that have appeared since the release of R 2.2.0 are described in the relevant regular sections. There is also an update of individuals and institutions who are providing support for the R Project via the R Foundation and there is an announcement and call for abstracts for the DSC 2007 conference to be held in Auckland, New Zealand.

A new addition with this issue is “News from the Bioconductor Project” by Seth Falcon, which describes important features of the 1.8 release of Bioconductor packages.

Finally, I would like to remind everyone of the upcoming useR! 2006 conference. It has attracted a huge number of presenters and promises to be a great success. I hope to see you there!

Paul Murrell
The University of Auckland, New Zealand
paul@stat.auckland.ac.nz

S4 Classes for Distributions

by Peter Ruckdeschel, Matthias Kohl, Thomas Stabla, and Florian Camphausen

distr is an R package that provides a conceptual treatment of random variables (r.v.'s) by means of S4–classes. A mother class `Distribution` is introduced with slots for a parameter and for methods `r`, `d`, `p`, and `q`, consecutively for simulation and for evaluation of the density, c.d.f., and quantile function of the corresponding distribution. All distributions of the **base** package are implemented as subclasses. By means of these classes, we can automatically generate new objects of these classes for the laws of r.v.'s under standard mathematical univariate transformations and under convolution of independent r.v.'s. In the **distrSim** and **distrTEst** packages, we also provide classes for a standardized treatment of simulations (also under contamination) and evaluations of statistical procedures on such simulations.

Motivation

R contains powerful techniques for virtually any useful distribution using the suggestive naming convention `[prefix]<name>` as functions, where `[prefix]` stands for `r`, `d`, `p`, or `q`, and `<name>` is the name of the distribution.

There are limitations of this concept: You can only use distributions that are already implemented in some library or for which you have provided an implementation. In many natural settings you want to formulate algorithms once for all distributions, so you should be able to treat the actual distribution `<name>` as if it were a variable.

You may of course paste together a prefix and the value of `<name>` as a string and then use `eval(parse(....))`. This is neither very elegant nor flexible, however.

Instead, we would prefer to implement the algorithm by passing an object of some distribution class as an argument to a function. Even better, though, we would use a generic function and let the S4-dispatching mechanism decide what to do at run-time. In particular, we would like to automatically generate the corresponding functions `r`, `d`, `p`, and `q` for the law of expressions like `X+3Y` for objects `X` and `Y` of class `Distribution`, or, more generally, of a transformation of X, Y under a function $f\colon \mathbb{R}^2 \to \mathbb{R}$ which is already realized as a function in R.

This is possible with package **distr**. As an example, try[1]

```
> require("distr")
Loading required package: distr
[1] TRUE
> N <- Norm(mean=2,sd=1.3)
> P <- Pois(lambda=1.2)
> (Z <- 2*N+3+P)
Distribution Object of Class: AbscontDistribution
> plot(Z)
> p(Z)(0.4)
[1] 0.002415384
> q(Z)(0.3)
[1] 6.70507
> r(Z)(10)
 [1] 11.072931  7.519611 10.567212  3.358912
 [5]  7.955618  9.094524  5.293376  5.536541
 [9]  9.358270 10.689527
```

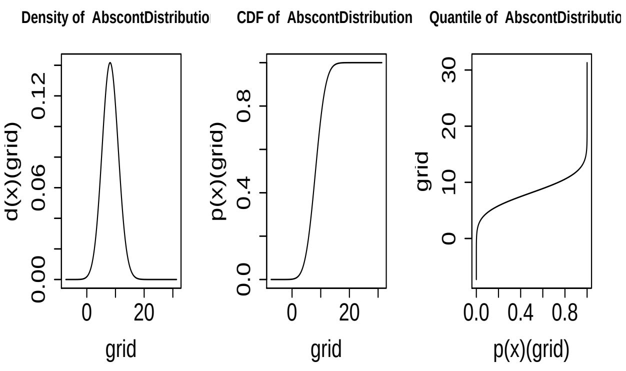

Figure 1: density, c.d.f. and quantile function of `Z`

[1]From version `1.7` on, additionally some startup message is shown after `require()`; confer Ruckdeschel et al. (2005, Section 5).

Comment:
N and P are assigned objects of classes "Norm" and "Pois", respectively, with corresponding parameters. The command Z <- 2*N+3+P generates a new distribution object and assigns it to Z — an object of class "AbscontDistribution". Identifying N, P, Z with r.v.'s distributed according to these distributions, $\mathcal{L}(\mathtt{Z}) = \mathcal{L}(2\mathtt{N} + 3 + \mathtt{P})$. Hence, p(Z)(0.4) returns $P(Z \le 0.4)$; q(Z)(0.3) returns the 30%-quantile of Z; r(Z)(10) generates 10 pseudo random numbers distributed according to Z; and the plot command produces Figure 1.

Concept

Our class design is deliberately open so that our classes may easily be extended by any volunteer in the R community; loose ends are multivariate distributions, time series distributions, and conditional distributions. As an exercise, the reader is encouraged to implement extreme value distributions from package **evd**. The greatest effort will be the documentation.

As to the naming of the slots, our goal was to preserve naming and notation from the **base** package as far as possible so that any programmer familiar with S could quickly use our **distr** package. In addition, as the distributions already implemented in R are all well tested and programmed, we use the existing r, d, p, and q-functions wherever possible, simply wrapping them in small code sniplets to our class hierarchy. All this should make intensive use of object orientation in order to be able to use inheritance and method overloading.

Contrary to the standard R-functions like rnorm, we only permit length 1 for parameters like mean, because we see the objects as implementations of univariate random variables, for which vector-valued parameters make no sense; rather one could gather several objects with possibly different parameters in a vector or list of distributions. Of course, the original functions rnorm, etc., remain unchanged and still allow vector-valued parameters.

Organization in classes

Loosely speaking, **distr** contains distribution classes and corresponding methods, **distrSim** simulation classes and corresponding methods, and **distrTEst** an evaluation class and corresponding methods. The latter two classes are to be considered only as tools that allow a unified treatment of simulations and evaluation of statistical estimation (perhaps also later, tests and predictions) under varying simulation situations.

Distribution classes

The Distribution class and its descendants implement the concept of a random variable/distribution in R. They all have a param slot for a parameter, an img slot for the range of the corresponding r.v., and r, d, p, and q slots.

Subclasses

At present, the **distr** package is limited to univariate distributions; these are derived from the subclass UnivariateDistribution, and as typical subclasses, we introduce classes for absolutely continuous (a.c.) and discrete distributions — AbscontDistribution and DiscreteDistribution. The latter has an extra slot support, a vector containing the support of the distribution, which is truncated to the lower/upper TruncQuantile in case of an infinite support. TruncQuantile is a global option of **distr** described in Ruckdeschel et al. (2005, Section 4).

As subclasses we have implemented all parametric families from the **base** package simply by providing wrappers to the original R-functions of form [prefix]<name>. More precisely, as subclasses of class AbscontDistribution, we have implemented Beta, Cauchy, Chisq, Exp, Logis, Lnorm, Norm, Unif, and Weibull; to avoid name conflicts, the F-, t-, and Γ-distributions are implemented as Fd, Td, Gammad. As subclasses of class DiscreteDistribution, we have implemented Binom, Dirac, Geom, Hyper, NBinom, and Pois.

Parameter classes

The param slot of a distribution class is of class Parameter. With method liesIn, some checking is possible; for details confer Ruckdeschel et al. (2005).

Simulation classes and evaluation class

Simulation classes and an evaluation class are provided in the **distrSim** and **distrTEst** packages, respectively. As a unified class for both "real" and simulated data, DataClass serves as a common mother class for both types of data. For simulations, we gather all relevant information in the derived classes Simulation and, for simulations under contamination, ContSimulation.

When investigating properties of a procedure by means of simulations, one typically evaluates this procedure on a lot of simulation runs, giving a result for each run. These results are typically worth storing. To organize all relevant information about these results, we introduce an Evaluation class.

For details concerning these groups of classes, consult Ruckdeschel et al. (2005).

Methods

We have made available quite general arithmetical operations for our distribution objects, generating new image distributions automatically. These arithmetics operate on the corresponding r.v.'s and **not** on the distributions. (For the latter, they only would make sense in restricted cases like convex combinations.)

Affine linear transformations

We have overloaded the operators `"+"`, `"-"`, `"*"`, and `"/"` such that affine linear transformations that involve only single univariate r.v.'s are available; i.e., expressions like `Y=(3*X+5)/4` are permitted for an object `X` of class `AbscontDistribution` or `DiscreteDistribution` (or some subclass), producing an object `Y` that is also of class `AbscontDistribution` or `DiscreteDistribution` (in general). Here the corresponding transformations of the `d`, `p`, and `q`-functions are determined analytically.

Unary mathematical operations

The `math` group of unary mathematical operations is also available for distribution classes; so expressions like `exp(sin(3*X+5)/4)` are permitted. The corresponding `r`-method consists in simply performing the transformation on the simulated values of `X`. The corresponding (default-) `d`-, `p`- and `q`-functions are obtained by simulation, using the technique described in the following subsection.

Construction of `d`, `p`, and `q` from `r`

For automatic generation of new distributions, in particular those arising as image distributions under transformations of correspondingly distributed r.v.'s, we provide ad hoc methods that should be overloaded by more exact ones wherever possible: Function `RtoDPQ` (re)constructs the slots `d`, `p`, and `q` from slot `r`, generating a certain number K of random numbers according to the desired law, where K is controlled by a global option of the **distr** package, as described in Ruckdeschel et al. (2005, Section 4). For a.c. distributions, a density estimator is evaluated along this sample, the distribution function is estimated by the empirical c.d.f., and, finally, the quantile function is produced by numerical inversion. Of course the result is rather crude as it relies only on the law of large numbers, but in this manner all transformations within the `math` group become available.

Convolution

A convolution method for two independent r.v.'s is implemented by means of explicit calculations for discrete summands, and by means of the FFT[2] if one of the summands is a.c. This method automatically generates the law of the sum of two independent variables X and Y of any univariate distributions — or, in `S4`-jargon, the addition operator `"+"` is overloaded for two objects of class `UnivariateDistribution` and corresponding subclasses.

Overloaded generic functions

Methods `print`, `plot`, and `summary` have been overloaded for our new classes `Distribution` (package **distr**), `DataClass`, `Simulation`, `ContSimulation` (package **distrSim**), as well as `EvaluationClass` (package **distrTEst**), to produce "pretty" output. For `Distribution`, `plot` displays the density/probability function, the c.d.f. and the quantile function of a distribution. For objects of class `DataClass` — or of a corresponding subclass — `plot` graphs the sample against the run index, and in the case of `ContSimulation`, the contaminating variables are highlighted by a different color. For an object of class `EvaluationClass`, `plot` yields a boxplot of the results of the evaluation.

Simulation

For the classes `Simulation` and `ContSimulation` in the **distrSim** package, we normally will not save the current values of the simulation, so when declaring a new object of either of the two classes, the slot `Data` will be empty (`NULL`). We may fill the slot by an application of the method `simulate` to the object. This has to be redone whenever another slot of the object is changed. To guarantee reproducibility, we use the slot `seed`. This slot is controlled and set through Paul Gilbert's **setRNG** package. For details confer Ruckdeschel et al. (2005).

Options

Contrary to the generally preferable functional style in `S`, we have chosen to set a few "constants" globally in order to retain binary notation for operators like `"+"`. Analogously to the `options` command in R, you can specify a number of these global constants. We only list them here; for details consult Ruckdeschel et al. (2005) or see `?distroptions`. Distribution options include

- `DefaultNrFFTGridPointsExponent`
- `DefaultNrGridPoints`
- `DistrResolution`

[2]Details to be found in Kohl et al. (2005)

- `RtoDPQ.e`
- `TruncQuantile`

These options may be inspected/modified by means of `distroptions()` and `getdistrOption()` which are defined just in the same way as `options()` and `getOption()`.

System/version requirements etc.

As our packages are completely written in R, there are no dependencies on the underlying OS. After some reorganization the three packages **distr**, **distrSim**, and **distrTEst**, as described in this paper, are only available from version `R 2.2.0` on, but there are precursors of our packages running under older versions of R; see Ruckdeschel et al. (2005). For the control of the seed of the random number generator in our simulation classes in package **distrSim**, we use Paul Gilbert's package **setRNG**, and for our startup messages, we need package **startupmsg** — both packages to be installed from `CRAN`. Packages **distr**, **distrSim**, and **distrTEst** are distributed under the terms of the GNU GPL Version 2, June 1991 (see `http://www.gnu.org/copyleft/gpl.html`).

Examples

In the `demo` folder of the **distr** package, the interested reader will find the sources for some worked out examples; we confine ourselves to discussing only one (the shortest) of them — `NormApprox.R`: In basic statistic courses, 12-fold convolution of Unif$(0, 1)$ variables is a standard illustration for the CLT. Also, this is an opaque generator of $\mathcal{N}(0, 1)$–variables; see Rice (1988, Example C, p. 163).

```
> N <- Norm(0,1) # a standard normal
> U <- Unif(0,1)
> U2 <- U + U
> U4 <- U2 + U2
> U8 <- U4 + U4
> U12 <- U4 + U8
> NormApprox <- U12 - 6
```

In the 3rd to 6th line, we try to minimize the number of calls to FFT for reasons of both time and accuracy; in the end, `NormApprox` contains the centered and standardized 12-fold convolution of Unif$(0, 1)$. Keep in mind, that instead of `U <- Unif(0,1)`, we might have assigned any distribution to `U` without having to change the subsequent code — except for a different centering/standardization. For a more sophisticated use of FFT for convolution powers see `nFoldConvolution.R` in the `demo` folder.

Next we explore the approximation quality of `NormApprox` for $\mathcal{N}(0, 1)$ graphically (see Figure 2):

```
> par(mfrow = c(2,1))
> x <- seq(-4,4,0.001)
> plot(x, d(NormApprox)(x), type = "l",
+    xlab = "",  ylab = "Density",
+    main = "Exact and approximated density")
> lines(x, d(N)(x), col = "red")
> legend(-4, d(N)(0),
+      legend = c("NormApprox",
+      "Norm(0,1)"), fill = c("black", "red"))
> plot(x, d(NormApprox)(x) - d(N)(x),
+    type = "l",  xlab = "",
+    ylab = "\"black\" - \"red\"",
+    col = "darkgreen", main = "Error")
> lines(c(-4,4), c(0,0))
```

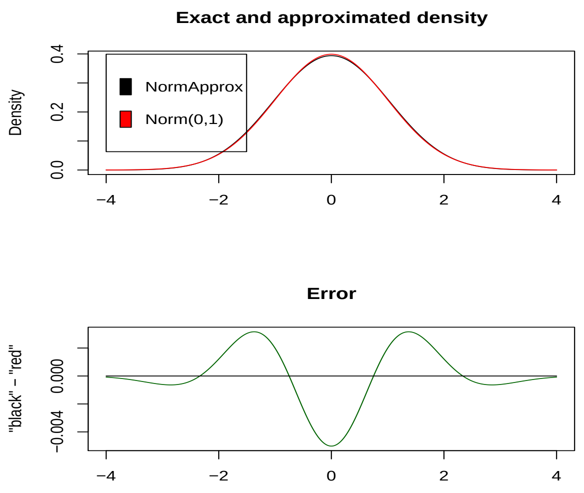

Figure 2: densities of $\mathcal{N}(0, 1)$ and `NormApprox` and their difference

Details/odds and ends

For further details of the implementation, see Ruckdeschel et al. (2005). In particular, we recommend Thomas Stabla's utility `standardMethods` for the automatic generation of `S4`-accessor and -replacement functions. For more details, see `?standardMethods`.

For odds and ends, consult the web-page for our packages,

`http://www.uni-bayreuth.de/departments/`
`/math/org/mathe7/DISTR/`.

Acknowledgement

We thank Martin Mächler and Josef Leydold for their helpful suggestions in conceiving the package. John Chambers also gave several helpful hints and insights. We got stimulating replies to an RFC on `r-devel` by Duncan Murdoch and Gregory Warnes. We also thank Paul Gilbert for drawing our attention to his package **setRNG** and making it available in a stand-alone version.

Bibliography

J. M. Chambers. *Programming with Data. A guide to the S language*. Springer, 1998. URL http://cm.bell-labs.com/cm/ms/departments/sia/Sbook/.

M. Kohl, P. Ruckdeschel, and T. Stabla. General Purpose Convolution Algorithm for Distributions in S4-Classes by means of FFT. Technical Report. Also available under http://www.uni-bayreuth.de/departments/math/org/mathe7/RUCKDESCHEL/pubs/comp.pdf, Feb. 2005.

J. A. Rice. *Mathematical statistics and data analysis*. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, California, 1988.

P. Ruckdeschel, M. Kohl, T. Stabla, and F. Camphausen. S4 *Classes for Distributions— a manual for packages* distr, distrSim, distrTEst, *version 1.7,* distrEx, *version 0.4-3,* May 2006. http://www.uni-bayreuth.de/departments/math/org/mathe7/DISTR.

Peter Ruckdeschel
Matthias Kohl
Thomas Stabla
Florian Camphausen
Mathematisches Institut
Universität Bayreuth
D-95440 Bayreuth
Germany
peter.ruckdeschel@uni-bayreuth.de

The regress function

An R function that uses the Newton Raphson algorithm for fitting certain doubly linear Gaussian models.

by David Clifford and Peter McCullagh

Introduction

The purpose of this article is to highlight the many uses of the `regress` function contained in the `regress` package. The function can be used to fit linear Gaussian models in which the mean is a linear combination of given covariates, and the covariance is a linear combination of given matrices. A Newton-Raphson algorithm is used to maximize the residual log likelihood with respect to the variance components. The regression coefficients are subsequently obtained by weighted least squares, and a further matrix computation gives the best linear predictor for the response on a further out-of-sample unit.

Many Gaussian models have a covariance structure that can be written as a linear combination of matrices, for example random effects models, polynomial spline models and some multivariate models. However it was our research on the nature of spatial variation in crop yields, McCullagh and Clifford (2006), that motivated the development of this function.

We begin this paper with a review of the kinds of spatial models we wish to fit and explain why a new function is needed to achieve this. Following this we discuss other examples that illustrate the broad range of uses for this function. Uses include basic random effects models, multivariate linear models and examples that include prediction and smoothing. The techniques used in these examples can also be applied to growth curves.

Spatial Models

McCullagh and Clifford (2006) model spatial dependence of crop yields as a planar Gaussian process in which the covariance function at points z, z' in the plane has the form

$$\sigma_0^2 \delta_{z-z'} + \sigma_1^2 K(|z-z'|) \qquad (1)$$

with non-negative coefficients σ_0^2 and σ_1^2. In the first term, Dirac's delta is the covariance function for white noise, and is independent on non-overlapping sets. The second term is a stationary isotropic covariance function from the Matérn class or power class.

McCullagh and Clifford (2006) proposed that the spatial correlation for any crop yield is well described by a convex combination of the Dirac function and the logarithmic function associated with the de Wijs process. The de Wijs process (de Wijs, 1951, 1953) is a special case of both the Matérn and power classes and is invariant to conformal transformation. The linear combination of white noise and the de Wijs process is called the conformal model. We examined many examples from a wide variety of crops worldwide, comparing the conformal model against the more general Matérn class. Apart from a few small-scale anisotropies, little evidence was found of systematic departures from the conformal model. Stein (1999) is a good reference for more details on spatial models.

It is important to point out that the de Wijs process and certain other boundary cases of the Matérn family are instances of generalized covariance functions corresponding to intrinsic processes. The fitting