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
PBox 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.Rnw
Last changed date: 2011-11-18 12:48:06 +0000 (Fr, 18 Nov 2011)
Last changes revision: 753
Version: Revision 753
Last changed by: Peter Ruckdeschel (ruckdeschel)

April 22, 2017

Abstract

In this vignette, we give short examples how to produce new distributions in packages "distr" and "distrEx". This vignette refers to package version 2.6.2.

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.

---

*University 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

\[
\begin{align*}
&> \text{require(distr)} \\
&> N \leftarrow \text{Norm(mean = 2, sd = 1.3)} \\
&> P \leftarrow \text{Pois(lambda = 1.2)} \\
&> Z \leftarrow 2*N + 3 + P \\
&> Z
\end{align*}
\]

Distribution Object of Class: AbscontDistribution

\[
\begin{align*}
&> \text{plot}(Z, \text{panel.first = grid()}, \text{lwd=2)} \\
&> \text{p}(Z)(0.4) \\
&[1] 0.002415387 \\
&> \text{q}(Z)(0.3) \\
&[1] 6.705068 \\
&> Zs \leftarrow \text{r}(Z)(50) \\
&> Zs
\end{align*}
\]

[49] 8.688506 2.267838
Comment:
Let \( N \) an object of class "Norm" with parameters \( \text{mean}=2, \text{sd}=1.3 \) and let \( P \) an object of class "Pois" with parameter \( \lambda=1.2 \). Assigning to \( Z \) the expression \( 2N+3+P \), a new distribution object is generated —of class "AbscontDistribution"— so that identifying \( N, P, Z \) with random variables distributed according to \( N, P, Z, L(Z) = L(2N+3+P) \), 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 \texttt{plot} command generates the above figure.

There a caveats to take care about; for details refer to the (larger) vignette \texttt{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 \texttt{LatticeDistribution} resp. \texttt{DiscreteDistribution}, \texttt{AbscontDistribution}, may be gen-
erated using the generating functions \texttt{LatticeDistribution()} resp. \texttt{DiscreteDistribution()} resp. \texttt{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

\begin{verbatim}
> 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)
\end{verbatim}

and to generate an absolutely continuos distribution with density proportional to \(e^{-|x|^3}\), we write

\begin{verbatim}
> AC ← AbscontDistribution(d = function(x) exp(-abs(x)^3), withStand = TRUE)
> AC

Distribution Object of Class: AbscontDistribution
\end{verbatim}
> plot(AC, panel.first = grid(), lwd = 2)

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 \( E \)-, \( \text{var} \)-, \( \text{skewness} \)-, and \( \text{kurtosis} \)-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, lines 187–196

   ```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, lines 972–1000

   ```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 ){
         pbinary(q, size = 1, prob = 0.5,
         lower.tail = lower.tail, log.p = log.p)
       },
     )
   ```

\(^{1}\)this is new, because so far, all \( E \)-, \( \text{var} \)-, \( \text{skewness} \)-, and \( \text{kurtosis} \)-methods for “basic” distributions are documented in the "distrEx" documentation to \( E \), \( \text{var} \),…, but this would not be operational any longer for new derived classes, possibly defined in other, new packages
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, lines 9–16, and 44–54

## 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 })

## wrapped access methods
setMethod("prob", "Binom",
  function(object) prob(param(object)))
setMethod("size", "Binom",
  function(object) size(param(object)))

## wrapped replace methods
setMethod("prob←", "Binom",
  function(object, value) new("Binom", prob = value,
       size = size(object)))
setMethod("size←", "Binom",
  function(object, value) new("Binom", prob = prob(object),
       size = value))

and R/AllGenerics, lines 159–162

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, lines 19–33

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)
}

5. in "distr", see source in R/BinomialDistribution.R, line 42

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

6. in "distr", see source in R/BinomialDistribution.R, lines 55–69

## 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, .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}{
\begin{itemize}
\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 \}
\end{itemize}

\section{Extends}{
Class \code{"Parameter"}, directly.

\section{Methods}{
\begin{itemize}
\item \code{initialize}\{\code{signature (.Object = "BinomParameter")}:
initialize method \}
\item \code{prob}\{\code{signature (object = "BinomParameter")}:
returns the slot \code{prob} of the parameter of the distribution \}
\item \code{prob}←\{\code{signature (object = "BinomParameter")}:
modifies the slot \code{prob} of the parameter of the distribution \}
\item \code{size}\{\code{signature (object = "BinomParameter")}:
returns the slot \code{size} of the parameter of the distribution \}
\item \code{size}←\{\code{signature (object = "BinomParameter")}:
modifies the slot \code{size} of the parameter of the distribution \}
\end{itemize}

\author{
Thomas Stabla \email{statho3@web.de},
Florian Camphausen \email{fcampi@gmx.de},
Peter Ruckdeschel \email{peter.ruckdeschel@uni-oldenburg.de},
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}
The quantile is defined as the smallest value \( x \) such that \( F(x) \geq p \), where \( F \) is the cumulative function. The quantile is defined as the smallest value \( x \) such that \( F(x) \geq p \), where \( F \) is the cumulative function.

C.f. \( \text{\S} \) \{\link{stats}{Binomial}\}(\text{\link{rbinom}})
8. in "distrEx", see sources in

\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}.,
Florian Camphausen \email{fcampi@gmx.de},
Peter Ruckdeschel \email{peter.ruckdeschel@uni-oldenburg.de},
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)(0.1) # x=0 is the smallest value x such that p(B)(x) ≥ 0.1.
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 # F 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
}

\concept{discrete distribution}
\concept{lattice distribution}
\concept{Binomial family}
\concept{Binomial distribution}
\concept{S4 distribution class}
\concept{generating function}

\item 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 "distrEx", there is such an extra .Rd file

8. in "distrEx", see sources in
• Expectation.R, lines 446–467

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

• Functionals.R, lines 219–230

```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, lines 69–80

```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, lines 90–102

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 "distrEx".

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 http://distr.r-forge.r-project.org/HOWTO-collaborate.txt

With this you should be able to start working.

References
