pCI = "orange", lty.pCI = 3, lwd.pCI = 2, pch.pCI = par("pch"),
  cex.pCI = par("cex"),
  col.sCI = "tomato2", lty.sCI = 4, lwd.sCI = 2, pch.sCI = par("pch"),
  cex.sCI = par("cex"),
  cex.pch = par("cex"), col.pch = par("col"),
  jit.fac = 0, check.NotInSupport = TRUE,
  col.NotInSupport = "red", with.legend = TRUE, legend.bg = "white",
  legend.pos = "topleft", legend.cex = 0.8, legend.pref = "",
  legend.postf = "", legend.alpha = alpha.CI, debug = FALSE, withSubst = TRUE)
## S4 method for signature 'ANY,ANY'
qqplot(x, y,
  plot.it = TRUE, xlab = deparse(substitute(x)),
  ylab = deparse(substitute(y)), ...)

```

**Arguments**

<code>x</code>	object of class "ANY" ( <b>stats</b> -method) or of code "UnivariateDistribution"; to be compared to <code>y</code> .
<code>y</code>	object of class "ANY" ( <b>stats</b> -method) or of class "UnivariateDistribution".
<code>n</code>	numeric; number of quantiles at which to do the comparison.
<code>withIdLine</code>	logical; shall line $y = x$ be plotted in?
<code>withConf</code>	logical; shall confidence lines be plotted?
<code>withConf.pw</code>	logical; shall pointwise confidence lines be plotted?
<code>withConf.sim</code>	logical; shall simultaneous confidence lines be plotted?
<code>plot.it</code>	logical; shall be plotted at all (inherited from <a href="#">qqplot</a> )?
<code>xlab</code>	x-label
<code>ylab</code>	y-label
<code>...</code>	further parameters for function <code>plot</code>
<code>width</code>	width (in inches) of the graphics device opened
<code>height</code>	height (in inches) of the graphics device opened
<code>withSweave</code>	logical: if TRUE (for working with Sweave) no extra device is opened and height/width are not set
<code>mfColRow</code>	shall default partition in panels be used — defaults to TRUE

<code>n.CI</code>	numeric; number of points to be used for confidence interval
<code>col.IdL</code>	color for the identity line
<code>lty.IdL</code>	line type for the identity line
<code>lwd.IdL</code>	line width for the identity line
<code>alpha.CI</code>	confidence level
<code>exact.pCI</code>	logical; shall pointwise CIs be determined with exact Binomial distribution?
<code>exact.sCI</code>	logical; shall simultaneous CIs be determined with exact kolmogorov distribution?
<code>nosym.pCI</code>	logical; shall we use (shortest) asymmetric CIs?
<code>col.pCI</code>	color for the pointwise CI
<code>lty.pCI</code>	line type for the pointwise CI
<code>lwd.pCI</code>	line width for the pointwise CI
<code>pch.pCI</code>	symbol for points (for discrete mass points) in pointwise CI
<code>cex.pCI</code>	magnification factor for points (for discrete mass points) in pointwise CI
<code>col.sCI</code>	color for the simultaneous CI
<code>lty.sCI</code>	line type for the simultaneous CI
<code>lwd.sCI</code>	line width for the simultaneous CI
<code>pch.sCI</code>	symbol for points (for discrete mass points) in simultaneous CI
<code>cex.sCI</code>	magnification factor for points (for discrete mass points) in simultaneous CI
<code>cex.pch</code>	magnification factor for the plotted symbols
<code>col.pch</code>	color for the plotted symbols
<code>jit.fac</code>	jittering factor used for discrete distributions
<code>check.NotInSupport</code>	logical; shall we check if all x-quantiles lie in support(y)?
<code>col.NotInSupport</code>	logical; if preceding check TRUE color of x-quantiles if not in support(y)
<code>with.legend</code>	logical; shall a legend be plotted?
<code>legend.bg</code>	background color for the legend
<code>legend.pos</code>	position for the legend
<code>legend.cex</code>	magnification factor for the legend
<code>legend.pref</code>	character to be prepended to legend text
<code>legend.postf</code>	character to be appended to legend text
<code>legend.alpha</code>	nominal coverage probability
<code>debug</code>	logical; if TRUE additional output to debug confidence bounds.
<code>withSubst</code>	logical; if TRUE (default) pattern substitution for titles and lables is used; otherwise no substitution is used.



## Details

**qqplot** signature(x = "ANY", y = "ANY"): function qqplot from package **stats**.

**qqplot** signature(x = "UnivariateDistribution", y = "UnivariateDistribution"): produces a QQ plot for two univariate distributions.

## Value

A list of elements containing the information needed to compute the respective QQ plot, in particular it extends the elements of the return value of function [qqplot](#) from package **stats**, i.e., a list with components x and y for x and y coordinates of the plotted points; more specifically it contains

x	The x coordinates of the points that were/would be plotted
y	The corresponding quantiles of the second distribution, <i>including NAs</i> .
crit	A matrix with the lower and upper confidence bounds (computed by <a href="#">qqbounds</a> ).
err	logical vector of length 2.

(elements `crit` and `err` are taken from the return value(s) of [qqbounds](#)). The return value allows to recover all information used to produce the plot for later use in enhanced graphics (e.g. with [ggplot](#)).

## Author(s)

Peter Ruckdeschel <[peter.ruckdeschel@uni-oldenburg.de](mailto:peter.ruckdeschel@uni-oldenburg.de)>

## References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

## See Also

[qqplot](#) from package **stats** – the standard QQ plot function and [qqbounds](#), used by [qqplot](#) to produce confidence intervals.

## Examples

```
qqplot(Norm(15,sqrt(30)), Chisq(df=15))
## some discrete Distributions:
P <- Pois(5)
B <- Binom(size=2000,prob=5/2000)
qqplot(B,P)

## takes too much time for R CMD check --as-cran
qqplot(B,P, nosym.pCI=TRUE)

## some Lebesgue-Decomposed distributions:
mylist <- UnivarLebDecDistribution(discretePart=Binom(3,.3), acPart=Norm(2,2),
                                acWeight=11/20)
mylist2 <- mylist+0.1
qqplot(mylist,mylist2)
```

```

qqplot(mylist,mylist2,exact.pCI=FALSE,exact.sCI=FALSE)

## takes too much time for R CMD check --as-cran
qqplot(mylist,mylist2,nosym.pCI=TRUE)
## some ac. distribution with a gap
mylist3 <- UnivarMixingDistribution(Unif(0,0.3),Unif(0.6,1),mixCoeff=c(0.8,0.2))
gaps(mylist3)
mylist4 <- UnivarMixingDistribution(Unif(0,0.3),Unif(0.6,1),mixCoeff=c(0.6,0.4))
qqplot(mylist3,mylist4)
qqplot(mylist3,mylist4,nosym.pCI=TRUE)

```

---

r-methods

*Methods for Function r in Package ‘distr’*


---

### Description

r-methods

### Methods

**r** signature(object = "Distribution"): generates random deviates according to the distribution

### See Also

[Distribution-class](#)

---

rate-methods

*Methods for Function rate in Package ‘distr’*


---

### Description

rate-methods

### Methods

**rate** signature(object = "ExpParameter"): returns the slot rate of the parameter of the distribution

**rate<-** signature(object = "ExpParameter"): modifies the slot rate of the parameter of the distribution

**rate** signature(object = "Exp"): returns the slot rate of the parameter of the distribution

**rate<-** signature(object = "Exp"): modifies the slot rate of the parameter of the distribution

**rate** signature(object = "DExp"): returns the slot rate of the parameter of the distribution

**rate<-** signature(object = "DExp"): modifies the slot rate of the parameter of the distribution

Reals-class

Class "Reals"

**Description**

Particular case of a one-dimensional Euclidean Space

**Usage**

```
Reals()
```

**Objects from the Class**

Objects could theoretically be created by calls of the form `new("Reals", dimension, name)`. Usually an object of this class is not needed on its own. It is generated automatically when a univariate absolutely continuous distribution is instantiated.

**Slots**

`dimension` Object of class "character": the dimension of the space, by default = 1

`name` Object of class "character": the name of the space, by default = "Real Space"

**Extends**

Class "EuclideanSpace", directly.

Class "rSpace", by class "EuclideanSpace".

**Methods**

**initialize** signature(.Object = "Reals"): initialize method

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[EuclideanSpace-class](#) [Naturals-class](#) [AbscontDistribution-class](#)

**Examples**

```
R <- Reals()
liesIn(R, c(0, 0)) # FALSE
```

---

rSpace-class

Class "rSpace"

---

### Description

The distribution-classes contain a slot where the sample space is stored. Typically, discrete random variables take naturals as values. rSpace is the mother-class of the class EuclideanSpace.

### Objects from the Class

A virtual Class: No objects may be created from it.

### Slots

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

### Methods

**name** signature(object = "rSpace"): returns the name of the space

**name<-** signature(object = "rSpace"): changes the name of the space

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

### See Also

[Lattice-class](#) [Naturals-class](#) [EuclideanSpace-class](#) [Distribution-class](#)

---

RtoDPQ

Default procedure to fill slots d,p,q given r for a.c. distributions

---

### Description

function to do get empirical density, cumulative distribution and quantile function from random numbers

### Usage

```
RtoDPQ(r, e = getdistrOption("RtoDPQ.e"),
       n = getdistrOption("DefaultNrGridPoints"), y = NULL)
```

**Arguments**

<code>r</code>	the random number generator
<code>e</code>	$10^e$ numbers are generated, a higher number leads to a better result.
<code>n</code>	The number of grid points used to create the approximated functions, a higher number leads to a better result.
<code>y</code>	a (numeric) vector or NULL

**Details**

RtoDPQ generates  $10^e$  random numbers, by default

$$e = RtoDPQ.e$$

. Instead of using simulated grid points, we have an optional parameter `y` for using N. Horbenko's quantile trick: i.e.; on an equally spaced grid `x.grid` on  $[0,1]$ , apply  $f(q(x)(x.grid))$  and write the result to `y` and produce density and cdf from this value `y` given to RtoDPQ as argument (instead of simulating grid points).

The density is formed on the basis of  $n$  points using `approxfun` and `density`, by default

$$n = DefaultNrGridPoints$$

. The cumulative distribution function and the quantile function are also created on the basis of  $n$  points using `approxfun` and `ecdf`. Of course, the results are usually not exact as they rely on random numbers.

**Value**

RtoDPQ returns a list of functions.

<code>dfun</code>	density
<code>pfun</code>	cumulative distribution function
<code>qfun</code>	quantile function

**Note**

Use RtoDPQ for absolutely continuous and RtoDPQ.d for discrete distributions.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[UnivariateDistribution-class](#), [density](#), [approxfun](#), [ecdf](#)

**Examples**

```

rn2 <- function(n){rnorm(n)^2}
x <- RtoDPQ(r = rn2, e = 4, n = 512)
# returns density, cumulative distribution and quantile function of
# squared standard normal distribution
x$dfun(4)
RtoDPQ(r = rn2, e = 5, n = 1024) # for a better result

rp2 <- function(n){rpois(n, lambda = 1)^2}
x <- RtoDPQ.d(r = rp2, e = 5)
# returns density, cumulative distribution and quantile function of
# squared Poisson distribution with parameter lambda=1

```

RtoDPQ.d

*Default procedure to fill slots d,p,q given r for discrete distributions***Description**

function to do get empirical density, cumulative distribution and quantile function from random numbers

**Usage**

```
RtoDPQ.d(r, e = getdistrOption("RtoDPQ.e"))
```

**Arguments**

**r** the random number generator

**e**  $10^e$  numbers are generated, a higher number leads to a better result.

**Details**

RtoDPQ.d generates  $10^e$  random numbers, by default  $e = \text{RtoDPQ.e}$  which are used to produce a density, cdf and quantile function. Of course, the results are usually not exact as they rely on random numbers.

**Value**

RtoDPQ returns a list of functions.

**dfun** density

**pfun** cumulative distribution function

**qfun** quantile function

**Note**

Use RtoDPQ for absolutely continuous and RtoDPQ.d for discrete distributions.

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[UnivariateDistribution-class](#), [density](#), [approxfun](#), [ecdf](#)

**Examples**

```
rn2 <- function(n){rnorm(n)^2}
x <- RtoDPQ(r = rn2, e = 4, n = 512)
# returns density, cumulative distribution and quantile function of
# squared standard normal distribution

x$dfun(4)
RtoDPQ(r = rn2, e = 5, n = 1024) # for a better result

rp2 <- function(n){rpois(n, lambda = 1)^2}
x <- RtoDPQ.d(r = rp2, e = 5)
# returns density, cumulative distribution and quantile function of
# squared Poisson distribution with parameter lambda=1
```

---

RtoDPQ.LC

*Default procedure to fill slots d,p,q given r for Lebesgue decomposed distributions*

---

**Description**

function to do get empirical density, cumulative distribution and quantile function from random numbers

**Usage**

```
RtoDPQ.LC(r, e = getdistrOption("RtoDPQ.e"),
          n = getdistrOption("DefaultNrGridPoints"), y = NULL)
```

**Arguments**

r	the random number generator
e	10 <sup>e</sup> numbers are generated, a higher number leads to a better result.
n	The number of grid points used to create the approximated functions, a higher number leads to a better result.
y	a (numeric) vector or NULL

## Details

RtoDPQ.LC generates  $10^e$  random numbers, by default

$$e = RtoDPQ.e$$

. Replicates are assumed to be part of the discrete part, unique values to be part of the a.c. part of the distribution. For the replicated ones, we generate a discrete distribution by a call to [DiscreteDistribution](#).

For the a.c. part, similarly to [RtoDPQ](#) we have an optional parameter `y` for using N. Horbenko's quantile trick: i.e.; on an equally spaced grid `x.grid` on  $[0,1]$ , apply  $f(q(x)(x.grid))$ , write the result to `y` and use these values instead of simulated ones.

The a.c. density is formed on the basis of  $n$  points using `approxfun` and `density` (applied to the unique values), by default

$$n = DefaultNrGridPoints$$

. The cumulative distribution function is based on all random variables, and, as well as the quantile function, is also created on the basis of  $n$  points using `approxfun` and `ecdf`. Of course, the results are usually not exact as they rely on random numbers.

## Value

RtoDPQ.LC returns an object of class `UnivarLebDecDistribution`.

## Note

Use `RtoDPQ` for absolutely continuous and `RtoDPQ.d` for discrete distributions.

## Author(s)

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

## See Also

[UnivariateDistribution-class](#), [density](#), [approxfun](#), [ecdf](#)

## Examples

```
rn2 <- function(n)ifelse(rbinom(n,1,0.3),rnorm(n)^2,rbinom(n,4,.3))
x <- RtoDPQ.LC(r = rn2, e = 4, n = 512)
plot(x)
# returns density, cumulative distribution and quantile function of
# squared standard normal distribution
d.discrete(x)(4)
x2 <- RtoDPQ.LC(r = rn2, e = 5, n = 1024) # for a better result
plot(x2)
```



**Description**

scale-methods

**Methods**

**scale** signature(object = "GammaParameter"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "GammaParameter"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "Gammad"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Gammad"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "LogisParameter"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "LogisParameter"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "Logis"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Logis"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "WeibullParameter"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "WeibullParameter"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "Weibull"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Weibull"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "CauchyParameter"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "CauchyParameter"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "Cauchy"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Cauchy"): modifies the slot scale of the parameter of the distribution

**scale** signature(object = "Chisq"): if ncp(object) is 0, a Chi-squared distribution is also a Gamma distribution; in this case, scale returns 2 else an error;

---

sd-methods

---

*Methods for Function sd in Package ‘distr’*


---

## Description

sd-methods

## Methods

**sd** signature(x = "Any"): see [sd](#)

**sd** signature(x = "NormParameter"): returns the slot sd of the parameter of the distribution

**sd<-** signature(object = "NormParameter"): modifies the slot sd of the parameter of the distribution

**sd** signature(x = "Norm"): returns the slot sd of the parameter of the distribution

**sd<-** signature(object = "Norm"): modifies the slot sd of the parameter of the distribution

## See Also

[sd](#)

---

sdlog-methods

---

*Methods for Function sdlog in Package ‘distr’*


---

## Description

sdlog-methods

## Methods

**sdlog** signature(object = "LnormParameter"): returns the slot sdlog of the parameter of the distribution

**sdlog<-** signature(object = "LnormParameter"): modifies the slot sdlog of the parameter of the distribution

**sdlog** signature(object = "Lnorm"): returns the slot sdlog of the parameter of the distribution

**sdlog<-** signature(object = "Lnorm"): modifies the slot sdlog of the parameter of the distribution

---

 shape-methods

*Methods for Function shape in Package 'distr'*


---

## Description

shape-methods

## Methods

**shape** signature(object = "GammaParameter"): returns the slot shape of a parameter of a Gamma distribution

**shape<-** signature(object = "GammaParameter"): modifies the slot shape of a parameter of a Gamma distribution

**shape** signature(object = "Gammad"): returns the slot shape of the parameter slot of a Gamma distribution

**shape<-** signature(object = "Gammad"): modifies the slot shape of the parameter slot of a Gamma distribution

**shape** signature(object = "WeibullParameter"): returns the slot shape of a parameter of a Weibull distribution

**shape<-** signature(object = "WeibullParameter"): modifies the slot shape of a parameter of a Weibull distribution

**shape** signature(object = "Weibull"): returns the slot shape of the parameter slot of the distribution

**shape<-** signature(object = "Weibull"): modifies the slot shape of the parameter slot of the distribution

**shape** signature(object = "Chisq"): if ncp(object) is 0, a Chi-squared distribution is also a Gamma distribution; in this case, shape returns df(object)/2 else an error;

**shape** signature(object = "Exp"): returns the slot shape of the parameter slot of the Exp distribution (=1)

---

 shape1-methods

*Methods for Function shape1 in Package 'distr'*


---

## Description

shape-methods

Methods

- shape1** signature(object = "BetaParameter"): returns the slot shape1 of the parameter of the distribution
- shape1<-** signature(object = "BetaParameter"): modifies the slot shape1 of the parameter of the distribution
- shape1** signature(object = "Beta"): returns the slot shape1 of the parameter of the distribution
- shape1<-** signature(object = "Beta"): modifies the slot shape1 of the parameter of the distribution

---

shape2-methods	<i>Methods for Function shape2 in Package ‘distr’</i>
----------------	---

---

Description

shape-methods

Methods

- shape2** signature(object = "BetaParameter"): returns the slot shape2 of the parameter of the distribution
- shape2<-** signature(object = "BetaParameter"): modifies the slot shape2 of the parameter of the distribution
- shape2** signature(object = "Beta"): returns the slot shape2 of the parameter of the distribution
- shape2<-** signature(object = "Beta"): modifies the slot shape2 of the parameter of the distribution

---

simplifyD-methods	<i>Methods for function simplifyD in Package ‘distr’</i>
-------------------	--

---

Description

simplifyD-methods

Usage

simplifyD(object)

Arguments

object                      distribution object

## Details

generating functions [UnivarMixingDistribution](#), [Minimum](#), [Maximum](#), [Truncate](#), and [Huberize](#) have an argument `withSimplify` which decides whether the respective result is filtered by/piped through a call to `simplifyD`. By default this argument is set to the distr-option `getdistrOption("simplifyD")` (for the inspection and modification of such global options see [distrOptions](#)). Depending on whether or not this option is TRUE, also arithmetic operations `"+"`, `"*"`, `"/"`, `"^"` and group `Math` give results filtered by/piped through a call to `simplifyD`.

## Value

the corresponding, possibly simplified distribution

## Methods

**simplifyD** signature(object = "AbscontDistribution"): returns object unchanged

**simplifyD** signature(object = "DiscreteDistribution"): returns object unchanged

**simplifyD** signature(object = "UnivarLebDecDistribution"): checks whether `acWeight` or `discreteWeight` is approximately (i.e.; up to `getdistrOption("TruncQuantile")`) zero and if so, accordingly returns `discretePart(object)` or `acPart(object)`, respectively.

**simplifyD** signature(object = "UnivarMixingDistribution"): returns the flattened version of object (using [flat.mix](#)). before doing so, it checks whether any component carries weight approximately (i.e.; up to `getdistrOption("TruncQuantile")`) one (in slot `mixCoeff`) and if so, returns this component; else, if not all weights are below `getdistrOption("TruncQuantile")`, it filters out those components with weight less than `getdistrOption("TruncQuantile")`.

## See Also

[Huberize](#), [Minimum](#)

## Examples

```
set.seed(123)
Mix1 <- UnivarMixingDistribution(Norm(), Binom(2, .3),
  UnivarLebDecDistribution(acPart = Chisq(df = 2), discretePart = Nbinom(3, .09),
    acWeight = 0.3),
  Norm()-Chisq(df=3), mixCoeff=c(0,0,0.2,0.8), withSimplify = FALSE)
Mix2 <- UnivarMixingDistribution(Norm(), Mix1, DExp(2),
  mixCoeff = c(0,0.2,0.8), withSimplify = FALSE)
Mix2
simplifyD(Mix2)
```

simplifyr-methods

*Methods for Function simplifyr in Package 'distr'***Description**

simplifyr-methods

**Methods**

**simplifyr** signature(.Object = "UnivariateDistribution"): After several transformations of a given distribution it may take quite a long time to generate random numbers from the resulting distribution. **simplifyr** generates a certain number, by default  $10^5$ , of random numbers once. This pool of random numbers forms the basis for further uses of the **r**-method. That is, random numbers are generated by sampling with replacement out of this pool.

**Note**

If you want to generate many random numbers, you should use **simplifyr** with a big size to be sure, that your numbers are really random.

**See Also**

[Distribution-class](#)

**Examples**

```
F <- ( Norm() + Binom() + Pois() + Exp() ) * 2 - 10
system.time(r(F)(10^6))
simplifyr(F, size = 10^6)
system.time(r(F)(10^6))
```

size-methods

*Methods for Function size in Package 'distr'***Description**

size-methods

**Methods**

**size** signature(object = "BinomParameter"): returns the slot size of the parameter of the distribution

**size<-** signature(object = "BinomParameter"): modifies the slot size of the parameter of the distribution

**size** signature(object = "Binom"): returns the slot size of the parameter of the distribution

**size**<- signature(object = "Binom"): modifies the slot size of the parameter of the distribution

**size** signature(object = "NbinomParameter"): returns the slot size of the parameter of the distribution

**size**<- signature(object = "NbinomParameter"): modifies the slot size of the parameter of the distribution

**size** signature(object = "Nbinom"): returns the slot size of the parameter of the distribution

**size**<- signature(object = "Nbinom"): modifies the slot size of the parameter of the distribution

**size** signature(object = "Geom"): returns the slot size of the parameter of the distribution

---

solve-methods

---

*Methods for Function solve in Package 'distr'*


---

## Description

solve-methods using generalized inverses for various types of matrices

## Usage

```
solve(a,b, ...)
## S4 method for signature 'ANY,ANY'
solve(a, b, generalized =
getdistrOption("use.generalized.inverse.by.default"), tol = 1e-10)
## S4 method for signature 'PosSemDefSymmMatrix,ANY'
solve(a, b, generalized =
getdistrOption("use.generalized.inverse.by.default"), tol = 1e-10)
## S4 method for signature 'PosDefSymmMatrix,ANY'
solve(a, b, tol = 1e-10)
```

## Arguments

a	matrix to be inverted / to be solved for RHS.
b	a numeric or complex vector or matrix giving the right-hand side(s) of the linear system. If missing, b is taken to be an identity matrix and solve will return the inverse of a.
...	further arguments to be passed to specific methods (see <a href="#">solve</a> ).
generalized	logical: should generalized / Moore-Penrose inverses be used? By default uses the corresponding global option to be set by <a href="#">distrOptions</a> .
tol	the tolerance for detecting linear dependencies in the columns of a. Default is <code>.Machine\$double.eps</code> .

**Details**

The method for the Moore-Penrose inverse for `signature(a = "PosSemDefSymmMatrix", b = "ANY")` uses `eigen` to find the eigenvalue decomposition of `a` and then simply "pseudo-inverts" the corresponding diagonal matrix built from `eigen(a)$values`, while for `signature(a = "ANY", b = "ANY")` it uses the `svd` decomposition of `a` and then simply "pseudo-inverts" the corresponding diagonal matrix built from `svd(a)$d`.

**Methods**

**solve** `signature(a = "ANY", b = "ANY")`: tries to evaluate `solve.default` method from **base** in classical way; if this gives an error, this one is returned if `generalized` is `TRUE`, else it will then return  $a^-b$  where  $a^-$  is the pseudo or Moore-Penrose inverse of  $a$ .

**solve** `signature(a = "PosSemDefSymmMatrix", b = "ANY")`: evaluates  $a^-b$  where  $a^-$  is the pseudo or Moore-Penrose inverse of  $a$ .

**solve** `signature(a = "PosDefSymmMatrix", b = "ANY")`: evaluates `solve` method from **base** in classical way.

**Author(s)**

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

**See Also**

[solve](#) for the default method, [eigen](#) and [svd](#) for the pseudo inversion

---

SphericalSymmetry

---

*Generating function for SphericalSymmetry-class*


---

**Description**

Generates an object of class "SphericalSymmetry".

**Usage**

```
SphericalSymmetry(SymmCenter = 0)
```

**Arguments**

`SymmCenter`      numeric: center of symmetry

**Value**

Object of class "SphericalSymmetry"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>



**See Also**

[SphericalSymmetry-class](#), [DistributionSymmetry-class](#)

**Examples**

```
SphericalSymmetry()  
  
## The function is currently defined as  
function(SymmCenter = 0){  
  new("SphericalSymmetry", SymmCenter = SymmCenter)  
}
```

---

SphericalSymmetry-class

*Class for Spherical Symmetric Distributions*

---

**Description**

Class for spherical symmetric distributions.

**Objects from the Class**

Objects can be created by calls of the form `new("SphericalSymmetry")`. More frequently they are created via the generating function `SphericalSymmetry`. Spherical symmetry for instance leads to a simplification for the computation of optimally robust influence curves.

**Slots**

`type` Object of class "character": contains "spherical symmetric distribution"  
`SymmCenter` Object of class "numeric": center of symmetry

**Extends**

Class "EllipticalSymmetry", directly.  
Class "DistributionSymmetry", by class "EllipticalSymmetry".  
Class "Symmetry", by class "EllipticalSymmetry".

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[SphericalSymmetry](#), [DistributionSymmetry-class](#)

**Examples**

```
new("SphericalSymmetry")
```

---

sqrt-methods	<i>Methods for Function sqrt in Package ‘distr’</i>
--------------	---

---

**Description**

sqrt-methods using generalized inverses for p.s.d. matrices

**Usage**

```
## S4 method for signature 'PosSemDefSymmMatrix'
sqrt(x)
```

**Arguments**

x                      a p.s.d. matrix (of class PosSemDefSymmMatrix)

**Methods**

**sqrt** signature(x = "PosSemDefSymmMatrix"): produces a symmetric, p.s.d. matrix  $y$  such that  $x = y^2$ .

**Author(s)**

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

**See Also**

[solve](#)

---

standardMethods	<i>Utility to automatically generate accessor and replacement functions</i>
-----------------	---

---

**Description**

Creates definitions for accessor and replacement functions of an given class.

**Usage**

```
standardMethods(class, writetofile = FALSE, directory)
```

**Arguments**

class	the class for which accessor and replacement functions are to be produced, given as a string
writetofile	logical value, indicating wheter output is to be written to a file
directory	if writetofile = TRUE, the output is written to a file in the given directory, the name of the file starting with "classname" and ending with "StandardMethods.txt"

**Value**

no value is returned

**Author(s)**

Thomas Stabla <statho@web.de>

**Examples**

```
setClass("testclass", representation(a = "numeric", b = "character"))
standardMethods("testclass")
```

---

support-methods

*Methods for Function support in Package 'distr'*


---

**Description**

support-methods

**Methods**

**support** signature(object = "DiscreteDistribution"): returns the support

---

Symmetry-class

*Class of Symmetries*


---

**Description**

Class of symmetries of various objects.

**Objects from the Class**

A virtual Class: No objects may be created from it.

**Slots**

**type** Object of class "character": describes type of symmetry.

**SymmCenter** Object of class "ANY": center of symmetry.

**Methods**

**type** signature(object = "Symmetry"): accessor function for slot type

**SymmCenter** signature(object = "Symmetry"): accessor function for slot SymmCenter

**show** signature(object = "Symmetry")

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[DistributionSymmetry-class](#), [OptionalNumeric-class](#)

---

Td-class	Class "Td"
----------	------------

---

**Description**

The  $t$  distribution with  $df = \nu$  degrees of freedom has density

$$f(x) = \frac{\Gamma((\nu+1)/2)}{\sqrt{\pi\nu}\Gamma(\nu/2)} (1 + x^2/\nu)^{-(\nu+1)/2}$$

for all real  $x$ . It has mean 0 (for  $\nu > 1$ ) and variance  $\frac{\nu}{\nu-2}$  (for  $\nu > 2$ ). C.f. [rt](#)

**Objects from the Class**

Objects can be created by calls of the form `Td(df)`. This object is a  $t$  distribution.

**Slots**

`img` Object of class "Reals": The domain of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "TParameter": the parameter of this distribution ( $df$ ), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `rt`)

`d` Object of class "function": density function (calls function `dt`)

`p` Object of class "function": cumulative function (calls function `pt`)

`q` Object of class "function": inverse of the cumulative function (calls function `qt`)

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

**Methods**

**initialize** signature(.Object = "Td"): initialize method  
**df** signature(object = "Td"): returns the slot df of the parameter of the distribution  
**df<-** signature(object = "Td"): modifies the slot df of the parameter of the distribution  
**ncp** signature(object = "Td"): returns the slot ncp of the parameter of the distribution  
**ncp<-** signature(object = "Td"): modifies the slot ncp of the parameter of the distribution

**Ad hoc methods**

For R Version <2.3.0 ad hoc methods are provided for slots q, r if ncp!=0; for R Version >=2.3.0 the methods from package **stats** are used.

**Note**

The general *non-central t* with parameters  $(\nu, \delta) = (df, ncp)$  is defined as a the distribution of  $T_\nu(\delta) := \frac{U + \delta}{\chi_\nu / \sqrt{\nu}}$  where  $U$  and  $\chi_\nu$  are independent random variables,  $U \sim \mathcal{N}(0, 1)$ , and

$\chi_\nu^2$  is chi-squared, see [rchisq](#).

The most used applications are power calculations for *t*-tests:

Let  $T = \frac{\bar{X} - \mu_0}{S/\sqrt{n}}$  where  $\bar{X}$  is the [mean](#) and  $S$  the sample standard deviation ([sd](#)) of  $X_1, X_2, \dots, X_n$  which are i.i.d.

$N(\mu, \sigma^2)$ . Then  $T$  is distributed as non-centrally *t* with  $df = n - 1$  degrees of freedom and **non-centrality** parameter  $ncp = (\mu - \mu_0)\sqrt{n}/\sigma$ .

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[TParameter-class](#), [AbscontDistribution-class](#), [Reals-class](#), [rt](#)

**Examples**

```
T <- Td(df = 1) # T is a t distribution with df = 1.
r(T)(1) # one random number generated from this distribution, e.g. -0.09697573
d(T)(1) # Density of this distribution is 0.1591549 for x = 1.
p(T)(1) # Probability that x < 1 is 0.75.
q(T)(.1) # Probability that x < -3.077684 is 0.1.
```

```
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
df(T) # df of this distribution is 1.
df(T) <- 2 # df of this distribution is now 2.
Tn <- Td(df = 1, ncp = 5)
# T is a noncentral t distribution with df = 1 and ncp = 5.
d(Tn)(1) ## from R 2.3.0 on ncp no longer ignored...
```

---

TParameter-class

Class "TParameter"

---

### Description

The parameter of a t distribution, used by Td-class

### Objects from the Class

Objects can be created by calls of the form `new("TParameter", df, ncp)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Td` is instantiated.

### Slots

`df` Object of class "numeric": the degrees of freedom of a T distribution  
`ncp` Object of class "numeric": the noncentrality parameter of a T distribution  
`name` Object of class "character": a name / comment for the parameters

### Extends

Class "Parameter", directly.

### Methods

**initialize** signature(.Object = "TParameter"): initialize method  
**df** signature(object = "TParameter"): returns the slot df of the parameter of the distribution  
**df<-** signature(object = "TParameter"): modifies the slot df of the parameter of the distribution  
**ncp** signature(object = "TParameter"): returns the slot ncp of the parameter of the distribution  
**ncp<-** signature(object = "TParameter"): modifies the slot ncp of the parameter of the distribution

### Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**[Td-class Parameter-class](#)**Examples**

```
W <- new("TParameter",df=1, ncp = 0)
df(W) # df of this distribution is 1.
df(W) <- 2 # df of this distribution is now 2.
```

Truncate-methods

*Methods for function Truncate in Package ‘distr’***Description**

Truncate-methods

**Usage**

```
Truncate(object, ...)
## S4 method for signature 'AbscontDistribution'
Truncate(object, lower = -Inf, upper = Inf)
## S4 method for signature 'DiscreteDistribution'
Truncate(object, lower= -Inf, upper = Inf)
## S4 method for signature 'LatticeDistribution'
Truncate(object, lower= -Inf, upper = Inf)
## S4 method for signature 'UnivarLebDecDistribution'
Truncate(object, lower = -Inf, upper = Inf,
          withSimplify = getdistrOption("simplifyD"))
```

**Arguments**

object	distribution object
...	not yet used; takes up lower, upper, withSimplify.
lower	numeric; lower truncation point
upper	numeric; upper truncation point
withSimplify	logical; is result to be piped through a call to <a href="#">simplifyD?</a>

**Value**

the corresponding distribution of the truncated random variable

## Methods

- Truncate** signature(object = "AbscontDistribution"): returns the distribution of  $\min(\text{upper}, \max(X, \text{lower}))$  conditioned to  $\text{lower} \leq X \leq \text{upper}$ , if  $X$  is distributed according to object; if slot `.logExact` of argument object is TRUE and if either there is only one-sided truncation or both truncation points lie on the same side of the median, we use this representation to enhance the range of applicability, in particular, for slot `r`, we profit from Peter Dalgaard's clever log-tricks as indicated in <http://r.789695.n4.nabble.com/help-on-sampling-from-the-truncated-normal-gamma-distribution.html#a868120>. To this end we use the internal functions (i.e.; non exported to namespace) `.trunc.up` and `.trunc.low` which provide functional slots `r`, `d`, `p`, `q` for one-sided truncation. In case of two sided truncation, we simply use one-sided truncation successively — first left and then right in case we are right of the median, and the other way round else; the result is again of class "AbscontDistribution";
- Truncate** signature(object = "DiscreteDistribution"): returns the distribution of  $\min(\text{upper}, \max(X, \text{lower}))$  conditioned to  $\text{lower} \leq X \leq \text{upper}$ , if  $X$  is distributed according to object; the result is again of class "DiscreteDistribution"
- Truncate** signature(object = "LatticeDistribution"): if length of the corresp. lattice is infinite and slot `.logExact` of argument object is TRUE, we proceed similarly as in case of AbscontDistribution, also using internal functions `.trunc.up` and `.trunc.low`; else we use the corresponding "DiscreteDistribution" method; the result is again of class "LatticeDistribution"
- Truncate** signature(object = "UnivarLebDecDistribution"): returns the distribution of  $\min(\text{upper}, \max(X, \text{lower}))$  conditioned to  $\text{lower} \leq X \leq \text{upper}$ , if  $X$  is distributed according to object; the result is again of class "UnivarLebDecDistribution"

## See Also

[Huberize](#), [Minimum](#)

## Examples

```
plot(Truncate(Norm(), lower=-1, upper=2))
TN <- Truncate(Norm(), lower=15, upper=15.7) ### remarkably right!
plot(TN)
r(TN)(30)
TNG <- Truncate(Geom(prob=0.05), lower=325, upper=329) ### remarkably right!
plot(TNG)
```

---

Unif-class

Class "Unif"

---

## Description

The uniform distribution has density

$$d(x) = \frac{1}{\max - \min}$$

for  $\min$ , by default = 0,  $\leq x \leq \max$ , by default = 1. C.f. [runif](#)



## Objects from the Class

Objects can be created by calls of the form `Unif(Min,Max)`. This object is a uniform distribution.

## Slots

`img` Object of class "Reals": The space of the image of this distribution has got dimension 1 and the name "Real Space".

`param` Object of class "UnifParameter": the parameter of this distribution (Min and Max), declared at its instantiation

`r` Object of class "function": generates random numbers (calls function `runif`)

`d` Object of class "function": density function (calls function `dunif`)

`p` Object of class "function": cumulative function (calls function `punif`)

`q` Object of class "function": inverse of the cumulative function (calls function `qunif`)

`.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 "AbscontDistribution", directly.

Class "UnivariateDistribution", by class "AbscontDistribution".

Class "Distribution", by class "AbscontDistribution".

## Is-Relations

By means of `setIs`, R "knows" that a distribution object `obj` of class "Unif" with Min 0 and Max 1 also is a Beta distribution with parameters `shape1 = 1, shape2 = 1, ncp = 0`.

## Methods

**initialize** signature(`.Object = "Unif"`): initialize method

**Min** signature(`object = "Unif"`): returns the slot Min of the parameter of the distribution

**Min<-** signature(`object = "Unif"`): modifies the slot Min of the parameter of the distribution

**Max** signature(`object = "Unif"`): returns the slot Max of the parameter of the distribution

**Max<-** signature(`object = "Unif"`): modifies the slot Max of the parameter of the distribution

**\*** signature(`e1 = "Unif", e2 = "numeric"`): multiplication of this uniform distribution by an object of class 'numeric'

**+** signature(`e1 = "Unif", e2 = "numeric"`): addition of this uniform distribution to an object of class 'numeric'

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[UnifParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [runif](#)

**Examples**

```
U <- Unif(Min=0,Max=2) # U is a uniform distribution with Min=0 and Max=2.
r(U)(1) # one random number generated from this distribution, e.g. 1.984357
d(U)(1) # Density of this distribution is 0.5 for x=1.
p(U)(1) # Probability that x<1 is 0.5.
q(U)(.1) # Probability that x<0.2 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
Min(U) # Min of this distribution is 0.
Min(U) <- 1 # Min of this distribution is now 1.
Min(U) # Min of this distribution is 1.
Min(U) <- 0
is(U/2,"Beta") # yes
V <- U/2; as(V,"Beta")
```

---

UnifParameter-class	Class "UnifParameter"
---------------------	-----------------------

---

**Description**

The parameter of a uniform distribution, used by Unif-class

**Objects from the Class**

Objects can be created by calls of the form `new("UnifParameter",Max,Min)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Unif` is instantiated.

**Slots**

`Max` Object of class "numeric": the Max of a uniform distribution  
`Min` Object of class "numeric": the Min of a uniform distribution  
`name` Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "UnifParameter"): initialize method

**Min** signature(object = "UnifParameter"): returns the slot Min of the parameter of the distribution

**Min<-** signature(object = "UnifParameter"): modifies the slot Min of the parameter of the distribution

**Max** signature(object = "UnifParameter"): returns the slot Max of the parameter of the distribution

**Max<-** signature(object = "UnifParameter"): modifies the slot Max of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Unif-class Parameter-class](#)

**Examples**

```
W <- new("UnifParameter",Min=0,Max=1)
Max(W) # Max of this distribution is 1.
Max(W) <- 2 # Max of this distribution is now 2.
```

---

UniNormParameter-class

*Class "UniNormParameter"*

---

**Description**

The parameter of a univariate normal distribution, used by Norm-class

**Objects from the Class**

Objects can be created by calls of the form `new("NormParameter",sd,mean)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class Norm is instantiated.

**Slots**

**sd** Object of class "numeric": the sd of a univariate normal distribution

**mean** Object of class "numeric": the mean of a univariate normal distribution

**name** Object of class "character": a name / comment for the parameters

**Extends**

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

**Methods**

**initialize** signature(.Object = "UniNormParameter"): initialize method

**mean** signature(object = "UniNormParameter"): returns the slot mean of the parameter of the distribution

**mean<-** signature(object = "UniNormParameter"): modifies the slot mean of the parameter of the distribution

**sd** signature(object = "UniNormParameter"): returns the slot sd of the parameter of the distribution

**sd<-** signature(object = "UniNormParameter"): modifies the slot sd of the parameter of the distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Norm-class](#) [NormParameter-class](#) [Parameter-class](#)

**Examples**

```
W <- new("UniNormParameter", mean = 0, sd = 1)
sd(W) # sd of this distribution is 1
sd(W) <- 2 # sd of this distribution is now 2
```

---

UnivarDistrList

---

*Generating function for UnivarDistrList-class*


---

**Description**

Generates an object of class "UnivarDistrList".

**Usage**

```
UnivarDistrList(..., Dlist)
```

**Arguments**

... Objects of class "UnivariateDistribution" (or subclasses)

Dlist an optional list or object of class "UnivarDistrList"; if not missing it is appended to argument ...; this way UnivarMixingDistribution may also be called with a list (or "UnivarDistrList"-object) as argument as suggested in an e-mail by Krunoslav Sever (thank you!)

**Value**

Object of class "UnivarDistrList"

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[DistrList-class](#), [UnivarDistrList-class](#), [UnivarDistrList](#)

**Examples**

```
(DL <- UnivarDistrList(Norm(), Exp(), Pois()))
plot(DL)
as(Norm(), "UnivarDistrList")

## The function is currently defined as
function(...){
  new("UnivarDistrList", list(...))
}
```

---

UnivarDistrList-class *List of univariate distributions*

---

**Description**

Create a list of univariate distributions

**Objects from the Class**

Objects can be created by calls of the form `new("UnivarDistrList", ...)`. More frequently they are created via the generating function [DistrList](#).

**Slots**

.Data Object of class "list". A list of univariate distributions.

**Extends**

Class "DistrList", directly.  
 Class "list", by class "DistrList".  
 Class "vector", by class "DistrList".

**Methods**

**coerce** signature(from = "UnivariateDistribution", to = "UnivarDistrList"): create a UnivarDistrList object from a univariate distribution

**Author(s)**

Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[UnivarDistrList](#), [DistrList-class](#), [UnivariateDistribution-class](#)

**Examples**

```
(DL <- new("UnivarDistrList", list(Norm(), Exp()))
plot(DL)
as(Norm(), "UnivarDistrList")
```

---

UnivariateDistribution-class

*Class "UnivariateDistribution"*

---

**Description**

The UnivariateDistribution-class is the mother-class of the classes AbscontDistribution and DiscreteDistribution.

**Objects from the Class**

Objects can be created by calls of the form new("UnivariateDistribution").

**Slots**

**img** Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

**param** Object of class "Parameter": the parameter of this distribution

**r** Object of class "function": generates random numbers

**d** Object of class "function": density function

**p** Object of class "function": cumulative distribution function

**q** Object of class "function": 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

**initialize** signature(.Object = "UnivariateDistribution"):  
 initialize method

**dim** signature(x = "UnivariateDistribution"):  
 returns the dimension of the support of the distribution

- signature(e1 = "UnivariateDistribution"):  
 application of '-' to this univariate distribution

\* signature(e1 = "UnivariateDistribution", e2 = "numeric"):  
 multiplication of this univariate distribution by an object of class 'numeric'

/ signature(e1 = "UnivariateDistribution", e2 = "numeric"):  
 division of this univariate distribution by an object of class 'numeric'

+ signature(e1 = "UnivariateDistribution", e2 = "numeric"):  
 addition of this univariate distribution to an object of class 'numeric'

- signature(e1 = "UnivariateDistribution", e2 = "numeric"):  
 subtraction of an object of class 'numeric' from this univariate distribution

\* signature(e1 = "numeric", e2 = "UnivariateDistribution"):  
 multiplication of this univariate distribution by an object of class 'numeric'

+ signature(e1 = "numeric", e2 = "UnivariateDistribution"):  
 addition of this univariate distribution to an object of class 'numeric'

- signature(e1 = "numeric", e2 = "UnivariateDistribution"):  
 subtraction of this univariate distribution from an object of class 'numeric'

+ signature(e1 = "UnivariateDistribution", e2 = "UnivariateDistribution"):  
 Convolution of two univariate distributions. The slots p, d and q are approximated by grids.

- signature(e1 = "UnivariateDistribution", e2 = "UnivariateDistribution"):  
 Convolution of two univariate distributions. The slots p, d and q are approximated by grids.

**simplifyr** signature(object = "UnivariateDistribution"):  
 simplifies the r-method of a distribution, see there for further information

**print** signature(object = "UnivariateDistribution"):  
 returns the class of the object and its parameters

**show** signature(object = "UnivariateDistribution"): as print

Author(s)

Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

See Also

[Parameter-class Distribution-class AbscontDistribution-class](#)  
[DiscreteDistribution-class Reals-class RtoDPQ](#) [simplifyr-methods](#)

---

UnivarLebDecDistribution
<i>Generating function for Class "UnivarLebDecDistribution"</i>

---

Description

Generates an object of class "UnivarLebDecDistribution".

Usage

```
UnivarLebDecDistribution(acPart, discretePart, acWeight, discreteWeight,  
                        r = NULL, e = NULL, n = NULL, y = NULL)
```

Arguments

acPart	Object of class "AbscontDistribution" (or subclasses); a.c. part of the distribution
discretePart	Object of class "AbscontDistribution" (or subclasses); discrete part of the distribution
acWeight	Object of class "numeric"; weight of the a.c. part of the distribution
discreteWeight	Object of class "numeric"; weight of the discrete part of the distribution
r	optional argument; if given, this is a random number generator as function <code>r &lt;-function(n){...}</code> to produce r.v.'s distributed according to the distribution; used in a call to <a href="#">RtoDPQ.LC</a> if acPart and discretePart are missing.
e	optional argument; if argument r is given, this is the number of r.v.'s drawn to fill the empty slots of this object; if missing filled with <code>getdistrOption("RtoDPQ.e")</code> .
n	optional argument; if argument r is given, this is the number gridpoints used in filling the empty p,d,q slots of this object; if missing filled with <code>getdistrOption("DefaultNrGridPoint")</code> .
y	a (numeric) vector or NULL



## Details

At least one of arguments `discretePart`, `acPart`, or `r` must be given; if the first two are missing, slots are filled by a call to `RtoDPQ.LC`. For this purpose argument `r` is used together with arguments `e` and `n`. If the latter are missing they are filled with `getdistrOption("RtoDPQ.e")` and `getdistrOption("DefaultNrGridPoints")`, respectively. For the a.c. part, similarly to [RtoDPQ](#) we have an optional parameter `y` for using N. Horbenko's quantile trick: i.e.; on an equally spaced grid `x.grid` on  $[0,1]$ , apply  $f(q(x)(x.grid))$ , write the result to `y` and use these values instead of simulated ones.

If argument `discretePart` is missing but `acPart` is not, `discreteWeight` is set to 0 and `discretePart` is set to `Dirac(0)`. If argument `acPart` is missing but `discretePart` is not, `acWeight` is set to 0 and `discretePart` is set to `Norm()`. If both arguments `acPart` and `discretePart` are given, at least one of arguments `discreteWeight` and `acWeight` must be given and lie in  $[0,1]$ , else an error is thrown. If only one argument `acWeight` or `discreteWeight` is given the other one is gotten as  $1-[ac/discrete]Weight$ . Else if both are given, they must sum up to 1. If a weight is smaller than `getdistrOption("TruncQuantile")`, it is set to 0.

## Value

Object of class "UnivarLebDecDistribution".

## Author(s)

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

## See Also

[UnivarLebDecDistribution-class](#), [simplifyD](#)

## Examples

```
mylist <- UnivarLebDecDistribution(discretePart=Binom(3,.3), acPart=Norm(2,2),
                                acWeight=11/20)
mylist
```

---

UnivarLebDecDistribution-class

*Class "UnivarLebDecDistribution"*

---

## Description

UnivarLebDecDistribution-class is a class to formalize a Lebesgue decomposed distribution with a discrete and an absolutely continuous part; it is a subclass to class `UnivarMixingDistribution`.

## Objects from the Class

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

**Slots**

**mixCoeff** Object of class "numeric": a vector of length 2 of probabilities for the respective a.c. and discrete part of the object

**mixDistr** Object of class "UnivarDistrList": a list of univariate distributions containing the a.c. and discrete components; must be of length 2; the first component must be of class "AbscontDistribution", the second of class "DiscreteDistribution".

**img** Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

**param** Object of class "Parameter": the parameter of this distribution, having only the slot name "Parameter of a discrete distribution"

**r** Object of class "function": generates random numbers

**d** fixed to NULL

**p** Object of class "function": cumulative distribution function

**q** Object of class "function": 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.

**support** numeric vector — the support slot of the discrete part

**gaps** (numeric) matrix or NULL; — the gaps slot of the absolutely continuous part

**Extends**

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

**Methods**

**show** signature(object = "UnivarLebDecDistribution")

**plot** signature(object = "UnivarLebDecDistribution")

**acPart** signature(object = "UnivarLebDecDistribution")

**acPart<-** signature(object = "UnivarLebDecDistribution")

**discretePart** signature(object = "UnivarLebDecDistribution")

**discretePart<-** signature(object = "UnivarLebDecDistribution")

**acWeight** signature(object = "UnivarLebDecDistribution")

**acWeight<-** signature(object = "UnivarLebDecDistribution")

**discreteWeight** signature(object = "UnivarLebDecDistribution")

**discreteWeight**<- signature(object = "UnivarLebDecDistribution")

**p.ac** signature(object = "UnivarLebDecDistribution") accessor to slot p of acPart(object), possibly weighted by acWeight(object); it has an extra argument CondOrAbs with default value "cond" which if it does not partially match (by [pmatch](#)) "abs", returns exactly slot p of acPart(object) else weighted by acWeight(object).

**d.ac** signature(object = "UnivarLebDecDistribution") accessor to slot d of the absolutely continuous part of the distribution, possibly weighted by acWeight(object); it has an extra argument CondOrAbs which acts as the one in p.ac.

**q.ac** signature(object = "UnivarLebDecDistribution") accessor to slot q of acPart(object).

**r.ac** signature(object = "UnivarLebDecDistribution") accessor to slot r of acPart(object).

**p.discrete** signature(object = "UnivarLebDecDistribution") accessor to slot p of discretePart(object), possibly weighted by discreteWeight(object); it has an extra argument CondOrAbs which acts as the one in p.ac.

**d.discrete** signature(object = "UnivarLebDecDistribution") accessor to slot d of discretePart(object), possibly weighted by discreteWeight(object); it has an extra argument CondOrAbs which acts as the one in p.ac.

**q.discrete** signature(object = "UnivarLebDecDistribution") accessor to slot q of discretePart(object).

**r.discrete** signature(object = "UnivarLebDecDistribution") accessor to slot r of discretePart(object).

**coerce** signature(from = "AffLinUnivarLebDecDistribution", to = "UnivarLebDecDistribution"): create a "UnivarLebDecDistribution" object from a "AffLinUnivarLebDecDistribution" object

**coerce** signature(from = "AbscontDistribution", to = "UnivarLebDecDistribution"): create a "UnivarLebDecDistribution" object from a "AbscontDistribution" object

**coerce** signature(from = "DiscreteDistribution", to = "UnivarLebDecDistribution"): create a "UnivarLebDecDistribution" object from a "DiscreteDistribution" object

**Math** signature(x = "UnivarLebDecDistribution"): application of a mathematical function, e.g. sin or tan to this discrete distribution

- abs: signature(x = "UnivarLebDecDistribution"): exact image distribution of abs(x).
- exp: signature(x = "UnivarLebDecDistribution"): exact image distribution of exp(x).
- sign: signature(x = "UnivarLebDecDistribution"): exact image distribution of sign(x).
- sign: signature(x = "AcDcLcDistribution"): exact image distribution of sign(x).
- sqrt: signature(x = "AcDcLcDistribution"): exact image distribution of sqrt(x).
- log: signature(x = "UnivarLebDecDistribution"): (with optional further argument base, defaulting to exp(1)) exact image distribution of log(x).
- log10: signature(x = "UnivarLebDecDistribution"): exact image distribution of log10(x).
- sqrt: signature(x = "UnivarLebDecDistribution"): exact image distribution of sqrt(x).
- sqrt: signature(x = "AcDcLcDistribution"): exact image distribution of sqrt(x).

- signature(e1 = "UnivarLebDecDistribution"): application of '-' to this distribution

\* signature(e1 = "UnivarLebDecDistribution", e2 = "numeric"): multiplication of this distribution by an object of class 'numeric'

/ signature(e1 = "UnivarLebDecDistribution", e2 = "numeric"): division of this distribution by an object of class 'numeric'

- + signature(e1 = "UnivarLebDecDistribution", e2 = "numeric"): addition of this distribution to an object of class 'numeric'
- signature(e1 = "UnivarLebDecDistribution", e2 = "numeric"): subtraction of an object of class 'numeric' from this distribution
- \* signature(e1 = "numeric", e2 = "UnivarLebDecDistribution"): multiplication of this distribution by an object of class 'numeric'
- + signature(e1 = "numeric", e2 = "UnivarLebDecDistribution"): addition of this distribution to an object of class 'numeric'
- signature(e1 = "numeric", e2 = "UnivarLebDecDistribution"): subtraction of this distribution from an object of class 'numeric'
- + signature(e1 = "UnivarLebDecDistribution", e2 = "UnivarLebDecDistribution"): Convolution of two Lebesgue decomposed distributions. Result is again of class "UnivarLebDecDistribution", but if option `getdistrOption("withSimplify")` is TRUE it is piped through a call to [simplifyD](#), hence may also be of class `AbscontDistribution` or `DiscreteDistribution`.
- signature(e1 = "UnivarLebDecDistribution", e2 = "UnivarLebDecDistribution"): Convolution of two Lebesgue decomposed distributions. The same applies as for the preceding item.

#### Internal subclass "AffLinUnivarLebDecDistribution"

To enhance accuracy of several functionals on distributions, mainly from package **distrEx**, there is an internally used (but exported) subclass "AffLinUnivarLebDecDistribution" which has extra slots `a`, `b` (both of class "numeric"), and `X0` (of class "UnivarLebDecDistribution"), to capture the fact that the object has the same distribution as  $a * X0 + b$ . This is the class of the return value of methods

- signature(e1 = "UnivarLebDecDistribution")
- \* signature(e1 = "UnivarLebDecDistribution", e2 = "numeric")
- / signature(e1 = "UnivarLebDecDistribution", e2 = "numeric")
- + signature(e1 = "UnivarLebDecDistribution", e2 = "numeric")
- signature(e1 = "UnivarLebDecDistribution", e2 = "numeric")
- \* signature(e1 = "numeric", e2 = "UnivarLebDecDistribution")
- + signature(e1 = "numeric", e2 = "UnivarLebDecDistribution")
- signature(e1 = "numeric", e2 = "UnivarLebDecDistribution")
- signature(e1 = "AffLinUnivarLebDecDistribution")
- \* signature(e1 = "AffLinUnivarLebDecDistribution", e2 = "numeric")
- / signature(e1 = "AffLinUnivarLebDecDistribution", e2 = "numeric")
- + signature(e1 = "AffLinUnivarLebDecDistribution", e2 = "numeric")
- signature(e1 = "AffLinUnivarLebDecDistribution", e2 = "numeric")
- \* signature(e1 = "numeric", e2 = "AffLinUnivarLebDecDistribution")
- + signature(e1 = "numeric", e2 = "AffLinUnivarLebDecDistribution")
- signature(e1 = "numeric", e2 = "AffLinUnivarLebDecDistribution")

There also is a class union of "AffLinAbscontDistribution", "AffLinDiscreteDistribution", "AffLinUnivarLebDecDistribution" and called "AffLinDistribution" which is used for functionals.

**Internal virtual superclass "AcDcLcDistribution"**

As many operations should be valid no matter whether the operands are of class "AbscontDistribution", "DiscreteDistribution", or "UnivarLebDecDistribution", there is a class union of these classes called "AcDcLcDistribution"; in particular methods for "\*", "/", "^" (see [operators-methods](#)) and methods [Minimum](#), [Maximum](#), [Truncate](#), and [Huberize](#), and [convpow](#) are defined for this class union.

**Author(s)**

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

**See Also**

[Parameter-class UnivarMixingDistribution-class DiscreteDistribution-class AbscontDistribution-class simplifyD flat.LCD](#)

**Examples**

```
wg <- flat.mix(UnivarMixingDistribution(Unif(0,1),Unif(4,5),
                                     withSimplify=FALSE))
myLC <- UnivarLebDecDistribution(discretePart=Binom(3,.3), acPart = wg,
                               discreteWeight=.2)
myLC
p(myLC)(0.3)
r(myLC)(30)
q(myLC)(0.9)
## in RStudio or Jupyter IRKernel, use q.l(.) instead of q(.)
acPart(myLC)
plot(myLC)
d.discrete(myLC)(2)
p.ac(myLC)(0)
acWeight(myLC)
plot(acPart(myLC))
plot(discretePart(myLC))
gaps(myLC)
support(myLC)
plot(as(Norm(),"UnivarLebDecDistribution"))
```

---

UnivarMixingDistribution

*Generating function for Class "UnivarMixingDistribution"*

---

**Description**

Generates an object of class "UnivarMixingDistribution".

**Usage**

```
UnivarMixingDistribution(..., Dlist, mixCoeff,
                        withSimplify = getdistrOption("simplifyD"))
```

**Arguments**

<code>...</code>	Objects of class "UnivariateDistribution" (or subclasses)
<code>Dlist</code>	an optional list or object of class "UnivarDistrList"; if not missing it is appended to argument <code>...</code> ; this way <code>UnivarMixingDistribution</code> may also be called with a list (or "UnivarDistrList"-object) as argument as suggested in an e-mail by Krunoslav Sever (thank you!)
<code>mixCoeff</code>	Objects of class "numeric" : a vector of probabilities for the mixing components (must be of same length as arguments in <code>...</code> ).
<code>withSimplify</code>	"logical": shall the return value be piped through a call to <code>simplifyD</code> ?

**Details**

If `mixCoeff` is missing, all elements in `...` are equally weighted.

**Value**

Object of class "UnivarMixingDistribution", or if argument `withSimplify` is TRUE and the resulting object would have one mixing component with probability (almost) 1, `UnivarMixingDistribution` will return this component.

**Author(s)**

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

**See Also**

[UnivarMixingDistribution-class](#), [simplifyD](#)

**Examples**

```
mylist <- UnivarMixingDistribution(Binom(3,.3), Dirac(2), Norm(),
  mixCoeff=c(1/4,1/5,11/20))
```

---

UnivarMixingDistribution-class

*Class "UnivarMixingDistribution"*

---

**Description**

`UnivarMixingDistribution-class` is a class to formalize univariate mixing distributions; it is a subclass to class `UnivariateDistribution`.

**Objects from the Class**

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

**Slots**

**mixCoeff** Object of class "numeric": a vector of probabilities for the mixing components.

**mixDistr** Object of class "UnivarDistrList": a list of univariate distributions containing the mixing components; must be of same length as **mixCoeff**.

**img** Object of class "Reals": the space of the image of this distribution which has dimension 1 and the name "Real Space"

**param** Object of class "Parameter": the parameter of this distribution, having only the slot name "Parameter of a discrete distribution"

**r** Object of class "function": generates random numbers

**d** fixed to NULL

**p** Object of class "function": cumulative distribution function

**q** Object of class "function": quantile function

**support** numeric vector — the union of all support slots of components, if existing

**gaps** (numeric) matrix or NULL; the merged gaps slots of all components, if existing (else 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

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

**Extends**

Class "UnivariateDistribution" class "Distribution" by class "UnivariateDistribution".

**Methods**

**show** signature(object = "UnivarMixingDistribution") prints the object

**mixCoeff<-** signature(object = "UnivarMixingDistribution") replaces the corresponding slot

**mixCoeff** signature(object = "UnivarMixingDistribution") returns the corresponding slot

**mixDistr<-** signature(object = "UnivarMixingDistribution") replaces the corresponding slot

**mixDistr** signature(object = "UnivarMixingDistribution") returns the corresponding slot

**support** signature(object = "UnivarMixingDistribution") returns the corresponding slot

**gaps** signature(object = "UnivarMixingDistribution") returns the corresponding slot

**.logExact** signature(object = "Distribution"): returns slot **.logExact** if existing; else tries to convert the object to a newer version of its class by [conv2NewVersion](#) and returns the corresponding slot of the converted object.

**.lowerExact** signature(object = "Distribution"): returns slot **.lowerExact** if existing; else tries to convert the object to a newer version of its class by [conv2NewVersion](#) and returns the corresponding slot of the converted object.

**Symmetry** returns slot **Symmetry** if existing; else tries to convert the object to a newer version of its class by [conv2NewVersion](#) and returns the corresponding slot of the converted object.

**Author(s)**

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

**See Also**

[Parameter-class](#), [UnivariateDistribution-class](#), [LatticeDistribution-class](#), [AbscontDistribution-class](#), [simplifyD](#), [flat.mix](#)

**Examples**

```
mylist <- UnivarMixingDistribution(Binom(3,.3), Dirac(2), Norm(),
  mixCoeff=c(1/4,1/5,11/20))
mylist2 <- UnivarMixingDistribution(Binom(3,.3), mylist,
  mixCoeff=c(.3,.7))
mylist2
p(mylist)(0.3)
mixDistr(mylist2)
```

---

Version Management

*Methods for Version Management in Package 'distr'*

---

**Description**

Version-Management-methods

**Usage**

```
isOldVersion(object)
conv2NewVersion(object)
## S4 method for signature 'ANY'
isOldVersion(object)
## S4 method for signature 'ANY'
conv2NewVersion(object)
## S4 method for signature 'LatticeDistribution'
conv2NewVersion(object)
```

**Arguments**

object                      object of class "ANY" (or subclasses)

**Details**

From version 1.9 of this package on, class "AbscontDistribution" has an extra slot gaps. As the addition of new slots will probably happen again in the future development of our packages, we provide the following two help functions `isOldVersion` and `conv2NewVersion` to check whether the object was generated by an older version of this package and to convert such an object to the new format, respectively. Also, the intermediate class "LatticeDistribution" is introduced at version 1.9 so that all subclasses of "DiscreteDistribution" like "Binom", "Nbinom" etc, now



have an extra slot `lattice`. `conv2NewVersion` takes this up and provides a particular method for signature `"LatticeDistribution"` which fills slot `lattice` accordingly.

**isOldVersion** `signature(object = "ANY")`: throws an error if `isClass(class(object))` is `FALSE`, i.e.; if the class of `object` is no formal (S4) class. Else it checks whether all slots of the actual class definition may be accessed and if so returns `FALSE` and else `TRUE` and issues a warning.

**conv2NewVersion** `signature(object = "ANY")`: Generates a valid copy of `object` (according to the actual class definition), using the slots of `object` where possible and for the slots which are not yet present in `object` (because it was generated by an older version of the class definition), it generates a prototype object of the class of `object` with `new(class(object))` and uses the slot values of this prototype to fill the missing slots.

**conv2NewVersion** `signature(object = "LatticeDistribution")`: Generates a valid copy of `object` (according to the actual class definition, i.e.; with a corresponding `lattice`-slot), by generating a new instance of this object by `new(class(object), <list-of-parameters>)`.

---

Weibull-class

Class "Weibull"

---

## Description

The Weibull distribution with shape parameter  $a$ , by default = 1, and scale parameter  $\sigma$  has density given by, by default = 1,

$$d(x) = (a/\sigma)(x/\sigma)^{a-1} \exp(-(x/\sigma)^a)$$

for  $x > 0$ .

C.f. [rweibull](#)

## Objects from the Class

Objects can be created by calls of the form `Weibull(shape,scale)`. This object is a Weibull distribution.

## Slots

**img** Object of class `"Reals"`: The space of the image of this distribution has got dimension 1 and the name `"Real Space"`.

**param** Object of class `"WeibullParameter"`: the parameter of this distribution (shape and scale), declared at its instantiation

**r** Object of class `"function"`: generates random numbers (calls function `rweibull`)

**d** Object of class `"function"`: density function (calls function `dweibull`)

**p** Object of class `"function"`: cumulative function (calls function `pweibull`)

**q** Object of class `"function"`: inverse of the cumulative function (calls function `qweibull`)

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

## Methods

**initialize** signature(.Object = "Weibull"): initialize method

**scale** signature(object = "Weibull"): returns the slot scale of the parameter of the distribution

**scale<-** signature(object = "Weibull"): modifies the slot scale of the parameter of the distribution

**shape** signature(object = "Weibull"): returns the slot shape of the parameter of the distribution

**shape<-** signature(object = "Weibull"): modifies the slot shape of the parameter of the distribution

\* signature(e1 = "Weibull", e2 = "numeric"): For the Weibull distribution we use its closedness under positive scaling transformations.

## Note

The density is  $d(x) = 0$  for  $x < 0$ .  
 The cumulative is  $p(x) = 1 - \exp(-(x/\sigma)^a)$ ,  
 the mean is  $E(X) = \sigma\Gamma(1 + 1/a)$ ,  
 and the  $Var(X) = \sigma^2(\Gamma(1 + 2/a) - (\Gamma(1 + 1/a))^2)$ .

## Author(s)

Thomas Stabla <statho3@web.de>,  
 Florian Camphausen <fcampi@gmx.de>,  
 Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
 Matthias Kohl <Matthias.Kohl@stamats.de>

## See Also

[WeibullParameter-class](#) [AbscontDistribution-class](#) [Reals-class](#) [rweibull](#)

**Examples**

```

W <- Weibull(shape=1,scale=1) # W is a Weibull distribution with shape=1 and scale=1.
r(W)(1) # one random number generated from this distribution, e.g. 0.5204105
d(W)(1) # Density of this distribution is 0.3678794 for x=1.
p(W)(1) # Probability that x<1 is 0.6321206.
q(W)(.1) # Probability that x<0.1053605 is 0.1.
## in RStudio or Jupyter IRKernel, use q.l(.)(.) instead of q(.)(.)
shape(W) # shape of this distribution is 1.
shape(W) <- 2 # shape of this distribution is now 2.

```

---

WeibullParameter-class

*Class "WeibullParameter"*


---

**Description**

The parameter of a Weibull distribution, used by Weibull-class

**Objects from the Class**

Objects can be created by calls of the form `new("WeibullParameter", shape, scale)`. Usually an object of this class is not needed on its own, it is generated automatically when an object of the class `Weibull` is instantiated.

**Slots**

**shape** Object of class "numeric": the shape of a Weibull distribution  
**scale** Object of class "numeric": the scale of a Weibull distribution  
**name** Object of class "character": a name / comment for the parameters

**Extends**

Class "Parameter", directly.

**Methods**

**initialize** signature(.Object = "WeibullParameter"): initialize method  
**scale** signature(object = "WeibullParameter"): returns the slot scale of a parameter of a Weibull distribution  
**scale<-** signature(object = "WeibullParameter"): modifies the slot scale of a parameter of a Weibull distribution  
**shape** signature(object = "WeibullParameter"): returns the slot shape of a parameter of a Weibull distribution  
**shape<-** signature(object = "WeibullParameter"): modifies the slot shape of a parameter of a Weibull distribution

**Author(s)**

Thomas Stabla <statho3@web.de>,  
Florian Camphausen <fcampi@gmx.de>,  
Peter Ruckdeschel <peter.ruckdeschel@uni-oldenburg.de>,  
Matthias Kohl <Matthias.Kohl@stamats.de>

**See Also**

[Weibull-class Parameter-class](#)

**Examples**

```
W <- new("WeibullParameter",shape=1,scale=1)
shape(W) # shape of this distribution is 1.
shape(W) <- 2 # shape of this distribution is now 2.
```

---

width-methods

*Methods for Function width in Package ‘distr’*

---

**Description**

width-methods

**Methods**

**width** signature(object = "Lattice"): returns the slot width of the lattice  
**width<-** signature(object = "Lattice"): modifies the slot width of the lattice  
**width** signature(object = "LatticeDistribution"): returns the slot width of the lattice slot of the distribution  
**width<-** signature(object = "LatticeDistribution"): modifies the slot width of the lattice slot of the distribution

# Index

- \* **Arcsine distribution**
  - Arcsine-class, 18
- \* **Beta distribution**
  - Beta-class, 19
  - BetaParameter-class, 21
  - shape1-methods, 155
  - shape2-methods, 156
- \* **Binomial distribution**
  - Binom-class, 22
  - BinomParameter-class, 24
  - prob-methods, 138
  - size-methods, 158
- \* **Binomial family**
  - Binom-class, 22
- \* **CDF**
  - p-methods, 125
  - p.l-methods, 125
- \* **Cauchy distribution**
  - Cauchy-class, 25
  - CauchyParameter-class, 27
  - location-methods, 98
  - scale-methods, 153
- \* **Chi Square distribution**
  - scale-methods, 153
- \* **Chi square distribution**
  - Chisq-class, 28
  - ChisqParameter-class, 30
  - df-methods, 39
  - ncp-methods, 113
  - shape-methods, 155
- \* **DFT**
  - convpow-methods, 34
- \* **Dirac distribution**
  - Dirac-class, 41
  - DiracParameter-class, 42
  - location-methods, 98
- \* **Double Exponential distribution**
  - rate-methods, 146
- \* **Double exponential distribution**
  - DExp-class, 37
- \* **Exp distribution**
  - ExpParameter-class, 65
  - shape-methods, 155
- \* **Exponential distribution**
  - Exp-class, 64
  - rate-methods, 146
- \* **F distribution**
  - df1-methods, 39
  - df2-methods, 40
  - Fd-class, 67
  - FParameter-class, 71
  - ncp-methods, 113
- \* **FFT**
  - convpow-methods, 34
  - operators-methods, 118
- \* **Gamma distribution**
  - Gammad-class, 72
  - GammaParameter-class, 74
  - scale-methods, 153
  - shape-methods, 155
- \* **Gaussian distribution**
  - mean-methods, 105
  - Norm-class, 114
  - NormParameter-class, 116
  - sd-methods, 154
- \* **Geometric distribution**
  - Geom-class, 76
  - prob-methods, 138
  - size-methods, 158
- \* **Hypergeometric distribution**
  - Hyper-class, 81
  - HyperParameter-class, 82
  - k-methods, 85
  - m-methods, 102
  - n-methods, 108
- \* **Laplace distribution**
  - DExp-class, 37
  - rate-methods, 146

- \* **Lebesgue decomposed distribution**
  - flat.LCD, 69
  - UnivarLebDecDistribution-class, 177
- \* **Log-Normal distribution**
  - Lnorm-class, 95
  - LnormParameter-class, 97
  - meanlog-methods, 105
  - sdlog-methods, 154
- \* **Logistic distribution**
  - Logis-class, 99
  - LogisParameter-class, 101
  - scale-methods, 153
- \* **Negative Binomial distribution**
  - Geom-class, 76
  - Nbinom-class, 110
  - NbinomParameter-class, 112
  - prob-methods, 138
  - size-methods, 158
- \* **Normal distribution**
  - mean-methods, 105
  - Norm-class, 114
  - NormParameter-class, 116
  - sd-methods, 154
- \* **Poisson distribution**
  - lambda-methods, 85
  - Pois-class, 133
  - PoisParameter-class, 135
- \* **RNG**
  - r-methods, 146
- \* **S4 classes**
  - distr-package, 5
- \* **S4 distribution class**
  - Norm-class, 114
- \* **S4 distribution class**
  - AbscontDistribution-class, 14
  - Arcsine-class, 18
  - Beta-class, 19
  - Binom-class, 22
  - Cauchy-class, 25
  - Chisq-class, 28
  - CompoundDistribution, 31
  - CompoundDistribution-class, 32
  - DExp-class, 37
  - DiscreteDistribution-class, 46
  - distr-package, 5
  - Distribution-class, 51
  - DistrList, 53
  - DistrList-class, 54
  - Exp-class, 64
  - Fd-class, 67
  - Gammad-class, 72
  - LatticeDistribution, 87
  - LatticeDistribution-class, 89
  - Lnorm-class, 95
  - Logis-class, 99
  - makeAbscontDistribution, 102
  - support-methods, 163
  - Td-class, 164
  - Unif-class, 168
  - UnivarDistrList, 172
  - UnivarDistrList-class, 173
  - UnivariateDistribution-class, 174
  - UnivarLebDecDistribution, 176
  - UnivarLebDecDistribution-class, 177
  - UnivarMixingDistribution, 181
  - UnivarMixingDistribution-class, 182
  - Weibull-class, 185
- \* **S4 parameter class**
  - BetaParameter-class, 21
  - BinomParameter-class, 24
  - CauchyParameter-class, 27
  - ChisqParameter-class, 30
  - Dirac-class, 41
  - DiracParameter-class, 42
  - distr-package, 5
  - ExpParameter-class, 65
  - FParameter-class, 71
  - GammaParameter-class, 74
  - Geom-class, 76
  - Hyper-class, 81
  - HyperParameter-class, 82
  - k-methods, 85
  - lambda-methods, 85
  - LnormParameter-class, 97
  - location-methods, 98
  - LogisParameter-class, 101
  - m-methods, 102
  - Max-methods, 104
  - mean-methods, 105
  - meanlog-methods, 105
  - Min-methods, 106
  - n-methods, 108
  - name-methods, 108

- Nbinom-class, 110
- NbinomParameter-class, 112
- ncp-methods, 113
- NormParameter-class, 116
- OptionalParameter-class, 123
- param-methods, 126
- Parameter-class, 126
- Pois-class, 133
- PoisParameter-class, 135
- prob-methods, 138
- rate-methods, 146
- scale-methods, 153
- sd-methods, 154
- sdlog-methods, 154
- size-methods, 158
- TParameter-class, 166
- UnifParameter-class, 170
- UniNormParameter-class, 171
- WeibullParameter-class, 187
- \* **S4 space class**
  - dimension-methods, 40
  - distr-package, 5
  - EuclideanSpace-class, 62
  - Lattice-class, 86
  - Length-methods, 92
  - liesIn-methods, 93
  - liesInSupport, 93
  - name-methods, 108
  - Naturals-class, 109
  - pivot-methods, 127
  - Reals-class, 147
  - rSpace-class, 148
  - width-methods, 188
- \* **T distribution**
  - df-methods, 39
  - ncp-methods, 113
  - Td-class, 164
  - TParameter-class, 166
- \* **T(1) distribution**
  - Cauchy-class, 25
- \* **Uniform distribution**
  - Max-methods, 104
  - Min-methods, 106
  - Unif-class, 168
  - UnifParameter-class, 170
- \* **Univariate normal / Gaussian distribution**
  - UniNormParameter-class, 171
- \* **Weibull distribution**
  - scale-methods, 153
  - shape-methods, 155
  - Weibull-class, 185
  - WeibullParameter-class, 187
- \* **absolutely continuous distribution**
  - AbscontDistribution, 12
  - AbscontDistribution-class, 14
  - Arcsine-class, 18
  - Beta-class, 19
  - Cauchy-class, 25
  - Chisq-class, 28
  - DExp-class, 37
  - distr-package, 5
  - Distribution-class, 51
  - Exp-class, 64
  - Fd-class, 67
  - flat.mix, 70
  - Gammad-class, 72
  - gaps-methods, 75
  - Lnorm-class, 95
  - Logis-class, 99
  - Norm-class, 114
  - RtoDPQ, 148
  - RtoDPQ.LC, 151
  - Td-class, 164
  - Unif-class, 168
  - UnivarLebDecDistribution-class, 177
  - Weibull-class, 185
- \* **accessor function**
  - d-methods, 36
  - df-methods, 39
  - df1-methods, 39
  - df2-methods, 40
  - dim-methods, 40
  - dimension-methods, 40
  - gaps-methods, 75
  - img-methods, 84
  - k-methods, 85
  - lambda-methods, 85
  - Length-methods, 92
  - location-methods, 98
  - m-methods, 102
  - Max-methods, 104
  - mean-methods, 105
  - meanlog-methods, 105
  - Min-methods, 106
  - n-methods, 108

- name-methods, 108
- ncp-methods, 113
- p-methods, 125
- param-methods, 126
- pivot-methods, 127
- prob-methods, 138
- q-methods, 139
- r-methods, 146
- rate-methods, 146
- scale-methods, 153
- sd-methods, 154
- sdlog-methods, 154
- shape-methods, 155
- shape1-methods, 155
- shape2-methods, 156
- size-methods, 158
- support-methods, 163
- width-methods, 188
- \* **affine linear lattice**
  - Length-methods, 92
  - pivot-methods, 127
  - width-methods, 188
- \* **affine linear**
  - operators-methods, 118
- \* **algebra**
  - solve-methods, 159
  - sqr-methods, 162
- \* **anamorphosis**
  - r-methods, 146
- \* **arithmetics for distributions**
  - convpow-methods, 34
  - distr-package, 5
  - distrARITH, 50
- \* **arith**
  - distrARITH, 50
  - flat.LCD, 69
  - flat.mix, 70
  - Math-methods, 103
  - operators-methods, 118
  - RtoDPQ, 148
  - RtoDPQ.d, 150
  - RtoDPQ.LC, 151
  - simplifyr-methods, 158
- \* **array**
  - PosDefSymmMatrix, 136
  - PosDefSymmMatrix-class, 137
  - solve-methods, 159
  - sqr-methods, 162
- \* **beta distribution**
  - ncp-methods, 113
- \* **class union**
  - OptionalParameter-class, 123
- \* **classes**
  - DistributionSymmetry-class, 52
  - DistrSymmList-class, 59
  - EllipticalSymmetry-class, 60
  - NoSymmetry-class, 118
  - PosDefSymmMatrix-class, 137
  - SphericalSymmetry-class, 161
  - Symmetry-class, 163
- \* **compound distribution**
  - CompoundDistribution, 31
  - CompoundDistribution-class, 32
- \* **conditional distribution**
  - Distribution-class, 51
- \* **convolution for distributions**
  - convpow-methods, 34
- \* **convolution**
  - operators-methods, 118
- \* **cumulative distribution function**
  - p-methods, 125
  - p.l-methods, 125
- \* **degree of freedom**
  - df-methods, 39
  - df1-methods, 39
  - df2-methods, 40
- \* **density**
  - d-methods, 36
- \* **dimension**
  - dim-methods, 40
  - dimension-methods, 40
- \* **discrete distribution**
  - Binom-class, 22
  - Dirac-class, 41
  - DiscreteDistribution, 44
  - DiscreteDistribution-class, 46
  - distr-package, 5
  - Distribution-class, 51
  - Geom-class, 76
  - Hyper-class, 81
  - LatticeDistribution-class, 89
  - Nbinom-class, 110
  - Pois-class, 133
  - RtoDPQ.d, 150
  - support-methods, 163
  - UnivarLebDecDistribution-class,



- 177
- UnivarMixingDistribution-class, 182
- \* distribution**
  - AbscontDistribution, 12
  - AbscontDistribution-class, 14
  - Arcsine-class, 18
  - Beta-class, 19
  - BetaParameter-class, 21
  - Binom-class, 22
  - BinomParameter-class, 24
  - Cauchy-class, 25
  - CauchyParameter-class, 27
  - Chisq-class, 28
  - ChisqParameter-class, 30
  - CompoundDistribution, 31
  - CompoundDistribution-class, 32
  - convpow-methods, 34
  - d-methods, 36
  - decomposePM-methods, 36
  - DExp-class, 37
  - df-methods, 39
  - df1-methods, 39
  - df2-methods, 40
  - Dirac-class, 41
  - DiracParameter-class, 42
  - DiscreteDistribution, 44
  - DiscreteDistribution-class, 46
  - distr-package, 5
  - distrARITH, 50
  - Distribution-class, 51
  - DistributionSymmetry-class, 52
  - DistrList, 53
  - DistrList-class, 54
  - distrMASK, 55
  - distroptions, 56
  - DistrSymmList, 58
  - DistrSymmList-class, 59
  - EllipticalSymmetry, 59
  - EllipticalSymmetry-class, 60
  - EmpiricalDistribution, 61
  - EuclideanSpace-class, 62
  - Exp-class, 64
  - ExpParameter-class, 65
  - Fd-class, 67
  - flat.LCD, 69
  - flat.mix, 70
  - FParameter-class, 71
  - Gammad-class, 72
  - GammaParameter-class, 74
  - gaps-methods, 75
  - Geom-class, 76
  - getLabel, 78
  - getLow, getUp, 79
  - Huberize-methods, 80
  - Hyper-class, 81
  - HyperParameter-class, 82
  - img-methods, 84
  - k-methods, 85
  - lambda-methods, 85
  - Lattice-class, 86
  - LatticeDistribution, 87
  - LatticeDistribution-class, 89
  - Length-methods, 92
  - liesIn-methods, 93
  - liesInSupport, 93
  - Lnorm-class, 95
  - LnormParameter-class, 97
  - location-methods, 98
  - Logis-class, 99
  - LogisParameter-class, 101
  - m-methods, 102
  - makeAbscontDistribution, 102
  - Math-methods, 103
  - Max-methods, 104
  - mean-methods, 105
  - meanlog-methods, 105
  - Min-methods, 106
  - Minimum-methods, 106
  - n-methods, 108
  - name-methods, 108
  - Naturals-class, 109
  - Nbinom-class, 110
  - NbinomParameter-class, 112
  - ncp-methods, 113
  - Norm-class, 114
  - NormParameter-class, 116
  - NoSymmetry, 117
  - NoSymmetry-class, 118
  - operators-methods, 118
  - OptionalParameter-class, 123
  - p-methods, 125
  - p.l-methods, 125
  - param-methods, 126
  - Parameter-class, 126
  - pivot-methods, 127

- plot-methods, 127
- Pois-class, 133
- PoisParameter-class, 135
- print-methods, 138
- prob-methods, 138
- q-methods, 139
- q.r-methods, 140
- qqbounds, 140
- qqplot, 142
- r-methods, 146
- rate-methods, 146
- Reals-class, 147
- rSpace-class, 148
- RtoDPQ, 148
- RtoDPQ.d, 150
- RtoDPQ.LC, 151
- scale-methods, 153
- sd-methods, 154
- sdlog-methods, 154
- shape-methods, 155
- shape1-methods, 155
- shape2-methods, 156
- simplifyD-methods, 156
- simplifyr-methods, 158
- size-methods, 158
- SphericalSymmetry, 160
- SphericalSymmetry-class, 161
- support-methods, 163
- Td-class, 164
- TParameter-class, 166
- Truncate-methods, 167
- Unif-class, 168
- UnifParameter-class, 170
- UniNormParameter-class, 171
- UnivarDistrList, 172
- UnivarDistrList-class, 173
- UnivariateDistribution-class, 174
- UnivarLebDecDistribution, 176
- UnivarLebDecDistribution-class, 177
- UnivarMixingDistribution, 181
- UnivarMixingDistribution-class, 182
- Weibull-class, 185
- WeibullParameter-class, 187
- width-methods, 188
- \* **documentation**
  - distrARITH, 50
  - distrMASK, 55
- \* **dplot**
  - options, 124
- \* **eigenvalue decomposition**
  - solve-methods, 159
  - sqrt-methods, 162
- \* **elliptical symmetry**
  - EllipticalSymmetry, 59
  - EllipticalSymmetry-class, 60
- \* **empirical distribution**
  - EmpiricalDistribution, 61
- \* **environment**
  - options, 124
- \* **generating function**
  - AbscontDistribution, 12
  - Arcsine-class, 18
  - Beta-class, 19
  - Binom-class, 22
  - Cauchy-class, 25
  - Chisq-class, 28
  - CompoundDistribution, 31
  - CompoundDistribution-class, 32
  - DExp-class, 37
  - DiscreteDistribution, 44
  - DistrList, 53
  - EmpiricalDistribution, 61
  - Exp-class, 64
  - Fd-class, 67
  - Gammad-class, 72
  - Geom-class, 76
  - Hyper-class, 81
  - LatticeDistribution, 87
  - LatticeDistribution-class, 89
  - Nbinom-class, 110
  - Pois-class, 133
  - Td-class, 164
  - Unif-class, 168
  - UnivarLebDecDistribution, 176
  - UnivarMixingDistribution, 181
  - UnivarMixingDistribution-class, 182
  - Weibull-class, 185
- \* **global options**
  - distroptions, 56
  - options, 124
- \* **hplot**
  - plot-methods, 127
  - qqbounds, 140

- qqplot, 142
- \* **image distribution**
  - flat.mix, 70
  - Math-methods, 103
  - operators-methods, 118
  - RtoDPQ, 148
  - RtoDPQ.d, 150
  - RtoDPQ.LC, 151
  - simplifyr-methods, 158
- \* **info file**
  - convpow-methods, 34
  - distrARITH, 50
  - distrMASK, 55
- \* **inverse**
  - solve-methods, 159
  - sqr-methods, 162
- \* **ipplot**
  - options, 124
- \* **lattice distribution**
  - Binom-class, 22
  - CompoundDistribution-class, 32
  - Dirac-class, 41
  - DiscreteDistribution-class, 46
  - distr-package, 5
  - Distribution-class, 51
  - Geom-class, 76
  - Hyper-class, 81
  - LatticeDistribution, 87
  - LatticeDistribution-class, 89
  - Nbinom-class, 110
  - Pois-class, 133
  - UnivarMixingDistribution-class, 182
- \* **lattice of a distribution**
  - LatticeDistribution, 87
  - LatticeDistribution-class, 89
- \* **lattice**
  - Length-methods, 92
  - pivot-methods, 127
  - width-methods, 188
- \* **left-continuous**
  - p.l-methods, 125
  - q-methods, 139
  - q.r-methods, 140
- \* **list**
  - CompoundDistribution, 31
  - DistrList, 53
  - DistrList-class, 54
  - UnivarDistrList, 172
  - UnivarDistrList-class, 173
  - UnivarLebDecDistribution, 176
  - UnivarMixingDistribution, 181
- \* **location family**
  - Dirac-class, 41
- \* **location parameter**
  - DiracParameter-class, 42
  - location-methods, 98
  - mean-methods, 105
- \* **location scale family**
  - CauchyParameter-class, 27
  - location-methods, 98
  - Logis-class, 99
  - LogisParameter-class, 101
  - mean-methods, 105
  - Norm-class, 114
  - NormParameter-class, 116
  - scale-methods, 153
  - sd-methods, 154
- \* **masking**
  - distrMASK, 55
- \* **math**
  - distrARITH, 50
  - flat.LCD, 69
  - igamma, 84
  - operators-methods, 118
  - RtoDPQ, 148
  - RtoDPQ.d, 150
  - RtoDPQ.LC, 151
  - simplifyr-methods, 158
- \* **maximum**
  - Minimum-methods, 106
- \* **memoryless distribution**
  - Exp-class, 64
- \* **methods**
  - d-methods, 36
  - decomposePM-methods, 36
  - df-methods, 39
  - df1-methods, 39
  - df2-methods, 40
  - dim-methods, 40
  - dimension-methods, 40
  - gaps-methods, 75
  - getLow, getUp, 79
  - Huberize-methods, 80
  - img-methods, 84
  - k-methods, 85

- lambda-methods, 85
- Length-methods, 92
- liesIn-methods, 93
- liesInSupport, 93
- location-methods, 98
- m-methods, 102
- Max-methods, 104
- mean-methods, 105
- meanlog-methods, 105
- Min-methods, 106
- Minimum-methods, 106
- n-methods, 108
- name-methods, 108
- ncp-methods, 113
- OptionalParameter-class, 123
- p-methods, 125
- p.l-methods, 125
- param-methods, 126
- Parameter-class, 126
- pivot-methods, 127
- plot-methods, 127
- print-methods, 138
- q-methods, 139
- q.r-methods, 140
- r-methods, 146
- rate-methods, 146
- scale-methods, 153
- sd-methods, 154
- sdlog-methods, 154
- shape-methods, 155
- shape1-methods, 155
- shape2-methods, 156
- simplifyD-methods, 156
- support-methods, 163
- Truncate-methods, 167
- width-methods, 188
- \* **minimum**
  - Minimum-methods, 106
- \* **misc**
  - distr-defunct, 49
- \* **multivariate distribution**
  - Distribution-class, 51
  - DistrList, 53
  - DistrList-class, 54
  - UnivarDistrList-class, 173
  - UnivarLebDecDistribution, 176
  - UnivarMixingDistribution, 181
- \* **naturals**
  - Naturals-class, 109
- \* **non centrality parameter**
  - ncp-methods, 113
- \* **options**
  - distroptions, 56
  - options, 124
- \* **package**
  - distr-package, 5
- \* **parameter**
  - BetaParameter-class, 21
  - BinomParameter-class, 24
  - CauchyParameter-class, 27
  - ChisqParameter-class, 30
  - df-methods, 39
  - df1-methods, 39
  - df2-methods, 40
  - DiracParameter-class, 42
  - ExpParameter-class, 65
  - FParameter-class, 71
  - GammaParameter-class, 74
  - HyperParameter-class, 82
  - k-methods, 85
  - lambda-methods, 85
  - LnormParameter-class, 97
  - location-methods, 98
  - LogisParameter-class, 101
  - m-methods, 102
  - Max-methods, 104
  - mean-methods, 105
  - meanlog-methods, 105
  - Min-methods, 106
  - n-methods, 108
  - NbinomParameter-class, 112
  - ncp-methods, 113
  - NormParameter-class, 116
  - OptionalParameter-class, 123
  - param-methods, 126
  - Parameter-class, 126
  - PoisParameter-class, 135
  - prob-methods, 138
  - rate-methods, 146
  - scale-methods, 153
  - sd-methods, 154
  - sdlog-methods, 154
  - shape-methods, 155
  - shape1-methods, 155
  - shape2-methods, 156
  - size-methods, 158

- TParameter-class, 166
  - UnifParameter-class, 170
  - UniNormParameter-class, 171
  - WeibullParameter-class, 187
- \* **positive-definite matrix**
  - PosDefSymmMatrix, 136
  - PosDefSymmMatrix-class, 137
- \* **positive-semi-definite matrix**
  - PosDefSymmMatrix, 136
  - PosDefSymmMatrix-class, 137
- \* **print**
  - print-methods, 138
- \* **probability function**
  - d-methods, 36
- \* **programming**
  - distrMASK, 55
  - standardMethods, 162
- \* **pseudo inverse**
  - solve-methods, 159
  - sqr-methods, 162
- \* **pseudo random number**
  - r-methods, 146
- \* **pseudo-inverse distribution function**
  - q-methods, 139
  - q.r-methods, 140
- \* **quantile function**
  - q-methods, 139
  - q.r-methods, 140
- \* **random number generator**
  - r-methods, 146
- \* **random sample**
  - flat.mix, 70
  - RtoDPQ, 148
  - RtoDPQ.d, 150
  - RtoDPQ.LC, 151
  - simplifyr-methods, 158
- \* **range space of a distribution**
  - img-methods, 84
- \* **replacement function**
  - df-methods, 39
  - df1-methods, 39
  - df2-methods, 40
  - dimension-methods, 40
  - gaps-methods, 75
  - k-methods, 85
  - lambda-methods, 85
  - Length-methods, 92
  - location-methods, 98
  - m-methods, 102
  - Max-methods, 104
  - mean-methods, 105
  - meanlog-methods, 105
  - Min-methods, 106
  - n-methods, 108
  - name-methods, 108
  - ncp-methods, 113
  - pivot-methods, 127
  - prob-methods, 138
  - rate-methods, 146
  - scale-methods, 153
  - sd-methods, 154
  - sdlog-methods, 154
  - shape-methods, 155
  - shape1-methods, 155
  - shape2-methods, 156
  - size-methods, 158
  - width-methods, 188
- \* **right-continuous**
  - p-methods, 125
  - q.r-methods, 140
- \* **scale parameter**
  - scale-methods, 153
  - sd-methods, 154
- \* **set**
  - EuclideanSpace-class, 62
  - Lattice-class, 86
  - liesIn-methods, 93
  - Naturals-class, 109
  - Reals-class, 147
  - rSpace-class, 148
- \* **shape**
  - shape-methods, 155
  - shape1-methods, 155
  - shape2-methods, 156
- \* **space**
  - dim-methods, 40
  - dimension-methods, 40
  - EuclideanSpace-class, 62
  - img-methods, 84
  - Lattice-class, 86
  - Length-methods, 92
  - liesIn-methods, 93
  - liesInSupport, 93
  - Naturals-class, 109
  - pivot-methods, 127
  - Reals-class, 147

- rSpace-class, 148
  - width-methods, 188
- \* **spherical symmetry**
  - SphericalSymmetry, 160
  - SphericalSymmetry-class, 161
- \* **support of a distribution**
  - CompoundDistribution-class, 32
  - DiscreteDistribution-class, 46
  - UnivarMixingDistribution-class, 182
- \* **support**
  - getLow, getUp, 79
  - liesInSupport, 93
  - support-methods, 163
- \* **svd**
  - solve-methods, 159
  - sqrT-methods, 162
- \* **symmetric matrix**
  - PosDefSymmMatrix, 136
  - PosDefSymmMatrix-class, 137
- \* **symmetry**
  - DistributionSymmetry-class, 52
  - DistrSymmList, 58
  - DistrSymmList-class, 59
  - EllipticalSymmetry, 59
  - EllipticalSymmetry-class, 60
  - NoSymmetry, 117
  - NoSymmetry-class, 118
  - Symmetry-class, 163
- \* **truncation**
  - decomposePM-methods, 36
  - Huberize-methods, 80
  - simplifyD-methods, 156
  - Truncate-methods, 167
- \* **univariate distribution**
  - Distribution-class, 51
  - UnivarDistrList, 172
  - UnivariateDistribution-class, 174
- \* **utilities**
  - getLow, getUp, 79
  - standardMethods, 162
  - Version Management, 184
- \* **utility**
  - flat.LCD, 69
  - flat.mix, 70
  - getLabel, 78
  - RtoDPQ, 148
  - RtoDPQ.d, 150
  - RtoDPQ.LC, 151
  - simplifyr-methods, 158
- \* **version management**
  - Version Management, 184
- \* **virtual class**
  - OptionalParameter-class, 123
- \*, AbscontDistribution, numeric-method (operators-methods), 118
- \*, AcDcLcDistribution, AcDcLcDistribution-method (operators-methods), 118
- \*, AffLinAbscontDistribution, numeric-method (operators-methods), 118
- \*, AffLinDiscreteDistribution, numeric-method (operators-methods), 118
- \*, AffLinLatticeDistribution, numeric-method (operators-methods), 118
- \*, AffLinUnivarLebDecDistribution, numeric-method (operators-methods), 118
- \*, Cauchy, numeric-method (operators-methods), 118
- \*, CompoundDistribution, numeric-method (operators-methods), 118
- \*, DExp, numeric-method (operators-methods), 118
- \*, Dirac, Dirac-method (operators-methods), 118
- \*, Dirac, UnivariateDistribution-method (operators-methods), 118
- \*, Dirac, numeric-method (operators-methods), 118
- \*, DiscreteDistribution, numeric-method (operators-methods), 118
- \*, Exp, numeric-method (operators-methods), 118
- \*, ExpOrGammaOrChisq, numeric-method (operators-methods), 118
- \*, LatticeDistribution, numeric-method (operators-methods), 118
- \*, Lnorm, numeric-method (operators-methods), 118
- \*, Logis, numeric-method (operators-methods), 118
- \*, Norm, numeric-method (operators-methods), 118
- \*, Unif, numeric-method (operators-methods), 118
- \*, UnivarLebDecDistribution, numeric-method (operators-methods), 118

- \* ,UnivariateDistribution,Dirac-method  
(operators-methods), 118
- \* ,UnivariateDistribution,numeric-method  
(operators-methods), 118
- \* ,Weibull,numeric-method  
(operators-methods), 118
- \* ,numeric,LatticeDistribution-method  
(operators-methods), 118
- \* ,numeric,UnivariateDistribution-method  
(operators-methods), 118
- + ,AbscontDistribution,AbscontDistribution-method  
(operators-methods), 118
- + ,AbscontDistribution,DiscreteDistribution-method  
(operators-methods), 118
- + ,AbscontDistribution,numeric-method  
(operators-methods), 118
- + ,AcDcLcDistribution,AcDcLcDistribution-method  
(operators-methods), 118
- + ,AffLinAbscontDistribution,numeric-method  
(operators-methods), 118
- + ,AffLinDiscreteDistribution,numeric-method  
(operators-methods), 118
- + ,AffLinLatticeDistribution,numeric-method  
(operators-methods), 118
- + ,AffLinUnivarLebDecDistribution,numeric-method  
(operators-methods), 118
- + ,Binom,Binom-method  
(operators-methods), 118
- + ,Cauchy,Cauchy-method  
(operators-methods), 118
- + ,Cauchy,numeric-method  
(operators-methods), 118
- + ,Chisq,Chisq-method  
(operators-methods), 118
- + ,CompoundDistribution,numeric-method  
(operators-methods), 118
- + ,Dirac,Dirac-method  
(operators-methods), 118
- + ,Dirac,DiscreteDistribution-method  
(operators-methods), 118
- + ,Dirac,UnivariateDistribution-method  
(operators-methods), 118
- + ,Dirac,numeric-method  
(operators-methods), 118
- + ,DiscreteDistribution,AbscontDistribution-method  
(operators-methods), 118
- + ,DiscreteDistribution,DiscreteDistribution-method  
(operators-methods), 118
- + ,DiscreteDistribution,numeric-method  
(operators-methods), 118
- + ,ExpOrGammaOrChisq,ExpOrGammaOrChisq-method  
(operators-methods), 118
- + ,LatticeDistribution,DiscreteDistribution-method  
(operators-methods), 118
- + ,LatticeDistribution,LatticeDistribution-method  
(operators-methods), 118
- + ,LatticeDistribution,numeric-method  
(operators-methods), 118
- + ,Logis,numeric-method  
(operators-methods), 118
- + ,Nbinom,Nbinom-method  
(operators-methods), 118
- + ,Norm,Norm-method (operators-methods),  
118
- + ,Norm,numeric-method  
(operators-methods), 118
- + ,Pois,Pois-method (operators-methods),  
118
- + ,Unif,numeric-method  
(operators-methods), 118
- + ,UnivarLebDecDistribution,UnivarLebDecDistribution-method  
(operators-methods), 118
- + ,UnivarLebDecDistribution,numeric-method  
(operators-methods), 118
- + ,UnivariateDistribution,Dirac-method  
(operators-methods), 118
- + ,UnivariateDistribution,UnivariateDistribution-method  
(operators-methods), 118
- + ,UnivariateDistribution,numeric-method  
(operators-methods), 118
- + ,numeric,LatticeDistribution-method  
(operators-methods), 118
- + ,numeric,UnivariateDistribution-method  
(operators-methods), 118
- ,AcDcLcDistribution,AcDcLcDistribution-method  
(operators-methods), 118
- ,Dirac,Dirac-method  
(operators-methods), 118
- ,LatticeDistribution,LatticeDistribution-method  
(operators-methods), 118
- ,LatticeDistribution,UnivariateDistribution-method  
(operators-methods), 118
- ,LatticeDistribution,missing-method  
(operators-methods), 118
- ,LatticeDistribution,numeric-method  
(operators-methods), 118

- , Norm, missing-method  
(operators-methods), 118
- , UnivariateDistribution, LatticeDistribution-method  
(operators-methods), 118
- , UnivariateDistribution, UnivariateDistribution-method  
(operators-methods), 118
- , UnivariateDistribution, missing-method  
(operators-methods), 118
- , UnivariateDistribution, numeric-method  
(operators-methods), 118
- , numeric, Beta-method  
(operators-methods), 118
- , numeric, LatticeDistribution-method  
(operators-methods), 118
- , numeric, UnivariateDistribution-method  
(operators-methods), 118
- .logExact (Distribution-class), 51
- .logExact, Distribution-method  
(Distribution-class), 51
- .logExact, UnivarMixingDistribution-method  
(UnivarMixingDistribution-class),  
182
- .logExact-methods (Distribution-class),  
51
- .lowerExact (Distribution-class), 51
- .lowerExact, Distribution-method  
(Distribution-class), 51
- .lowerExact, UnivarMixingDistribution-method  
(UnivarMixingDistribution-class),  
182
- .lowerExact-methods  
(Distribution-class), 51
- .mergegaps2, 76
- .trunc.low, 168
- .trunc.up, 168
- / , AcDcLcDistribution, AcDcLcDistribution-method  
(operators-methods), 118
- / , Dirac, Dirac-method  
(operators-methods), 118
- / , LatticeDistribution, numeric-method  
(operators-methods), 118
- / , UnivariateDistribution, numeric-method  
(operators-methods), 118
- / , numeric, AcDcLcDistribution-method  
(operators-methods), 118
- / , numeric, Dirac-method  
(operators-methods), 118
- ^ , AcDcLcDistribution, AcDcLcDistribution-method  
(operators-methods), 118
- ^ , AcDcLcDistribution, Dirac-method  
(UnivarLebDecDistribution-class),  
177
- ^ , AcDcLcDistribution, Integer-method  
(operators-methods), 118
- ^ , AcDcLcDistribution, numeric-method  
(operators-methods), 118
- ^ , numeric, AcDcLcDistribution-method  
(operators-methods), 118
- abs, AbscontDistribution-method  
(Math-methods), 103
- abs, DiscreteDistribution-method  
(Math-methods), 103
- abs, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- AbscontDistribution, 12, 15, 17, 103
- AbscontDistribution-class, 14
- AcDcLcDistribution-class  
(UnivarLebDecDistribution-class),  
177
- acPart  
(UnivarLebDecDistribution-class),  
177
- acPart, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- acPart-methods  
(UnivarLebDecDistribution-class),  
177
- acPart<-  
(UnivarLebDecDistribution-class),  
177
- acPart<-, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- acPart<--methods  
(UnivarLebDecDistribution-class),  
177
- acWeight  
(UnivarLebDecDistribution-class),  
177
- acWeight, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- acWeight-methods  
(UnivarLebDecDistribution-class),



- 177
- acWeight<-  
    (UnivarLebDecDistribution-class),  
    177
- acWeight<-, UnivarLebDecDistribution-method  
    (UnivarLebDecDistribution-class),  
    177
- acWeight<--methods  
    (UnivarLebDecDistribution-class),  
    177
- AffLinAbscontDistribution-class  
    (AbscontDistribution-class), 14
- AffLinDiscreteDistribution-class  
    (DiscreteDistribution-class),  
    46
- AffLinDistribution-class  
    (AbscontDistribution-class), 14
- AffLinLatticeDistribution-class  
    (LatticeDistribution-class), 89
- AffLinUnivarLebDecDistribution-class  
    (UnivarLebDecDistribution-class),  
    177
- approxfun, 70, 149, 151, 152
- Arcsine (Arcsine-class), 18
- Arcsine-class, 18
- ARITHMETICS (distrARITH), 50
  
- Beta (Beta-class), 19
- Beta-class, 19
- BetaParameter-class, 21
- Binom (Binom-class), 22
- Binom-class, 22
- BinomParameter-class, 24
  
- Cauchy (Cauchy-class), 25
- Cauchy-class, 25
- CauchyParameter-class, 27
- Chisq (Chisq-class), 28
- Chisq-class, 28
- ChisqParameter-class, 30
- coerce, AbscontDistribution, UnivarLebDecDistribution-method  
    (UnivarLebDecDistribution-class),  
    177
- coerce, AffLinDiscreteDistribution, LatticeDistribution-method  
    (DiscreteDistribution-class),  
    46
- coerce, AffLinLatticeDistribution, AffLinDiscreteDistribution-method  
    (LatticeDistribution-class), 89
- coerce, AffLinUnivarLebDecDistribution, UnivarLebDecDistribution-method  
    (UnivarLebDecDistribution-class),  
    177
- coerce, CompoundDistribution, UnivarLebDecDistribution-method  
    (CompoundDistribution-class),  
    32
- coerce, DiscreteDistribution, LatticeDistribution-method  
    (DiscreteDistribution-class),  
    46
- coerce, DiscreteDistribution, UnivarLebDecDistribution-method  
    (UnivarLebDecDistribution-class),  
    177
- coerce, Distribution, DistrList-method  
    (DistrList-class), 54
- coerce, LatticeDistribution, DiscreteDistribution-method  
    (LatticeDistribution-class), 89
- coerce, UnivariateDistribution, UnivarDistrList-method  
    (UnivarDistrList-class), 173
- CompoundDistribution, 31, 33
- CompoundDistribution-class, 32
- conv2NewVersion, 6, 15, 52, 183
- conv2NewVersion (Version Management),  
    184
- conv2NewVersion, ANY-method (Version  
    Management), 184
- conv2NewVersion, LatticeDistribution-method  
    (Version Management), 184
- conv2NewVersion-methods (Version  
    Management), 184
- convpow, 9, 17, 48, 69, 181
- convpow (convpow-methods), 34
- convpow, AbscontDistribution-method  
    (convpow-methods), 34
- convpow, AcDcLcDistribution-method  
    (convpow-methods), 34
- convpow, Binom-method (convpow-methods),  
    34
- convpow, Cauchy-method  
    (convpow-methods), 34
- convpow, Dirac-method (convpow-methods),  
    34
- convpow, DiscreteDistribution-method  
    (convpow-methods), 34
- convpow, ExponentialDistribution, GammaOrChisq-method  
    (convpow-methods), 34
- convpow, LatticeDistribution-method  
    (convpow-methods), 34
- convpow, Nbinom-method

- (convpow-methods), 34
- convpow, Norm-method (convpow-methods), 34
- convpow, Pois-method (convpow-methods), 34
- convpow-methods, 34
- d (d-methods), 36
- d, Distribution-method (d-methods), 36
- d-methods, 36
- d.ac (UnivarLebDecDistribution-class), 177
- d.ac, UnivarLebDecDistribution-method (UnivarLebDecDistribution-class), 177
- d.ac-methods (UnivarLebDecDistribution-class), 177
- d.discrete (UnivarLebDecDistribution-class), 177
- d.discrete, UnivarLebDecDistribution-method (UnivarLebDecDistribution-class), 177
- d.discrete-methods (UnivarLebDecDistribution-class), 177
- decomposePM, 122
- decomposePM (decomposePM-methods), 36
- decomposePM, AbscontDistribution-method (decomposePM-methods), 36
- decomposePM, DiscreteDistribution-method (decomposePM-methods), 36
- decomposePM, UnivarLebDecDistribution-method (decomposePM-methods), 36
- decomposePM-methods, 36
- DefaultNrFFTGridPointsExponent (distributions), 56
- DefaultNrGridPoints (distributions), 56
- Defunct, 50
- density, 70, 149, 151, 152
- devNew, 124
- DExp (DExp-class), 37
- DExp-class, 37
- df (df-methods), 39
- df, Chisq-method (df-methods), 39
- df, ChisqParameter-method (df-methods), 39
- df, Td-method (df-methods), 39
- df, TParameter-method (df-methods), 39
- df-methods, 39
- df1 (df1-methods), 39
- df1, Fd-method (df1-methods), 39
- df1, FParameter-method (df1-methods), 39
- df1-methods, 39
- df1<- (df1-methods), 39
- df1<- , Fd-method (df1-methods), 39
- df1<- , FParameter-method (df1-methods), 39
- df1<--methods (df1-methods), 39
- df2 (df2-methods), 40
- df2, Fd-method (df2-methods), 40
- df2, FParameter-method (df2-methods), 40
- df2-methods, 40
- df2<- (df2-methods), 40
- df2<- , Fd-method (df2-methods), 40
- df2<- , FParameter-method (df2-methods), 40
- df2<--methods (df2-methods), 40
- df<- (df-methods), 39
- df<- , Chisq-method (df-methods), 39
- df<- , ChisqParameter-method (df-methods), 39
- df<- , Td-method (df-methods), 39
- df<- , TParameter-method (df-methods), 39
- df<--methods (df-methods), 39
- digamma, AbscontDistribution-method (Math-methods), 103
- digamma, DiscreteDistribution-method (Math-methods), 103
- dim (dim-methods), 40
- dim, UnivariateDistribution-method (dim-methods), 40
- dim-methods, 40
- dimension (dimension-methods), 40
- dimension, EuclideanSpace-method (dimension-methods), 40
- dimension-methods, 40
- dimension<- (dimension-methods), 40
- dimension<- , EuclideanSpace-method (dimension-methods), 40
- dimension<--methods (dimension-methods), 40
- Dirac (Dirac-class), 41
- Dirac-class, 41
- DiracParameter-class, 42
- DiscreteDistribution, 44, 62, 70, 152

- DiscreteDistribution-class, [46](#)
- discretePart
  - (UnivarLebDecDistribution-class), [177](#)
- discretePart,UnivarLebDecDistribution-method
  - (UnivarLebDecDistribution-class), [177](#)
- discretePart-methods
  - (UnivarLebDecDistribution-class), [177](#)
- discretePart<-
  - (UnivarLebDecDistribution-class), [177](#)
- discretePart<-,UnivarLebDecDistribution-method
  - (UnivarLebDecDistribution-class), [177](#)
- discretePart<--methods
  - (UnivarLebDecDistribution-class), [177](#)
- discreteWeight
  - (UnivarLebDecDistribution-class), [177](#)
- discreteWeight,UnivarLebDecDistribution-method
  - (UnivarLebDecDistribution-class), [177](#)
- discreteWeight-methods
  - (UnivarLebDecDistribution-class), [177](#)
- discreteWeight<-
  - (UnivarLebDecDistribution-class), [177](#)
- discreteWeight<-,UnivarLebDecDistribution-method
  - (UnivarLebDecDistribution-class), [177](#)
- discreteWeight<--methods
  - (UnivarLebDecDistribution-class), [177](#)
- distr (distr-package), [5](#)
- distr-defunct, [49](#)
- distr-package, [5](#)
- distrARITH, [50](#)
- DistrCollapse (distroptions), [56](#)
- Distribution-class, [51](#)
- DistributionAggregate.Unique.Warn
  - (distroptions), [56](#)
- DistributionSymmetry-class, [52](#)
- DistrList, [53](#), [54](#), [173](#)
- DistrList-class, [54](#)
- distrMASK, [55](#)
- distroptions, [11](#), [56](#), [157](#), [159](#)
- DistrResolution (distroptions), [56](#)
- DistrSymmList, [58](#)
- DistrSymmList-class, [59](#)
- DoubleExponential (DExp-class), [37](#)
- ecdf, [70](#), [149](#), [151](#), [152](#)
- eigen, [160](#)
- EllipticalSymmetry, [59](#), [61](#)
- EllipticalSymmetry-class, [60](#)
- EmpiricalDistribution, [61](#)
- EuclideanSpace (EuclideanSpace-class), [62](#)
- EuclideanSpace-class, [62](#)
- Exp (Exp-class), [64](#)
- exp,AbscontDistribution-method
  - (Math-methods), [103](#)
- exp,DiscreteDistribution-method
  - (Math-methods), [103](#)
- exp,UnivarLebDecDistribution-method
  - (UnivarLebDecDistribution-class), [177](#)
- Exp-class, [64](#)
- ExpParameter-class, [65](#)
- Fd (Fd-class), [67](#)
- Fd-class, [67](#)
- flat.LCD, [69](#), [121–123](#), [181](#)
- flat.mix, [34](#), [69](#), [70](#), [157](#), [184](#)
- FParameter-class, [71](#)
- gamma,AbscontDistribution-method
  - (Math-methods), [103](#)
- gamma,DiscreteDistribution-method
  - (Math-methods), [103](#)
- gamma-methods (Math-methods), [103](#)
- Gammad (Gammad-class), [72](#)
- Gammad-class, [72](#)
- GammaParameter-class, [74](#)
- gaps, [6](#), [15](#)
- gaps (gaps-methods), [75](#)
- gaps,AbscontDistribution-method
  - (gaps-methods), [75](#)
- gaps,UnivarMixingDistribution-method
  - (UnivarMixingDistribution-class), [182](#)
- gaps-methods, [75](#)
- gaps<- (gaps-methods), [75](#)

- gaps<- , AbscontDistribution-method  
(gaps-methods), 75
- gaps<--methods (gaps-methods), 75
- Geom (Geom-class), 76
- Geom-class, 76
- GeomParameter-class (distr-defunct), 49
- getdistrOption (distrOptions), 56
- getLabel, 78
- getLow (getLow, getUp), 79
- getLow, AbscontDistribution-method  
(getLow, getUp), 79
- getLow, DiscreteDistribution-method  
(getLow, getUp), 79
- getLow, getUp, 79
- getLow, LatticeDistribution-method  
(getLow, getUp), 79
- getLow, UnivarLebDecDistribution-method  
(getLow, getUp), 79
- getLow, UnivarMixingDistribution-method  
(getLow, getUp), 79
- getLow-methods (getLow, getUp), 79
- getOption, 57, 124, 125
- getUp (getLow, getUp), 79
- getUp, AbscontDistribution-method  
(getLow, getUp), 79
- getUp, DiscreteDistribution-method  
(getLow, getUp), 79
- getUp, LatticeDistribution-method  
(getLow, getUp), 79
- getUp, UnivarLebDecDistribution-method  
(getLow, getUp), 79
- getUp, UnivarMixingDistribution-method  
(getLow, getUp), 79
- getUp-methods (getLow, getUp), 79
  
- Huberize, 17, 48, 107, 157, 168, 181
- Huberize (Huberize-methods), 80
- Huberize, AcDcLcDistribution-method  
(Huberize-methods), 80
- Huberize-methods, 80
- Hyper (Hyper-class), 81
- Hyper-class, 81
- HyperParameter-class, 82
  
- igamma, 84
- img (img-methods), 84
- img, Distribution-method (img-methods),  
84
- img-methods, 84
  
- initialize, AbscontDistribution-method  
(AbscontDistribution-class), 14
- initialize, AffLinAbscontDistribution-method  
(AbscontDistribution-class), 14
- initialize, AffLinDiscreteDistribution-method  
(DiscreteDistribution-class),  
46
- initialize, AffLinLatticeDistribution-method  
(LatticeDistribution-class), 89
- initialize, Arcsine-method  
(Arcsine-class), 18
- initialize, Beta-method (Beta-class), 19
- initialize, BetaParameter-method  
(BetaParameter-class), 21
- initialize, Binom-method (Binom-class),  
22
- initialize, BinomParameter-method  
(BinomParameter-class), 24
- initialize, Cauchy-method  
(Cauchy-class), 25
- initialize, CauchyParameter-method  
(CauchyParameter-class), 27
- initialize, Chisq-method (Chisq-class),  
28
- initialize, ChisqParameter-method  
(ChisqParameter-class), 30
- initialize, DExp-method (DExp-class), 37
- initialize, Dirac-method (Dirac-class),  
41
- initialize, DiracParameter-method  
(DiracParameter-class), 42
- initialize, DiscreteDistribution-method  
(DiscreteDistribution-class),  
46
- initialize, EuclideanSpace-method  
(EuclideanSpace-class), 62
- initialize, Exp-method (Exp-class), 64
- initialize, ExpParameter-method  
(ExpParameter-class), 65
- initialize, Fd-method (Fd-class), 67
- initialize, FParameter-method  
(FParameter-class), 71
- initialize, Gammad-method  
(Gammad-class), 72
- initialize, GammaParameter-method  
(GammaParameter-class), 74
- initialize, Geom-method (Geom-class), 76
- initialize, GeomParameter-method

- (distr-defunct), 49
- initialize,Hyper-method (Hyper-class), 81
- initialize,HyperParameter-method (HyperParameter-class), 82
- initialize,LatticeDistribution-method (LatticeDistribution-class), 89
- initialize,Lnorm-method (Lnorm-class), 95
- initialize,LnormParameter-method (LnormParameter-class), 97
- initialize,Logis-method (Logis-class), 99
- initialize,LogisParameter-method (LogisParameter-class), 101
- initialize,Naturals-method (Naturals-class), 109
- initialize,Nbinom-method (Nbinom-class), 110
- initialize,NbinomParameter-method (NbinomParameter-class), 112
- initialize,Norm-method (Norm-class), 114
- initialize,NormParameter-method (NormParameter-class), 116
- initialize,Pois-method (Pois-class), 133
- initialize,PoisParameter-method (PoisParameter-class), 135
- initialize,Reals-method (Reals-class), 147
- initialize,Td-method (Td-class), 164
- initialize,TParameter-method (TParameter-class), 166
- initialize,Unif-method (Unif-class), 168
- initialize,UnifParameter-method (UnifParameter-class), 170
- initialize,UniNormParameter-method (UniNormParameter-class), 171
- initialize,UnivariateDistribution-method (UnivariateDistribution-class), 174
- initialize,Weibull-method (Weibull-class), 185
- initialize,WeibullParameter-method (WeibullParameter-class), 187
- isOldVersion, 6, 15
- isOldVersion (Version Management), 184
- isOldVersion,ANY-method (Version Management), 184
- isOldVersion-methods (Version Management), 184
- k (k-methods), 85
- k,Hyper-method (k-methods), 85
- k,HyperParameter-method (k-methods), 85
- k-methods, 85
- k<- (k-methods), 85
- k<- ,Hyper-method (k-methods), 85
- k<- ,HyperParameter-method (k-methods), 85
- k<--methods (k-methods), 85
- ks.test, 141, 142
- lambda (lambda-methods), 85
- lambda,Pois-method (lambda-methods), 85
- lambda,PoisParameter-method (lambda-methods), 85
- lambda-methods, 85
- lambda<- (lambda-methods), 85
- lambda<- ,Pois-method (lambda-methods), 85
- lambda<- ,PoisParameter-method (lambda-methods), 85
- lambda<--methods (lambda-methods), 85
- Laplace (DExp-class), 37
- Lattice (Lattice-class), 86
- lattice (LatticeDistribution-class), 89
- lattice,LatticeDistribution-method (LatticeDistribution-class), 89
- Lattice-class, 86
- lattice-method (LatticeDistribution-class), 89
- LatticeDistribution, 87, 90, 92
- LatticeDistribution-class, 89
- Length (Length-methods), 92
- Length,Lattice-method (Length-methods), 92
- Length,LatticeDistribution-method (Length-methods), 92
- Length-methods, 92
- Length<- (Length-methods), 92
- Length<- ,Lattice-method (Length-methods), 92
- Length<- ,LatticeDistribution-method (Length-methods), 92
- Length<--methods (Length-methods), 92
- lgamma,AbscontDistribution-method (Math-methods), 103

- lgamma, DiscreteDistribution-method  
(Math-methods), 103
- lgamma-methods (Math-methods), 103
- liesIn (liesIn-methods), 93
- liesIn, EuclideanSpace, numeric-method  
(liesIn-methods), 93
- liesIn, Naturals, numeric-method  
(liesIn-methods), 93
- liesIn-methods, 93
- liesInSupport, 93
- liesInSupport, AbscontDistribution, numeric-method  
(liesInSupport), 93
- liesInSupport, Beta, numeric-method  
(liesInSupport), 93
- liesInSupport, Cauchy, numeric-method  
(liesInSupport), 93
- liesInSupport, DExp, numeric-method  
(liesInSupport), 93
- liesInSupport, DiscreteDistribution, numeric-method  
(liesInSupport), 93
- liesInSupport, Distribution, matrix-method  
(liesInSupport), 93
- liesInSupport, ExpOrGammaOrChisq, numeric-method  
(liesInSupport), 93
- liesInSupport, Fd, numeric-method  
(liesInSupport), 93
- liesInSupport, LatticeDistribution, numeric-method  
(liesInSupport), 93
- liesInSupport, Lnorm, numeric-method  
(liesInSupport), 93
- liesInSupport, Logis, numeric-method  
(liesInSupport), 93
- liesInSupport, Norm, numeric-method  
(liesInSupport), 93
- liesInSupport, Td, numeric-method  
(liesInSupport), 93
- liesInSupport, Unif, numeric-method  
(liesInSupport), 93
- liesInSupport, UnivarLebDecDistribution, numeric-method  
(liesInSupport), 93
- liesInSupport, UnivarMixingDistribution, numeric-method  
(liesInSupport), 93
- liesInSupport, Weibull, numeric-method  
(liesInSupport), 93
- liesInSupport-methods (liesInSupport),  
93
- Lnorm (Lnorm-class), 95
- Lnorm-class, 95
- LnormParameter-class, 97
- location (location-methods), 98
- location, Cauchy-method  
(location-methods), 98
- location, CauchyParameter-method  
(location-methods), 98
- location, Dirac-method  
(location-methods), 98
- location, DiracParameter-method  
(location-methods), 98
- location, Logis-method  
(location-methods), 98
- location, LogisParameter-method  
(location-methods), 98
- location-methods, 98
- location<- (location-methods), 98
- location<-, Cauchy-method  
(location-methods), 98
- location<-, CauchyParameter-method  
(location-methods), 98
- location<-, Dirac-method  
(location-methods), 98
- location<-, DiracParameter-method  
(location-methods), 98
- location<-, Logis-method  
(location-methods), 98
- location<-, LogisParameter-method  
(location-methods), 98
- location<--methods (location-methods),  
98
- log, AbscontDistribution-method  
(Math-methods), 103
- log, Dirac-method (Dirac-class), 41
- log, DiscreteDistribution-method  
(Math-methods), 103
- log, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- log-gamma-methods (Math-methods), 103
- log10, AbscontDistribution-method  
(Math-methods), 103
- log10, DiscreteDistribution-method  
(Math-methods), 103
- log10, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- log10-methods (Math-methods), 103
- Logis (Logis-class), 99

- Logis-class, 99
- LogisParameter-class, 101
- m (m-methods), 102
- m, Hyper-method (m-methods), 102
- m, HyperParameter-method (m-methods), 102
- m-methods, 102
- m<- (m-methods), 102
- m<-, Hyper-method (m-methods), 102
- m<-, HyperParameter-method (m-methods), 102
- m<--methods (m-methods), 102
- makeAbscontDistribution, 102
- MASKING (distrMASK), 55
- Math, 9, 103, 104, 119
- Math, AbscontDistribution-method (Math-methods), 103
- Math, AcDcLcDistribution-method (Math-methods), 103
- Math, Dirac-method (Dirac-class), 41
- Math, DiscreteDistribution-method (Math-methods), 103
- Math, UnivarLebDecDistribution-method (UnivarLebDecDistribution-class), 177
- Math-methods, 103
- Max (Max-methods), 104
- Max, Unif-method (Max-methods), 104
- Max, UnifParameter-method (Max-methods), 104
- Max-methods, 104
- Max<- (Max-methods), 104
- Max<-, Unif-method (Max-methods), 104
- Max<-, UnifParameter-method (Max-methods), 104
- Max<--methods (Max-methods), 104
- Maximum (Minimum-methods), 106
- Maximum, AcDcLcDistribution, AcDcLcDistribution-method (Minimum-methods), 106
- Maximum, AcDcLcDistribution, numeric-method (Minimum-methods), 106
- Maximum-methods (Minimum-methods), 106
- mean, 165
- mean (mean-methods), 105
- mean, Norm-method (mean-methods), 105
- mean, NormParameter-method (mean-methods), 105
- mean-methods, 105
- mean<- (mean-methods), 105
- mean<-, Norm-method (mean-methods), 105
- mean<-, NormParameter-method (mean-methods), 105
- mean<--methods (mean-methods), 105
- meanlog (meanlog-methods), 105
- meanlog, Lnorm-method (meanlog-methods), 105
- meanlog, LnormParameter-method (meanlog-methods), 105
- meanlog-methods, 105
- meanlog<- (meanlog-methods), 105
- meanlog<-, Lnorm-method (meanlog-methods), 105
- meanlog<-, LnormParameter-method (meanlog-methods), 105
- meanlog<--methods (meanlog-methods), 105
- Min (Min-methods), 106
- Min, Unif-method (Min-methods), 106
- Min, UnifParameter-method (Min-methods), 106
- Min-methods, 106
- Min<- (Min-methods), 106
- Min<-, Unif-method (Min-methods), 106
- Min<-, UnifParameter-method (Min-methods), 106
- Min<--methods (Min-methods), 106
- Minimum, 17, 48, 69, 157, 168, 181
- Minimum (Minimum-methods), 106
- Minimum, AbscontDistribution, AbscontDistribution-method (Minimum-methods), 106
- Minimum, AbscontDistribution, Dirac-method (Minimum-methods), 106
- Minimum, AbscontDistribution, numeric-method (Minimum-methods), 106
- Minimum, AcDcLcDistribution, AcDcLcDistribution-method (Minimum-methods), 106
- Minimum, AcDcLcDistribution, numeric-method (Minimum-methods), 106
- Minimum, DiscreteDistribution, DiscreteDistribution-method (Minimum-methods), 106
- Minimum, DiscreteDistribution, numeric-method (Minimum-methods), 106
- Minimum-methods, 106
- mixCoeff (UnivarMixingDistribution-class), 182
- mixCoeff, UnivarMixingDistribution-method (UnivarMixingDistribution-class),



- 182
- mixCoeff-methods
  - (UnivarMixingDistribution-class), 182
- mixCoeff<-
  - (UnivarMixingDistribution-class), 182
- mixCoeff<-, UnivarMixingDistribution-method
  - (UnivarMixingDistribution-class), 182
- mixCoeff<--methods
  - (UnivarMixingDistribution-class), 182
- mixDistr
  - (UnivarMixingDistribution-class), 182
- mixDistr, UnivarMixingDistribution-method
  - (UnivarMixingDistribution-class), 182
- mixDistr-methods
  - (UnivarMixingDistribution-class), 182
- mixDistr<-
  - (UnivarMixingDistribution-class), 182
- mixDistr<-, UnivarMixingDistribution-method
  - (UnivarMixingDistribution-class), 182
- mixDistr<--methods
  - (UnivarMixingDistribution-class), 182
- n (n-methods), 108
- n, Hyper-method (n-methods), 108
- n, HyperParameter-method (n-methods), 108
- n-methods, 108
- n<- (n-methods), 108
- n<-, Hyper-method (n-methods), 108
- n<-, HyperParameter-method (n-methods), 108
- n<--methods (n-methods), 108
- NA, 145
- name (name-methods), 108
- name, Parameter-method (name-methods), 108
- name, rSpace-method (name-methods), 108
- name-methods, 108
- name<- (name-methods), 108
- name<-, Parameter-method (name-methods), 108
- name<-, rSpace-method (name-methods), 108
- name<--methods (name-methods), 108
- Naturals (Naturals-class), 109
- Naturals-class, 109
- Nbinom (Nbinom-class), 110
- Nbinom-class, 110
- NbinomParameter-class, 112
- ncp (ncp-methods), 113
- ncp, Beta-method (ncp-methods), 113
- ncp, BetaParameter-method (ncp-methods), 113
- ncp, Chisq-method (ncp-methods), 113
- ncp, ChisqParameter-method
  - (ncp-methods), 113
- ncp, Fd-method (ncp-methods), 113
- ncp, FParameter-method (ncp-methods), 113
- ncp, Td-method (ncp-methods), 113
- ncp, TParameter-method (ncp-methods), 113
- ncp-methods, 113
- ncp<- (ncp-methods), 113
- ncp<-, Beta-method (ncp-methods), 113
- ncp<-, BetaParameter-method
  - (ncp-methods), 113
- ncp<-, Chisq-method (ncp-methods), 113
- ncp<-, ChisqParameter-method
  - (ncp-methods), 113
- ncp<-, Fd-method (ncp-methods), 113
- ncp<-, FParameter-method (ncp-methods), 113
- ncp<-, Td-method (ncp-methods), 113
- ncp<-, TParameter-method (ncp-methods), 113
- ncp<--methods (ncp-methods), 113
- newDevice (options), 124
- Norm (Norm-class), 114
- Norm-class, 114
- NormParameter-class, 116
- NoSymmetry, 117, 118
- NoSymmetry-class, 118
- NumbOfSummandsDistr
  - (CompoundDistribution-class), 32
- NumbOfSummandsDistr, CompoundDistribution-method
  - (CompoundDistribution-class), 32
- NumbOfSummandsDistr-methods



- (CompoundDistribution-class),  
32
- operators, 35
- operators (operators-methods), 118
- operators-methods, 9, 17, 26, 42, 48, 69,  
115, 118, 181
- OptionalMatrix-class
  - (OptionalParameter-class), 123
- OptionalParameter-class, 123
- options, 57, 124, 124, 125
- p (p-methods), 125
- p, Distribution-method (p-methods), 125
- p-methods, 125
- p.ac (UnivarLebDecDistribution-class),  
177
- p.ac, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- p.ac-methods  
(UnivarLebDecDistribution-class),  
177
- p.discrete  
(UnivarLebDecDistribution-class),  
177
- p.discrete, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- p.discrete-methods  
(UnivarLebDecDistribution-class),  
177
- p.l (p.l-methods), 125
- p.l, AbscontDistribution-method  
(p.l-methods), 125
- p.l, DiscreteDistribution-method  
(p.l-methods), 125
- p.l, UnivarLebDecDistribution-method  
(p.l-methods), 125
- p.l, UnivarMixingDistribution-method  
(p.l-methods), 125
- p.l-methods, 125
- p.r (p-methods), 125
- p.r, Distribution-method (p-methods), 125
- p.r-methods (p-methods), 125
- par, 129, 131
- param (param-methods), 126
- param, Distribution-method  
(param-methods), 126
- param-methods, 126
- Parameter-class, 126
- pivot (pivot-methods), 127
- pivot, Lattice-method (pivot-methods),  
127
- pivot, LatticeDistribution-method  
(pivot-methods), 127
- pivot-methods, 127
- pivot<- (pivot-methods), 127
- pivot<-, Lattice-method (pivot-methods),  
127
- pivot<-, LatticeDistribution-method  
(pivot-methods), 127
- pivot<--methods (pivot-methods), 127
- plot, 130, 131
- plot (plot-methods), 127
- plot, AbscontDistribution, missing-method  
(plot-methods), 127
- plot, AffLinUnivarLebDecDistribution, missing-method  
(plot-methods), 127
- plot, CompoundDistribution, missing-method  
(plot-methods), 127
- plot, DiscreteDistribution, missing-method  
(plot-methods), 127
- plot, DistrList, missing-method  
(plot-methods), 127
- plot, UnivarLebDecDistribution, missing-method  
(plot-methods), 127
- plot-methods, 127
- plot.default, 129–131
- plot.new, 124
- plot.stepfun, 129–131
- pmatch, 179
- points, 129
- Pois (Pois-class), 133
- Pois-class, 133
- PoisParameter-class, 135
- PosDefSymmMatrix, 136, 137
- PosDefSymmMatrix-class, 137
- PosSemDefSymmMatrix (PosDefSymmMatrix),  
136
- PosSemDefSymmMatrix-class  
(PosDefSymmMatrix-class), 137
- print, UnivariateDistribution-method  
(print-methods), 138
- print-methods, 138
- prob (prob-methods), 138
- prob, Binom-method (prob-methods), 138

- prob,BinomParameter-method  
(prob-methods), 138
- prob,DiscreteDistribution-method  
(prob-methods), 138
- prob,Geom-method (prob-methods), 138
- prob,GeomParameter-method  
(prob-methods), 138
- prob,Nbinom-method (prob-methods), 138
- prob,NbinomParameter-method  
(prob-methods), 138
- prob,UnivarLebDecDistribution-method  
(prob-methods), 138
- prob-methods, 138
- prob<- (prob-methods), 138
- prob<-,Binom-method (prob-methods), 138
- prob<-,BinomParameter-method  
(prob-methods), 138
- prob<-,DiscreteDistribution-method  
(prob-methods), 138
- prob<-,Geom-method (prob-methods), 138
- prob<-,GeomParameter-method  
(prob-methods), 138
- prob<-,Nbinom-method (prob-methods), 138
- prob<-,NbinomParameter-method  
(prob-methods), 138
- prob<--methods (prob-methods), 138
- q (q-methods), 139
- q,Distribution-method (q-methods), 139
- q-methods, 139
- q.ac (UnivarLebDecDistribution-class),  
177
- q.ac,UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- q.ac-methods  
(UnivarLebDecDistribution-class),  
177
- q.discrete  
(UnivarLebDecDistribution-class),  
177
- q.discrete,UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- q.discrete-methods  
(UnivarLebDecDistribution-class),  
177
- q.l (q-methods), 139
- q.l,Distribution-method (q-methods), 139
- q.l-methods (q-methods), 139
- q.r (q.r-methods), 140
- q.r,AbscontDistribution-method  
(q.r-methods), 140
- q.r,DiscreteDistribution-method  
(q.r-methods), 140
- q.r,UnivarLebDecDistribution-method  
(q.r-methods), 140
- q.r,UnivarMixingDistribution-method  
(q.r-methods), 140
- q.r-methods, 140
- qqbounds, 140, 145
- qqplot, 142, 142, 143, 145
- qqplot,ANY,ANY-method (qqplot), 142
- qqplot,UnivariateDistribution,UnivariateDistribution-method  
(qqplot), 142
- qqplot-methods (qqplot), 142
- r (r-methods), 146
- r,Distribution-method (r-methods), 146
- r-methods, 146
- r.ac (UnivarLebDecDistribution-class),  
177
- r.ac,UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- r.ac-methods  
(UnivarLebDecDistribution-class),  
177
- r.discrete  
(UnivarLebDecDistribution-class),  
177
- r.discrete,UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class),  
177
- r.discrete-methods  
(UnivarLebDecDistribution-class),  
177
- rate (rate-methods), 146
- rate,DExp-method (rate-methods), 146
- rate,Exp-method (rate-methods), 146
- rate,ExpParameter-method  
(rate-methods), 146
- rate-methods, 146
- rate<- (rate-methods), 146
- rate<-,DExp-method (rate-methods), 146
- rate<-,Exp-method (rate-methods), 146
- rate<-,ExpParameter-method  
(rate-methods), 146

- rate<--methods (rate-methods), 146
- rbeta, 20
- rbinom, 22, 24
- rcauchy, 25, 27
- rchisq, 28, 30, 165
- Reals (Reals-class), 147
- Reals-class, 147
- rexp, 37, 38, 64, 65
- rf, 67, 68
- rgamma, 72, 73
- rgeom, 76, 77
- rhyper, 81, 82
- rlnorm, 95, 97
- rlogis, 99, 101
- rnbinom, 110, 111
- rnorm, 5, 114, 115
- rpois, 133, 134
- rSpace-class, 148
- rt, 164, 165
- RtoDPQ, 13, 14, 17, 119, 148, 152, 176, 177
- RtoDPQ.d, 45, 49, 89, 92, 119, 150
- RtoDPQ.e (distrOptions), 56
- RtoDPQ.LC, 151, 176
- runif, 168, 170
- rweibull, 185, 186
  
- scale (scale-methods), 153
- scale, Cauchy-method (scale-methods), 153
- scale, CauchyParameter-method (scale-methods), 153
- scale, Chisq-method (scale-methods), 153
- scale, Gammad-method (scale-methods), 153
- scale, GammaParameter-method (scale-methods), 153
- scale, Logis-method (scale-methods), 153
- scale, LogisParameter-method (scale-methods), 153
- scale, Weibull-method (scale-methods), 153
- scale, WeibullParameter-method (scale-methods), 153
- scale-methods, 153
- scale<- (scale-methods), 153
- scale<-, Cauchy-method (scale-methods), 153
- scale<-, CauchyParameter-method (scale-methods), 153
- scale<-, Gammad-method (scale-methods), 153
- scale<-, GammaParameter-method (scale-methods), 153
- scale<-, Logis-method (scale-methods), 153
- scale<-, LogisParameter-method (scale-methods), 153
- scale<-, Weibull-method (scale-methods), 153
- scale<-, WeibullParameter-method (scale-methods), 153
- scale<--methods (scale-methods), 153
- sd, 154, 165
- sd (sd-methods), 154
- sd, Norm-method (sd-methods), 154
- sd, NormParameter-method (sd-methods), 154
- sd-methods, 154
- sd<- (sd-methods), 154
- sd<-, Norm-method (sd-methods), 154
- sd<-, NormParameter-method (sd-methods), 154
- sd<--methods (sd-methods), 154
- sdlog (sdlog-methods), 154
- sdlog, Lnorm-method (sdlog-methods), 154
- sdlog, LnormParameter-method (sdlog-methods), 154
- sdlog-methods, 154
- sdlog<- (sdlog-methods), 154
- sdlog<-, Lnorm-method (sdlog-methods), 154
- sdlog<-, LnormParameter-method (sdlog-methods), 154
- sdlog<--methods (sdlog-methods), 154
- setgaps, 15
- setgaps (gaps-methods), 75
- setgaps, AbscontDistribution-method (gaps-methods), 75
- setgaps, UnivarMixingDistribution-method (gaps-methods), 75
- setgaps-methods (gaps-methods), 75
- shape (shape-methods), 155
- shape, Chisq-method (shape-methods), 155
- shape, Exp-method (shape-methods), 155
- shape, Gammad-method (shape-methods), 155
- shape, GammaParameter-method (shape-methods), 155
- shape, Weibull-method (shape-methods), 155

- shape, WeibullParameter-method  
(shape-methods), 155
- shape-methods, 155
- shape1 (shape1-methods), 155
- shape1, Beta-method (shape1-methods), 155
- shape1, BetaParameter-method  
(shape1-methods), 155
- shape1-methods, 155
- shape1<- (shape1-methods), 155
- shape1<-, Beta-method (shape1-methods), 155
- shape1<-, BetaParameter-method  
(shape1-methods), 155
- shape1<--methods (shape1-methods), 155
- shape2 (shape2-methods), 156
- shape2, Beta-method (shape2-methods), 156
- shape2, BetaParameter-method  
(shape2-methods), 156
- shape2-methods, 156
- shape2<- (shape2-methods), 156
- shape2<-, Beta-method (shape2-methods), 156
- shape2<-, BetaParameter-method  
(shape2-methods), 156
- shape2<--methods (shape2-methods), 156
- shape<- (shape-methods), 155
- shape<-, Gammad-method (shape-methods), 155
- shape<-, GammaParameter-method  
(shape-methods), 155
- shape<-, Weibull-method (shape-methods), 155
- shape<-, WeibullParameter-method  
(shape-methods), 155
- shape<--methods (shape-methods), 155
- show, DistrList-method  
(DistrList-class), 54
- show, LatticeDistribution-method  
(print-methods), 138
- show, Symmetry-method (Symmetry-class), 163
- show, UnivariateDistribution-method  
(print-methods), 138
- show, UnivarLebDecDistribution-method  
(print-methods), 138
- show, UnivarMixingDistribution-method  
(print-methods), 138
- show-methods (print-methods), 138
- sign, AbscontDistribution-method  
(Math-methods), 103
- sign, AcDcLcDistribution-method  
(UnivarLebDecDistribution-class), 177
- sign, DiscreteDistribution-method  
(Math-methods), 103
- sign, UnivarLebDecDistribution-method  
(UnivarLebDecDistribution-class), 177
- sign-methods (Math-methods), 103
- simplifyD, 32, 34, 80, 107, 121–123, 167, 177, 180–182, 184
- simplifyD (simplifyD-methods), 156
- simplifyD, AbscontDistribution-method  
(simplifyD-methods), 156
- simplifyD, DiscreteDistribution-method  
(simplifyD-methods), 156
- simplifyD, UnivarLebDecDistribution-method  
(simplifyD-methods), 156
- simplifyD, UnivarMixingDistribution-method  
(simplifyD-methods), 156
- simplifyD-methods, 156
- simplifyr (simplifyr-methods), 158
- simplifyr, UnivariateDistribution-method  
(simplifyr-methods), 158
- simplifyr-methods, 158
- size (size-methods), 158
- size, Binom-method (size-methods), 158
- size, BinomParameter-method  
(size-methods), 158
- size, Geom-method (size-methods), 158
- size, Nbinom-method (size-methods), 158
- size, NbinomParameter-method  
(size-methods), 158
- size-methods, 158
- size<- (size-methods), 158
- size<-, Binom-method (size-methods), 158
- size<-, BinomParameter-method  
(size-methods), 158
- size<-, Geom-method (size-methods), 158
- size<-, Nbinom-method (size-methods), 158
- size<-, NbinomParameter-method  
(size-methods), 158
- size<--methods (size-methods), 158
- solve, 57, 159, 160, 162
- solve (solve-methods), 159
- solve, ANY, ANY-method (solve-methods),

- 159
- solve, ANY-method (solve-methods), 159
- solve, PosDefSymmMatrix, ANY-method (solve-methods), 159
- solve, PosDefSymmMatrix-method (solve-methods), 159
- solve, PosSemDefSymmMatrix, ANY-method (solve-methods), 159
- solve, PosSemDefSymmMatrix-method (solve-methods), 159
- solve-methods, 159
- SphericalSymmetry, 160, 161
- SphericalSymmetry-class, 161
- sqrt (sqrt-methods), 162
- sqrt, AbscontDistribution-method (AbscontDistribution-class), 14
- sqrt, AcDcLcDistribution-method (UnivarLebDecDistribution-class), 177
- sqrt, DiscreteDistribution-method (DiscreteDistribution-class), 46
- sqrt, LatticeDistribution-method (LatticeDistribution-class), 89
- sqrt, PosSemDefSymmMatrix-method (sqrt-methods), 162
- sqrt, UnivarLebDecDistribution-method (UnivarLebDecDistribution-class), 177
- sqrt-methods, 162
- standardMethods, 162
- SummandsDistr (CompoundDistribution-class), 32
- SummandsDistr, CompoundDistribution-method (CompoundDistribution-class), 32
- SummandsDistr-methods (CompoundDistribution-class), 32
- support, 6
- support (support-methods), 163
- support, DiscreteDistribution-method (support-methods), 163
- support, UnivarMixingDistribution-method (UnivarMixingDistribution-class), 182
- support-methods, 163
- svd, 160
- SymmCenter (Symmetry-class), 163
- SymmCenter, Symmetry-method (Symmetry-class), 163
- Symmetry (Distribution-class), 51
- Symmetry, Distribution-method (Distribution-class), 51
- Symmetry, UnivarMixingDistribution-method (UnivarMixingDistribution-class), 182
- Symmetry-class, 163
- Symmetry-methods (Distribution-class), 51
- Td (Td-class), 164
- Td-class, 164
- TParameter-class, 166
- Truncate, 17, 48, 80, 107, 157, 181
- Truncate (Truncate-methods), 167
- Truncate, AbscontDistribution-method (Truncate-methods), 167
- Truncate, DiscreteDistribution-method (Truncate-methods), 167
- Truncate, LatticeDistribution-method (Truncate-methods), 167
- Truncate, UnivarLebDecDistribution-method (Truncate-methods), 167
- Truncate-methods, 167
- TruncQuantile (distributions), 56
- type (Symmetry-class), 163
- type, Symmetry-method (Symmetry-class), 163
- Unif (Unif-class), 168
- Unif-class, 168
- UnifParameter-class, 170
- UniNormParameter-class, 171
- UnivarDistrList, 53, 172, 173, 174
- UnivarDistrList-class, 173
- UnivariateDistribution-class, 174
- UnivarLebDecDistribution, 176, 177
- UnivarLebDecDistribution-class, 177
- UnivarMixingDistribution, 157, 181, 182
- UnivarMixingDistribution-class, 182
- UnivDistrListOrDistribution-class (CompoundDistribution-class), 32
- use.generalized.inverse.by.default (distributions), 56

Version Management, [184](#)

warn.makeDNew (distroptions), [56](#)

WarningArith (distroptions), [56](#)

WarningSim (distroptions), [56](#)

Weibull (Weibull-class), [185](#)

Weibull-class, [185](#)

WeibullParameter-class, [187](#)

width (width-methods), [188](#)

width,Lattice-method (width-methods),  
[188](#)

width,LatticeDistribution-method  
(width-methods), [188](#)

width-methods, [188](#)

width<- (width-methods), [188](#)

width<- ,Lattice-method (width-methods),  
[188](#)

width<- ,LatticeDistribution-method  
(width-methods), [188](#)

width<--methods (width-methods), [188](#)

withgaps (distroptions), [56](#)

withSweave (distroptions), [56](#)