

RandVar: Implementation of random variables by means of S4 classes and methods

Matthias Kohl

e-Mail: `matthias.kohl@stamats.de`

Version control information:

Head URL: <file:///srv/svn/robast/pkg/RandVar/vignettes/RandVar.Rnw>
Last changed date: 2017-04-22 12:27:44 +0200 (Sat, 22 Apr 2017)
Last changes revision: 931
Version: Revision 931
Last changed by: Peter Ruckdeschel (ruckdeschel)

January 12, 2025

Abstract

In this package we implement random variables by means of S4 classes and methods. This vignette corresponds to Appendix D.2 in Kohl (2005) [3].

Contents

1	S4 Classes	2
2	Functions and Methods	3
3	Odds and Ends	9

1 S4 Classes

The S4 class `RandVariable` (cf. Figure 1) has the slots `Map`, `Domain` and `Range` where `Map` contains a list of functions which are measurable maps from `Domain` to `Range`. The elements contained in the list `Map` must be functions in one argument named `x`. We do not allow further parameters for these functions as this would lead to inconsistent objects. Strictly speaking, an object of class `RandVariable` would represent not only one random variable but a whole set of random variables depending on these parameters.

The slots `Domain` and `Range` are filled with an object of class `OptionalrSpace`; i.e., they contain `NULL` or an object of class `rSpace` (see package `distr` [4]). In case of `EuclRandVariable` and `RealRandVariable` the slot `Range` is filled with an object of class `EuclideanSpace` and `Reals`, respectively. The class `EuclRandMatrix` additionally has the slot `Dim` which is a vector of integers and contains the dimensions of the Euclidean random matrix.

Using these S4 classes there are two possibilities to implement a \mathbb{R}^k valued random variable. First, we could define a `EuclRandVariable` whose slot `Map` contains a list with one function which maps to \mathbb{R}^k ; i.e., the slot `Range` is a k -dimensional Euclidean space. Second, we could define a `EuclRandVariable` whose slot `Map` contains a list with n functions (projections) which map to \mathbb{R}^m where $k = m * n$. Now, the slot `Range` is an m -dimensional Euclidean space. Since it is sometimes convenient to regard a \mathbb{R}^k valued random variable as measurable map consisting of \mathbb{R}^{k_i} valued maps where $\sum k_i = k$, we introduce a further class called `EuclRandVarList`. With this class we can now define \mathbb{R}^k valued random variables as a list of \mathbb{R}^{k_i} valued random variables with compatible domains. More precisely, the elements of a `EuclRandVarList` may even have very different ranges (not necessarily Euclidean spaces) they only need to have compatible domains which is checked via the generic function `compatibleDomains`.

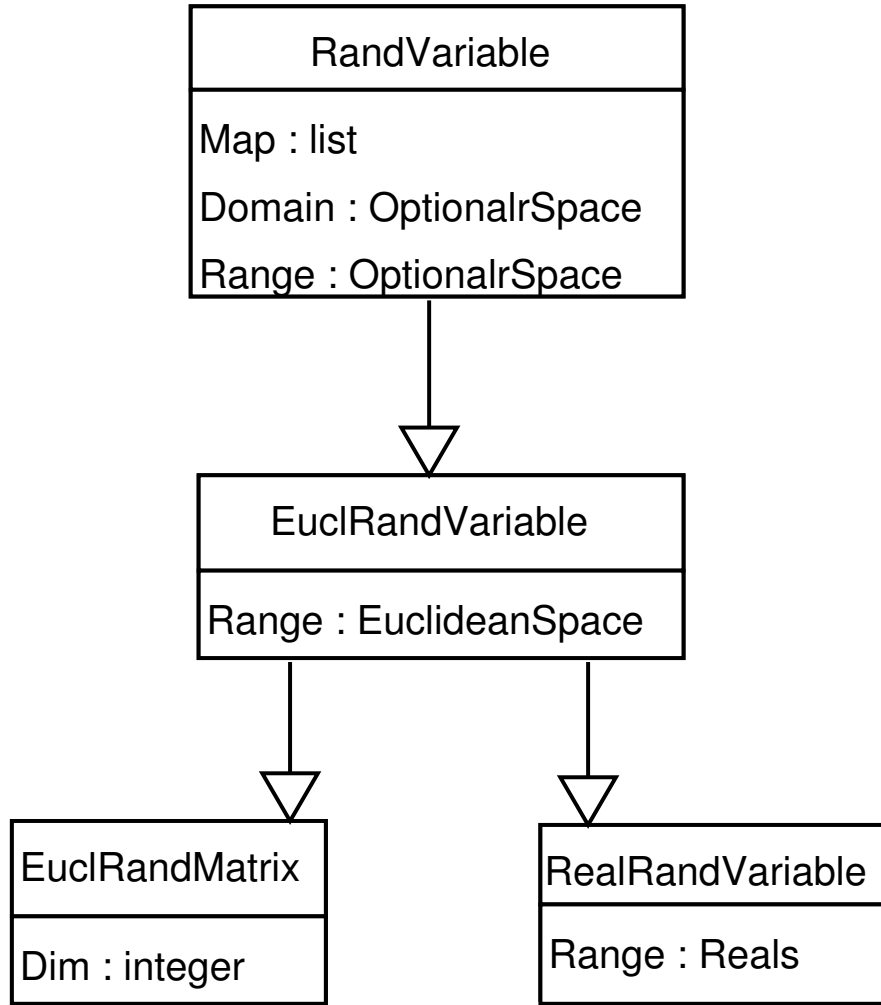


Figure 1: Class `RandVariable` and subclasses.

2 Functions and Methods

As in case of package `distrEx` [4], we follow the advices of Section 7.3 of [1] and [2]. That is, we introduce generating functions as well as accessor and replacement functions. A short description of the implemented generating functions is given in Table 1.

While there are accessor functions for all slots of the newly defined **S4** classes, replacement functions are only implemented for those slots a user should modify.

Our next goal was that one can use these classes of random variables like ordinary nu-

Generating Function	Description
<code>EuclRandMatrix</code>	Generates an object of class <code>EuclRandMatrix</code>
<code>EuclRandVariable</code>	Generates an object of class <code>EuclRandVariable</code>
<code>EuclRandVarList</code>	Generates an object of class <code>EuclRandVarList</code>
<code>RandVariable</code>	Generates an object of class <code>RandVariable</code>
<code>RealRandVariable</code>	Generates an object of class <code>RealRandVariable</code>

Table 1: Generating functions of package `RandVar`.

meric vectors or matrices. Hence, we overloaded the `S4` group generic functions `Arith` and `Math` as well as matrix multiplication `%*%`. For the matrix multiplication of `EuclRandVarLists` we additionally introduced the operator `%m%`. Now, if we have random variables `X` and `Y`, a numerical vector `v` and a numerical matrix `M` (with compatible dimensions) we can for instance generate

```
> library(RandVar)
> distroptions("withSweave" = TRUE) ## only for use in Sweave - document; set to FALSE else!
> (X <- RealRandVariable(Map = list(function(x){x}, function(x){x^2}), Domain = Reals(), Range = Reals()))

An object of class "RealRandVariable"
length of Map:      2
Domain:      Real Space with dimension 1
Range:      Real Space with dimension 1

> Map(X)

[[1]]
function(x){x}

[[2]]
function(x){x^2}

> evalRandVar(X, 2)

      [,1]
[1,]    2
[2,]    4

> evalRandVar(X, as.matrix(seq(2, 10, 2)))
```

```
, , 1
```

```
      [,1] [,2] [,3] [,4] [,5]  
[1,]    2    4    6    8   10  
[2,]    4   16   36   64  100
```

```
> R1 <- exp(X-1)
```

```
> Map(R1)
```

```
[[1]]
```

```
function (x)
```

```
{
```

```
  f1 <- function (x)
```

```
  {
```

```
    f1 <- function (x)
```

```
    {
```

```
      x
```

```
    }
```

```
    f1(x) - 1
```

```
  }
```

```
  exp(f1(x))
```

```
}
```

```
<environment: 0x5634aa520160>
```

```
[[2]]
```

```
function (x)
```

```
{
```

```
  f1 <- function (x)
```

```
  {
```

```
    f1 <- function (x)
```

```
    {
```

```
      x^2
```

```
    }
```

```
    f1(x) - 1
```

```
  }
```

```
  exp(f1(x))
```

```
}
```

```
<environment: 0x5634aa520160>
```

```
> R2 <- exp(X-1:2)
```

```
> Map(R2)
```

```
[[1]]
function (x)
{
  f1 <- function (x)
  {
    f1 <- function (x)
    {
      x
    }
    f1(x) - 1L
  }
  exp(f1(x))
}
<environment: 0x5634aa59e550>
```

```
[[2]]
function (x)
{
  f1 <- function (x)
  {
    f1 <- function (x)
    {
      x^2
    }
    f1(x) - 2L
  }
  exp(f1(x))
}
<environment: 0x5634aa59e550>
```

```
> (Y <- RealRandVariable(Map = list(function(x){sin(x)}), function(x){cos(x)}), Domain = Real)
```

```
An object of class "RealRandVariable"
length of Map:      2
Domain:      Real Space with dimension 1
Range:      Real Space with dimension 1
```

```
> Map(Y)
```

```
[[1]]
function(x){sin(x)}
```

```

[[2]]
function(x){cos(x)}

> R3 <- X %*% Y
> dimension(R3)

[1] 1

> #evalRandVar(R3, 2)
> 2*sin(2) + 2^2*cos(2)

[1] 0.1540075

> (R4 <- X %*% t(Y))

An object of class "EuclRandMatrix"
Dim of Map:      2 2
Domain:          Real Space with dimension 1
Range:           Euclidean Space with dimension 1

> dimension(R4)

[1] 4

> #evalRandVar(R4, 2)
> (M <- matrix(c(2*sin(2), 2^2*sin(2), 2*cos(2), 2^2*cos(2)), ncol = 2))

      [,1]      [,2]
[1,] 1.818595 -0.8322937
[2,] 3.637190 -1.6645873

> (R5 <- M %*% R4)

An object of class "EuclRandMatrix"
Dim of Map:      2 2
Domain:          Real Space with dimension 1
Range:           Real Space with dimension 1

```

We also implemented S4 methods for the generic function `E` of package `distrEx` [4]. That is, given some distribution `D`, respectively some conditional distribution `CD` and some random variable `X` we can compute the (conditional) expectation of `X` under `D`, respectively `CD` simply by

```

> D <- Norm()
> E(object = D, fun = X)

[1] 0.0000000 0.9999942

> E(D)

[1] 0

> var(D)

[1] 1

> (CD <- LMCondDistribution(theta = 1))

Distribution object of class: AbscontCondDistribution
  theta: 1
 intercept: 0
  scale: 1
## cond:
name:      conditioning by an Euclidean space
Range:      Euclidean Space with dimension 1

> E(object = CD, fun = X, cond = 2)

[1] 2.0000000 4.999993

> E(Norm(mean = 2))

[1] 2

> E(Norm(mean = 2), fun = function(x){x^2})

[1] 4.999993

```

for some given condition `cond`.

In addition, we define methods for the generic function `show` for the classes `RandVariable`, `EuclRandMatrix` and `EuclRandVarList`. There are also methods for the generic functions `dimension` (see package `distr` [4]), `length`, `ncol`, `nrow`, `t` and `[]` (cf. package `base`). For more details we refer to the corresponding help pages.

Finally, we introduce several new generic functions. A brief description of these functions is given in Table 2.

For more details about the full functionality of package `RandVar` we refer to the source code and the corresponding help pages, respectively.

Generic Function	Description
<code>%m%</code>	matrix multiplication for <code>EuclRandVarLists</code>
<code>compatibleDomains</code>	test if the domains of two random variables are compatible
<code>evalRandVar</code>	evaluation of random variables
<code>imageDistr</code>	image distribution of some distribution under some random variable
<code>numberOfMaps</code>	number of functions contained in the slots <code>Map</code> of the members of a <code>EuclRandVarList</code>

Table 2: New generic functions of package `RandVar` (without accessor and replacement functions).

3 Odds and Ends

The main issue is to reduce the computation time for methods using objects of class `RandVariable` and its subclasses as these classes play an important role in the computation of optimally robust estimators; confer Kohl (2005) [3]. In particular, we are looking for ways to increase the computation speed of `evalRandVar` and `E`.

References

- [1] Chambers J.M. *Programming with data. A guide to the S language*. Springer. <http://cm.bell-labs.com/stat/Sbook/index.html> 3
- [2] Gentleman R. *Object Orientated Programming. Slides of a Short Course held in Auckland*. <http://www.stat.auckland.ac.nz/S-Workshop/Gentleman/Methods.pdf> 3
- [3] Kohl M. *Numerical Contributions to the Asymptotic Theory of Robustness*. Dissertation, Universität Bayreuth. See also <http://stamats.de/ThesisMKohl.pdf> 1, 9
- [4] Ruckdeschel P., Kohl M., Stabla T., and Camphausen F. S4 Classes for Distributions. *R-News*, **6**(2): 10–13. https://CRAN.R-project.org/doc/Rnews/Rnews_2006-2.pdf See also <http://www.uni-bayreuth.de/departments/math/org/mathe7/RUCKDESCHEL/pubs/distr.pdf> 2, 3, 7, 8