How to generate new distributions in packages "distr", "distrEx"

Peter Ruckdeschel*
Matthias Kohl†

Institut für Mathematik
Fakultät V - Mathematik und Naturwissenschaften
Carl von Ossietzky Universität Oldenburg
PObox 2503
26111 Oldenburg (Oldb)
Germany

e-Mail: peter.ruckdeschel@uni-oldenburg.de

Version control information:

Head URL:	svn+ssh://ruckdeschel@svn.r-forge.r-project.org/svnroot/distr/pkg/distr/vignettes/newDistributions-knitr.Rnw
Last changed date:	2022-11-12 14:50:48 +0100 (Sa, 12. Nov 2022)
Last changes revision:	1374
Version:	Revision 1374
Last changed by:	Peter Ruckdeschel (ruckdeschel)

August 29, 2024

Abstract

In this vignette, we give short examples how to produce new distributions in packages "distr" and "distrEx". This vignette refers to package version 2.7.

Basically there are three ways to produce new distributions in packages "distr" and "distrEx":

1. automatic generation of single distribution objects by arithmetics and the like
2. using generating functions to produce single distribution objects
3. defining new distribution classes / doing it from scratch

We will give short examples of all three of them.

*Universität Oldenburg, Oldenburg
†FH Furtwangen

1 Automatic generation by arithmetics and the like

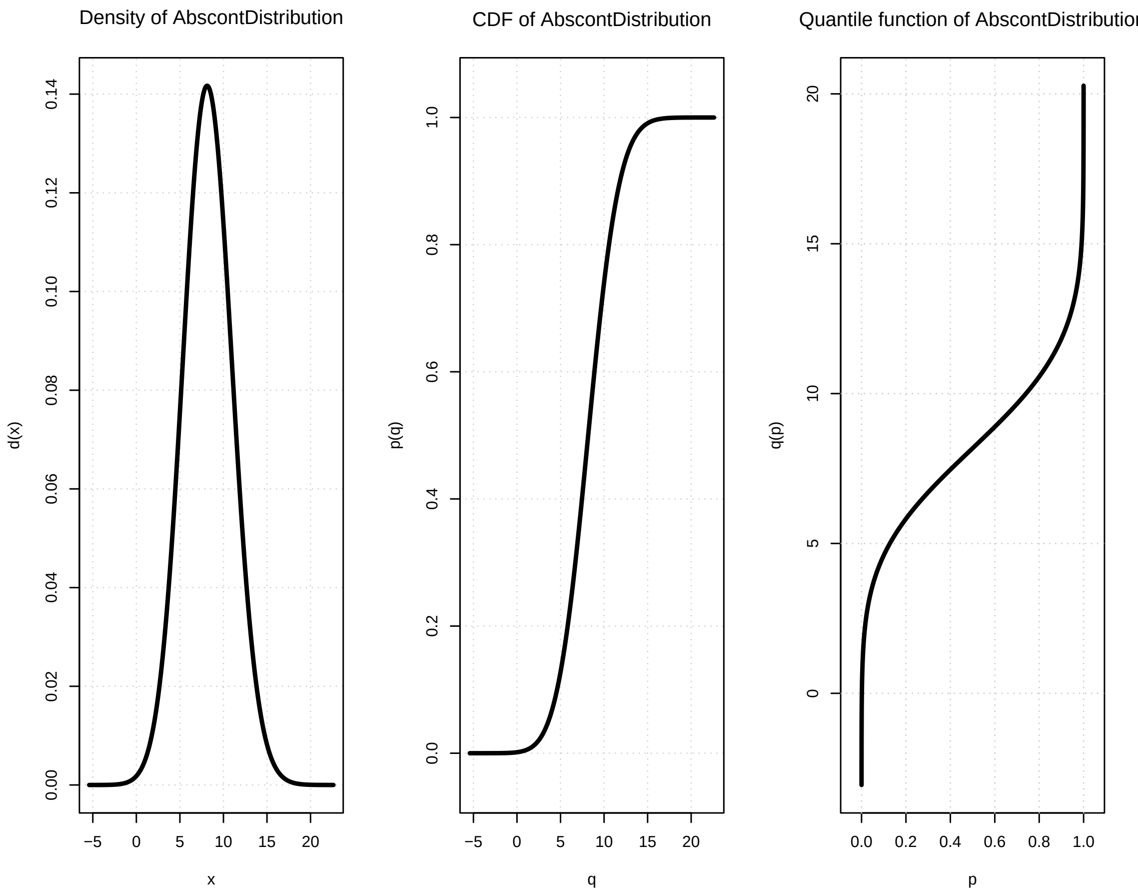
We have made available quite general arithmetical operations to our distribution objects, generating new image distribution objects automatically. As an example, try

```
require(distr)
N <- Norm(mean = 2, sd = 1.3)
P <- Pois(lambda = 1.2)
Z <- 2*N + 3 + P
Z

## Distribution Object of Class: AbscontDistribution

## Warning in methods::show(x):  arithmetics on distributions are understood as operations
on r.v.'s
## see 'distrARITH()'; for switching off this warning see '?distroptions'

plot(Z, panel.first = grid(), lwd=3)
```




```
p(Z)(0.4)

## [1] 0.002415402

q(Z)(0.3)

## [1] 6.705068

## in RStudio or Jupyter IRKernel, use q.l(.)(.) instead of q(.)(.)
Zs <- r(Z)(50)
Zs

##  [1] 10.352462  6.509291 11.173548  9.207643  7.287584  3.463618
##  [7]  7.680321  6.929722  7.418385 14.320462  8.038908  4.650845
## [13]  7.520613 10.023606  8.675249  3.044480  7.172894  9.586251
## [19]  8.912981  6.332528 10.482732  8.256915 11.204764 11.327381
## [25]  9.381405  4.734411  8.182785  7.445780  9.549316  6.920184
```

```
## [31]  9.877390  7.569935  5.851272  6.529696  5.926618  9.673276
## [37]  6.974117  7.846853 10.040872  8.624792  8.722698  8.599782
## [43]  9.022871  6.970110 10.291078  5.928526 10.630355  6.655079
## [49]  9.840334 12.038462
```

Comment:
Let N an object of class "Norm" with parameters **mean**=2, **sd**=1.3 and let P an object of class "Pois" with parameter lambda=1.2. Assigning to Z the expression 2∗N+3+P, a new distribution object is generated —of class "AbscontDistribution" in our case— so that identifying N, P, Z with random variables distributed according to N, P, Z, $\mathcal{L}(\mathtt{Z}) = \mathcal{L}(2 * \mathtt{N} + 3 + \mathtt{P})$, and writing p(Z)(0.4) we get $P(Z \leq 0.4)$, **q**(Z)(0.3) the 30%-quantile of Z, and with r(Z)(50) we generate 50 pseudo random numbers distributed according to Z, while the **plot** command generates the above figure.

In the environments of `RStudio`, see https://posit.co/ 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, from version 2.6 on, we provide function **q**.l (for left-continuous quantile function) as alias to our accessors **q**, so that all our package functionality also becomes available in `RStudio` and `IRKernel`.

There are caveats to take care about; for details refer to the (larger) vignette `distr` in package `"distrDoc"`.

2 Using generating functions

If you want to generate a single distribution object (without any particular parameter) generating functions are the method of choice:

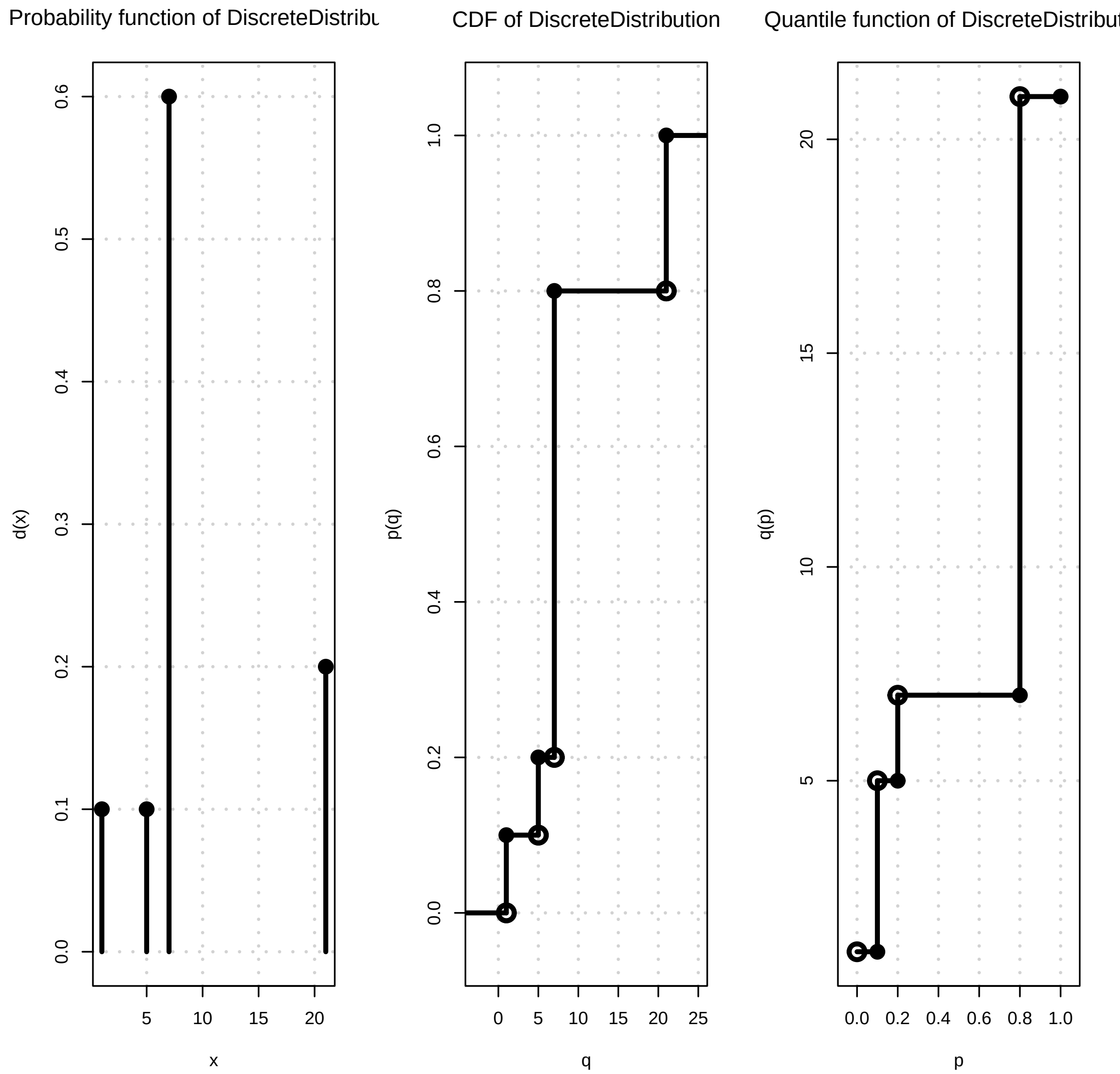
Objects of classes LatticeDistribution resp. DiscreteDistribution, AbscontDistribution, may be generated using the generating functions LatticeDistribution () resp. DiscreteDistribution () resp. AbscontDistribution(); see also the corresponding help.

E.g., to produce a discrete distribution with support $(1, 5, 7, 21)$ with corresponding probabilities $(0.1, 0.1, 0.6, 0.2)$ we may write

```
D <- DiscreteDistribution(supp = c(1,5,7,21), prob = c(0.1,0.1,0.6,0.2))
D

## Distribution Object of Class: DiscreteDistribution

plot(D, panel.first = grid(lwd=2), lwd = 3)
```

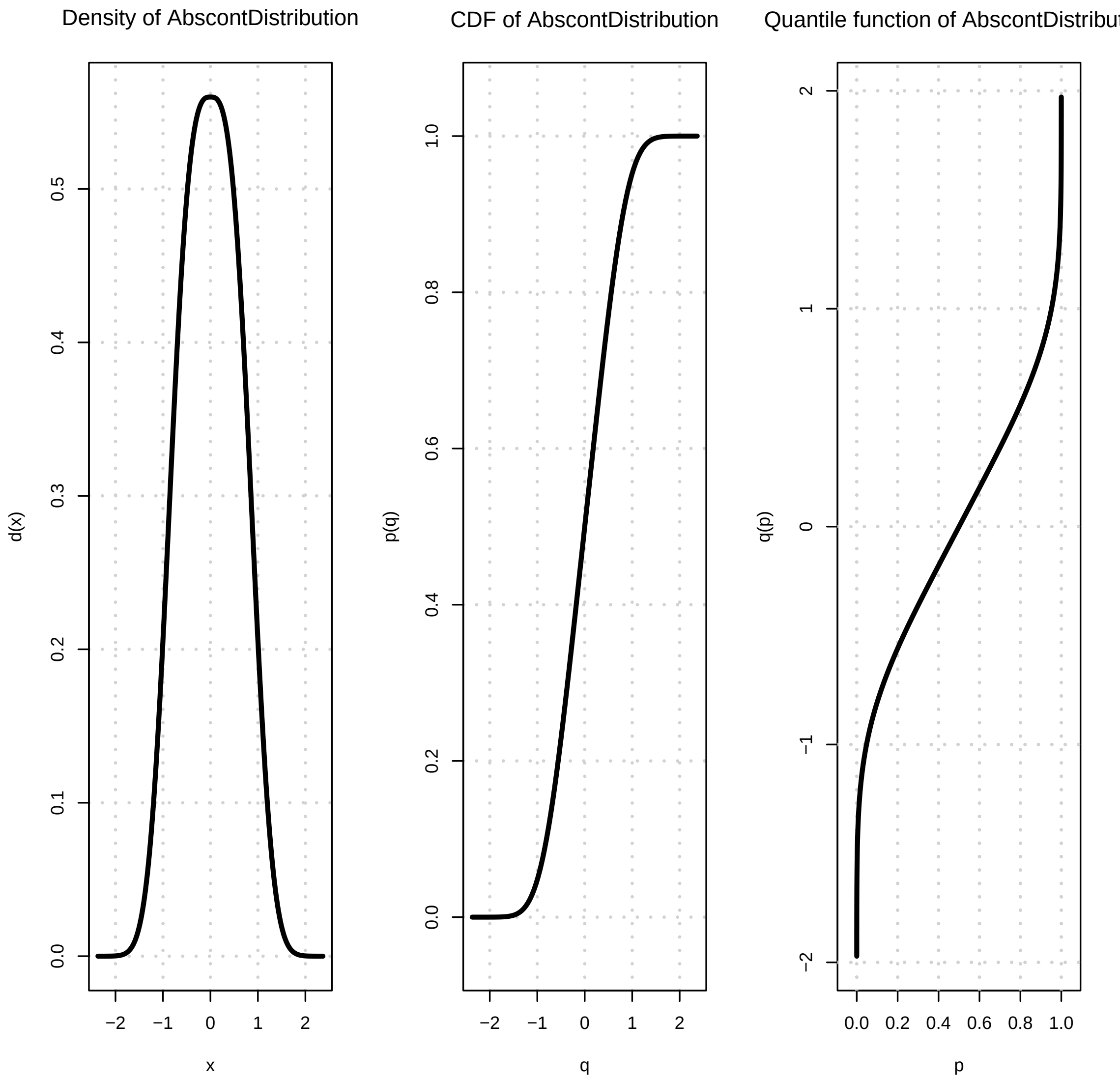


and to generate an absolutely continuous distribution with density proportional to $e^{-|x|^3}$, we write

```
AC <- AbscontDistribution(d = function(x) exp(-abs(x)^3), withStand = TRUE)
AC

## Distribution Object of Class: AbscontDistribution

plot(AC, panel.first = grid(lwd=2), lwd = 3)
```



3 Doing it from scratch

If you would like to create new parametric distributions, using already implemented `r`, `d`, `p`, and `q` functions (e.g. implementing additional distributions realized in another CRAN package), you should probably envisage introducing new distribution `S4` (sub-)classes and hence better look at the implementation of some discrete and continuous parametric distribution classes in package `"distr"`.
Hint: download the `.tar.gz` file; extract it to some `temp` folder; look at subdirectories `R` and `man`

The general procedure is as follows

1. introduce a new subclass of class Parameter
2. introduce a new subclass of LatticeDistribution/DiscreteDistribution (if discrete) or of class AbscontDistribution (if continuous).
3. define accessor and replacement functions for the "slots" of the parameter (e.g. "size" and "prob" in the binomial case), possibly with new generics
4. (possibly) define a validity function
5. define a generating function
6. if existing, define particular convolution methods or similar particular methods for this new distribution class
7. create `.Rd` files for the
   - parameter class
   - distribution class
8. if analytic expressions are available, define particular E-, **var**-, skewness-, and kurtosis-methods and if so, also document[1] the corresponding methods in the distribution class `.Rd` file

Let's go through the steps in the example case of the Binomial implementation in packages `"distr"` and `"distrEx"`:

1. in `"distr"`, see source in `R/AllClasses.R`,

```
## Class: BinomParameter
setClass("BinomParameter",
          representation = representation(size = "numeric", prob = "numeric"),
          prototype = prototype(size = 1, prob = 0.5, name =
                      gettext("Parameter of a Binomial distribution")
                      ),
          contains = "Parameter"
          )
```

2. in `"distr"`, see source in `R/AllClasses.R`,

[1]this is new, because so far, all E-, **var**-, skewness-, and kurtosis-methods for "basic" distributions are documented in the `"distrEx"` documentation to E, **var**, ..., but this would not be operational any longer for new derived classes, possibly defined in other, new packages

```r
## Class: binomial distribution
setClass("Binom",
          prototype = prototype(
                        r = function(n){ rbinom(n, size = 1,prob = 0.5) },
                        d = function(x, log = FALSE){
                                dbinom(x, size = 1, prob = 0.5, log = log)
                                          },
                        p = function(q, lower.tail = TRUE, log.p = FALSE ){
                                pbinom(q, size = 1, prob = 0.5,
                                       lower.tail = lower.tail, log.p = log.p)
                                          },
                        q = function(p, lower.tail = TRUE, log.p = FALSE ){
                                qbinom(p, size = 1, prob = 0.5,
                                       lower.tail = lower.tail, log.p = log.p)
                                          },
                        img = new("Naturals"),
                        param = new("BinomParameter"),
                        support = 0:1,
                        lattice = new("Lattice",
                                  pivot = 0, width = 1, Length = 2, name =
                                  gettext(
                                    "lattice of a Binomial distribution"
                                         )
                                  ),
                       .logExact = TRUE,
                       .lowerExact = TRUE
                        ),
          contains = "LatticeDistribution"
          )
```

3. in "distr", see source in R/BinomialDistribution.R,

```r
## Access Methods
setMethod("size", "BinomParameter", function(object) object@size)
setMethod("prob", "BinomParameter", function(object) object@prob)
## Replace Methods
setReplaceMethod("size", "BinomParameter",
                  function(object, value){ object@size <- value; object})
setReplaceMethod("prob", "BinomParameter",
                  function(object, value){ object@prob <- value; object})
```

and R/AllGenerics,

```r
if(!isGeneric("size"))
   setGeneric("size", function(object) standardGeneric("size"))
```

```
if(!isGeneric("prob"))
   setGeneric("prob", function(object) standardGeneric("prob"))
```

4. in "distr", see source in R/BinomialDistribution.R,

```
setValidity("BinomParameter", function(object){
  if(length(prob(object)) != 1)
    stop("prob has to be a numeric of length 1")
  if(prob(object) < 0)
    stop("prob has to be in [0,1]")
  if(prob(object) > 1)
    stop("prob has to be in [0,1]")
  if(length(size(object)) != 1)
    stop("size has to be a numeric of length 1")
  if(size(object) < 1)
    stop("size has to be a natural greater than 0")
  if(!identical(floor(size(object)), size(object)))
    stop("size has to be a natural greater than 0")
  else return(TRUE)
})
```

Class "BinomParameter" [in ".GlobalEnv"]

Slots:

Name: size prob name Class: numeric numeric character

Extends: Class "Parameter", directly Class "OptionalParameter", by class "Parameter", distance 2

5. in "distr", see source in R/BinomialDistribution.R,

```
Binom <- function(size = 1,prob = 0.5) new("Binom", size = size, prob = prob)
```

6. in "distr", see source in R/BinomialDistribution.R,

```
## Convolution for two binomial distributions Bin(n1,p1) and Bin(n2,p2)
## Distinguish cases
## p1 == p2 und p1 != p2


setMethod("+", c("Binom","Binom"),
          function(e1,e2){
            newsize <- size(e1) + size(e2)

            if(isTRUE(all.equal(prob(e1),prob(e2))))
               return(new("Binom", prob = prob(e1), size = newsize,
```

```r
                        .withArith = TRUE))

          return(as(e1, "LatticeDistribution") + e2)
        })
```

7. in "distr", see sources in

   - man/BinomParameter-class.Rd

```
\name{BinomParameter-class}
\docType{class}
\alias{BinomParameter-class}
\alias{initialize,BinomParameter-method}

\title{Class "BinomParameter"}
\description{ The parameter of a binomial distribution, used by Binom-class}
\section{Objects from the Class}{
Objects can be created by calls of the form
      \code{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.
}
\section{Slots}{
  \describe{
    \item{\code{prob}}{Object of class \code{"numeric"}:
          the probability of a binomial distribution }
    \item{\code{size}}{Object of class \code{"numeric"}:
          the size of a binomial distribution }
    \item{\code{name}}{Object of class \code{"character"}:
          a name / comment for the parameters }
  }
}
\section{Extends}{
Class \code{"Parameter"}, directly.
}
\section{Methods}{
  \describe{
    \item{initialize}{\code{signature(.Object = "BinomParameter")}:
          initialize method }
    \item{prob}{\code{signature(object = "BinomParameter")}: returns the slot
          \code{prob} of the parameter of the distribution }
    \item{prob<-}{\code{signature(object = "BinomParameter")}: modifies the slot
          \code{prob} of the parameter of the distribution }
    \item{size}{\code{signature(object = "BinomParameter")}: returns the slot
          \code{size} of the parameter of the distribution }
    \item{size<-}{\code{signature(object = "BinomParameter")}: modifies the slot
          \code{size} of the parameter of the distribution}
  }
}

\author{
  Thomas Stabla \email{statho3@web.de},\cr
  Florian Camphausen \email{fcampi@gmx.de},\cr
  Peter Ruckdeschel \email{peter.ruckdeschel@uni-oldenburg.de},\cr
  Matthias Kohl \email{Matthias.Kohl@stamats.de}
```

```
  }


\seealso{
\code{\link{Binom-class}}
\code{\link{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.
}
\keyword{distribution}
\concept{parameter}
\concept{Binomial distribution}
\concept{S4 parameter class}
```

- man/Binom-class.Rd

```
\name{Binom-class}
\docType{class}
\alias{Binom-class}
\alias{Binom}
\alias{initialize,Binom-method}

\title{Class "Binom" }
\description{The binomial distribution with \code{size} \eqn{= n}, by default
  \eqn{=1}, and
  \code{prob} \eqn{= p}, by default \eqn{=0.5}, has density
  \deqn{p(x) = {n \choose x} {p}^{x} {(1-p)}^{n-x}}{
    p(x) = choose(n,x) p^x (1-p)^(n-x)}
  for \eqn{x = 0, \ldots, n}.

  C.f.\code{\link[stats:Binomial]{rbinom}}
}
\section{Objects from the Class}{
Objects can be created by calls of the form \code{Binom(prob, size)}.
This object is a binomial distribution.
}
\section{Slots}{
  \describe{
    \item{\code{img}}{Object of class \code{"Naturals"}: The space of the
     image of this distribution has got dimension 1 and the
     name "Natural_Space". }
    \item{\code{param}}{Object of class \code{"BinomParameter"}: the parameter
          of this distribution (\code{prob}, \code{size}), declared at its
          instantiation }
    \item{\code{r}}{Object of class \code{"function"}: generates random
          numbers (calls function \code{rbinom}) }
    \item{\code{d}}{Object of class \code{"function"}: density function (calls
          function \code{dbinom}) }
    \item{\code{p}}{Object of class \code{"function"}: cumulative function
          (calls function \code{pbinom}) }
    \item{\code{q}}{Object of class \code{"function"}: inverse of the
           cumulative function (calls function \code{qbinom}).
    The quantile is defined as the smallest value x such that F(x) >= p, where
            F is the cumulative function. }
    \item{\code{support}}{Object of class \code{"numeric"}: a (sorted)
```

```
          vector containing the support of the discrete density function}
    \item{\code{.withArith}}{logical: used internally to issue warnings as to
          interpretation of arithmetics}
    \item{\code{.withSim}}{logical: used internally to issue warnings as to
        accuracy}
    \item{\code{.logExact}}{logical: used internally to flag the case where
    there are explicit formulae for the log version of density, cdf, and
    quantile function}
    \item{\code{.lowerExact}}{logical: used internally to flag the case where
    there are explicit formulae for the lower tail version of cdf and quantile
    function}
    \item{\code{Symmetry}}{object of class \code{"DistributionSymmetry"};
     used internally to avoid unnecessary calculations.}
  }
}
\section{Extends}{
Class \code{"DiscreteDistribution"}, directly.\cr
Class \code{"UnivariateDistribution"}, by class \code{"DiscreteDistribution"}.\cr
Class \code{"Distribution"}, by class \code{"DiscreteDistribution"}.
}
\section{Methods}{
  \describe{
    \item{+}{\code{signature(e1 = "Binom", e2 = "Binom")}: For two binomial
            distributions with equal probabilities the exact convolution
            formula is implemented thereby improving the general numerical
            accuracy.}
    \item{initialize}{\code{signature(.Object = "Binom")}: initialize method }
    \item{prob}{\code{signature(object = "Binom")}: returns the slot \code{prob}
            of the parameter of the distribution }
    \item{prob<-}{\code{signature(object = "Binom")}: modifies the slot
            \code{prob} of the parameter of the distribution }
    \item{size}{\code{signature(object = "Binom")}: returns the slot \code{size}
            of the parameter of the distribution }
    \item{size<-}{\code{signature(object = "Binom")}: modifies the slot
            \code{size} of the parameter of the distribution }
  }
}


\author{
  Thomas Stabla \email{statho3@web.de},\cr
  Florian Camphausen \email{fcampi@gmx.de},\cr
  Peter Ruckdeschel \email{peter.ruckdeschel@uni-oldenburg.de},\cr
  Matthias Kohl \email{Matthias.Kohl@stamats.de}
  }


\seealso{
\code{\link{BinomParameter-class}}
\code{\link{DiscreteDistribution-class}}
\code{\link{Naturals-class}}
\code{\link[stats:Binomial]{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
}
\keyword{distribution}
\concept{discrete distribution}
\concept{lattice distribution}
\concept{Binomial family}
\concept{Binomial distribution}
\concept{S4 distribution class}
\concept{generating function}
```

- you could have: `man/Binom.Rd` for the generating function; in the Binomial case, documentation is in `Binom-class.Rd`; but in case of the Gumbel distribution, in package `"RobExtremes"`, there is such an extra `.Rd` file

8. in `"distrEx"`, see sources in

```
## Lade nötiges Paket:  distrEx
## Extensions of Package 'distr' (version 2.9.2)
## Note:  Packages "e1071", "moments", "fBasics" should be attached /before/ package
"distrEx".  See distrExMASK().Note:  Extreme value distribution functionality has
been moved to
##       package "RobExtremes".  See distrExMOVED().
## For more information see ?"distrEx", NEWS("distrEx"), as well as
##   http://distr.r-forge.r-project.org/
## Package "distrDoc" provides a vignette to this package as well as to several
related packages; try vignette("distr").
##
## Attache Paket:  'distrEx'
## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     IQR, mad, median, var
```

- `Expectation.R`,

```
setMethod("E", signature(object = "Binom",
                         fun = "missing",
                         cond = "missing"),
    function(object, low = NULL, upp = NULL, ...){
    if(!is.null(low)) if(low <= min(support(object))) low <- NULL
    if(!is.null(upp)) if(upp >= max(support(object))) upp <- NULL
    if(is.null(low) && is.null(upp))
```

```
        return(size(object)*prob(object))
    else{
        if(is.null(low)) low <- -Inf
        if(is.null(upp)) upp <- Inf
        if(low == -Inf){
           if(upp == Inf) return(size(object)*prob(object))
           else return(m1df(object, upper = upp, ...))
        }else{
           E1 <- m1df(object, upper = low, ...)
           E2 <- if(upp == Inf)
                    size(object)*prob(object) else m1df(object, upper = upp, ...)
           return(E2-E1)
        }
    }
   })
```

- Functionals.R,

```
setMethod("var", signature(x = "Binom"),
    function(x,...){
    dots <- match.call(call = sys.call(sys.parent(1)),
                       expand.dots = FALSE)$"..."
    fun <- NULL; cond <- NULL; low <- NULL; upp <- NULL
    if(hasArg(low)) low <- dots$low
    if(hasArg(upp)) upp <- dots$upp
    if(hasArg(fun)||hasArg(cond)||!is.null(low)||!is.null(upp))
        return(var(as(x,"DiscreteDistribution"),...))
    else
        return(size(x)*prob(x)*(1-prob(x)))
    })
```

- skewness.R,

```
setMethod("skewness", signature(x = "Binom"),
    function(x,  ...){
    dots <- match.call(call = sys.call(sys.parent(1)),
                       expand.dots = FALSE)$"..."
    fun <- NULL; cond <- NULL; low <- NULL; upp <- NULL
    if(hasArg(low)) low <- dots$low
    if(hasArg(upp)) upp <- dots$upp
    if(hasArg(fun)||hasArg(cond)||!is.null(low)||!is.null(upp))
       return(skewness(as(x,"DiscreteDistribution"),...))
    else
        return((1-2*prob(x))/sqrt(size(x)*prob(x)*(1-prob(x))))
    })
```

- kurtosis.R,

```
setMethod("kurtosis", signature(x = "Binom"),
    function(x,  ...){
    dots <- match.call(call = sys.call(sys.parent(1)),
                       expand.dots = FALSE)$"..."
    fun <- NULL; cond <- NULL; low <- NULL; upp <- NULL
    if(hasArg(low)) low <- dots$low
    if(hasArg(upp)) upp <- dots$upp
    if(hasArg(fun)||hasArg(cond)||!is.null(low)||!is.null(upp))
       return(kurtosis(as(x,"DiscreteDistribution"),...))
    else
        p <- prob(x)
        return((1-6*p*(1-p))/(size(x)*p*(1-p)))
    })
```

The procedure will be similar for *any* new class of distributions.

Comment In the classes in package "distr" (historically the "oldest" in the development of this project), we still use initialize methods; this is no longer needed, if you provide generating functions; for this "more recent" approach, confer the realization of class Gumbel in package "RobExtremes".

4 Help needed / collaboration welcome

You are — as announced on http://distr.r-forge.r-project.org — very welcome to collaborate in this project! See in particular https://distr.r-forge.r-project.org/HOWTO-collaborate.txt
With this you should be able to start working.

References

[1] Ruckdeschel P. and Kohl, M. (2014): General Purpose Convolution Algorithm for Distributions in S4-Classes by means of FFT. *J. Statist. Software*, **59**(4): 1–25.

[2] Ruckdeschel P., Kohl M., Stabla T., and Camphausen F. (2006): S4 Classes for Distributions. *R-News*, **6**(2): 10–13. https://CRAN.R-project.org/doc/Rnews/Rnews_2006-2.pdf