Software for Distributions in R

David Scott\textsuperscript{1} \quad Diethelm Würtz\textsuperscript{2} \quad Christine Dong\textsuperscript{1}

\textsuperscript{1}Department of Statistics
The University of Auckland

\textsuperscript{2}Institut für Theoretische Physik
ETH Zürich

July 10, 2009
1 Introduction

2 Distributions in Base R

3 Design for Distribution Implementation
Outline

1. Introduction

2. Distributions in Base R

3. Design for Distribution Implementation
1 Introduction

2 Distributions in Base R

3 Design for Distribution Implementation
1 Introduction

2 Distributions in Base R

3 Design for Distribution Implementation
Distributions

- Distributions are how we model uncertainty
- There is well-established theory concerning distributions
- There are standard approaches for fitting distributions
- There are many distributions which have been found to be of interest
- Software implementation of distributions is a well-defined subject in comparison to say modelling of time-series
Introduction

- In base R there are 20 distributions implemented, at least in part
- All univariate—consider univariate distributions only
- Numerous other distributions have been implemented in R
  - CRAN packages solely devoted to one or more distributions
  - CRAN packages which implement distributions incidentally (e.g. VGAM)
  - implementations of distributions not on CRAN
- See the task view
  http://cran.r-project.org/web/views/Distributions.html
There are overlaps in coverage of distributions in R

Implementations of distributions in R are inconsistent
- naming of objects
- parameterizations
- function arguments
- functionality
- return structures

It is useful to discuss some standardization of implementation of software for distributions
1 Introduction

2 Distributions in Base R

3 Design for Distribution Implementation
Implementation in R is essentially the provision of *dpqr* functions: the density (or probability) function, distribution function, quantile or inverse distribution function and random number generation.

The distributions are the binomial (`binom`), geometric (`geom`), hypergeometric (`hyper`), negative binomial (`nbinom`), Poisson (`pois`), Wilcoxon signed rank statistic (`signrank`), Wilcoxon rank sum statistic (`wilcox`), beta (`beta`), Cauchy (`Cauchy`), non-central chi-squared (`chisq`), exponential (`exp`), F (`f`), gamma (`gamma`), log-normal (`lnorm`), logistic (`logis`), normal (`norm`), t (`t`), uniform (`unif`), Weibull (`weibull`), and Tukey studentized range (`tukey`) for which only the *p* and *q* functions are implemented.
Any experienced R user will be aware of the naming conventions for the density, cumulative distribution, quantile and random number generation functions for the base R distributions.

The argument lists for the dpqr functions are standard.

First argument is \( x, p, q \) and \( n \) for respectively a vector of quantiles, a vector of quantiles, a vector of probabilities, and the sample size.

\texttt{rwilcox} is an exception using \( nn \) because \( n \) is a parameter.

Subsequent arguments give the parameters.

The gamma distribution is unusual, with argument list \( \text{shape}, \text{rate} = 1, \text{scale} = 1/\text{rate} \).

This mechanism allows the user to specify the second parameter as either the scale or the rate.
dpqr Functions

- Other arguments differ among the dpqr functions
- The d functions take the argument log, the p and q functions the argument log.p
- These allow the extension of the range of accurate computation for these quantities
- The p and q functions have the argument lower.tail
- The dpqr functions are coded in C and may be found in the source software tree at /src/math/
- They are in large part due to Ross Ihaka and Catherine Loader
- Martin Mächler is now responsible for on-going maintenance
Testing and Validation

- The algorithms used in the dpqr functions are well-established algorithms taken from a substantial scientific literature.
- There are also tests performed, found in the directory tests in two files d-p-q-r-tests.R and p-r-random-tests.R.
- Tests in d-p-q-r-tests.R are “inversion tests” which check that \( q_{\text{dist}}(p_{\text{dist}}(x)) = x \) for values \( x \) generated by \( r_{\text{dist}} \).
- There are tests relying on special distribution relationships, and tests using extreme values of parameters or arguments.
- For discrete distributions equality of \( \text{cumsum}(d_{\text{dist}}(.)) = p_{\text{dist}}(.) \).
Tests in `p-r-random-tests.R` are based on an inequality of Massart:

$$\Pr\left(\sup_x |\hat{F}_n(x) - F(x)| > \lambda\right) \leq 2\exp(-2n\lambda^2)$$

where $\hat{F}_n$ is the empirical distribution function for a

- This is a version of the Dvoretsky-Kiefer-Wolfowitz inequality with the best possible constant, namely the leading 2 in the right hand side of the inequality
- The inequality above is true for all distribution functions, for all $n$ and $\lambda$
- Distributions are tested by generating a sample of size 10,000 and comparing the difference between the empirical distribution function and distribution function
1 Introduction

2 Distributions in Base R

3 Design for Distribution Implementation
What Should be Provided?

Besides the obvious dpqr functions, what else is needed?

- moments, at least low order ones
- the mode for unimodal distributions
- moment generating function and characteristic function
- functions for changing parameterisations
- functions for fitting of distributions and fit diagnostics
- goodness-of-fit tests
- methods associated with fit results: print, plot, summary, print.summary and coef
- for maximum likelihood fits, methods such as logLik and profile
To assess the fit of a distribution, diagnostic plots should be provided.

Some useful plots are:
- a histogram or empirical density with fitted density
- a log-histogram with fitted log-density
- a QQ-plot with optional fitted line
- a PP-plot with optional fitted line

For maximum likelihood estimation, contour plots and perspective plots for pairs of parameters, and likelihood profile plots.
Some generic fitting routines are currently available

- `mle` from `stats4` can be used to fit distributions but the log likelihood and starting values must be supplied

- `fitdistr` from `MASS` will automatically fit most of the distributions from base `R`

Other distributions can be fitted using `mle` by supplying the density and started values

In designing fitting functions, the structure of the object returned and the methods available are vital aspects

- `mle` returns an S4 object of class `mle`

- `fitdistr` produces and S3 object of class `fitdistr`
The methods available for an object of class `mle` are:

<table>
<thead>
<tr>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>confint</td>
<td>Confidence intervals from likelihood profiles</td>
</tr>
<tr>
<td>logLik</td>
<td>Extract maximized log-likelihood</td>
</tr>
<tr>
<td>profile</td>
<td>Likelihood profile generation</td>
</tr>
<tr>
<td>show</td>
<td>Display object briefly</td>
</tr>
<tr>
<td>summary</td>
<td>Generate object summary</td>
</tr>
<tr>
<td>update</td>
<td>Update fit</td>
</tr>
<tr>
<td>vcov</td>
<td>Extract variance-covariance matrix</td>
</tr>
</tbody>
</table>

For `fitdistr` the methods are `print`, `coef`, and `logLik`.

Neither function returns the data, so a plot method which produces suitable diagnostic plots is not possible.

Ideally a fit should return an object of class `distFit` say, and the `mle` class should extend that.
Some Principles

- Some principles are
  - the major guide to the design should be what exists in base R
  - the design should be logical with as few special cases as possible
  - the design should minimize the possibility of programming mistakes by users and developers
  - the design should simplify as much as possible the provision of the range of facilities needed to implement a distribution

- A naming scheme for functions is an important part of a standard
A Possible Naming Scheme

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddist</td>
<td>Density function or probability function</td>
</tr>
<tr>
<td>pdist</td>
<td>Cumulative distribution function</td>
</tr>
<tr>
<td>qdist</td>
<td>Quantile function</td>
</tr>
<tr>
<td>rdist</td>
<td>Random number generator</td>
</tr>
<tr>
<td>distMean</td>
<td>Theoretical mean</td>
</tr>
<tr>
<td>distVar</td>
<td>Theoretical variance</td>
</tr>
<tr>
<td>distSkew</td>
<td>Theoretical skewness</td>
</tr>
<tr>
<td>distKurt</td>
<td>Theoretical kurtosis</td>
</tr>
<tr>
<td>distMode</td>
<td>Mode of the density</td>
</tr>
<tr>
<td>distMoments</td>
<td>Theoretical moments (and mode)</td>
</tr>
<tr>
<td>distMGF</td>
<td>Moment generating function</td>
</tr>
<tr>
<td>distChFn</td>
<td>Characteristic function</td>
</tr>
<tr>
<td>distChangePars</td>
<td>Change parameterization of the distribution</td>
</tr>
<tr>
<td>distFit</td>
<td>Result of fitting the distribution to data</td>
</tr>
<tr>
<td>distTest</td>
<td>Test the distribution</td>
</tr>
</tbody>
</table>
Some Alternatives

- Use dots: `hyperb.mean`, usually deprecated because of confusion with S3 methods
- Use initial letters: `mhyperb`, but what about the mgf, multivariate distributions?
Function Arguments

- Specification of parameters could allow for the parameters to be specified as a vector
- This is helpful for maximum likelihood estimation
- The approach used for the gamma distribution can be used to allow for both single parameter specification and vector parameter specification
- Separate parameters should be in the order of location, scale, skewness, kurtosis
I don’t have a view on whether S3 or S4 classes should be used, but probably S4 classes should be aimed for. For a fit from a distribution, the class could be called `distfit`. A fit for a particular distribution would add the distribution name: `hyperbfit, distfit` with S3 methods. For S4 methods the class `distfit` would be extended. Similar ideas would be used for tests: a Kolomogorov-Smirnov test would have class `kstest, htest` with S3 methods.
Testing

- Testing is vital to quality of software and developers should provide test data and code
- Firstly test parameter sets should be provided, which cover the range of the parameter set
- Unit tests should be provided for all functions: the RUnit package supports this
- The distributions in Rmetrics have tests of this type, although further development seems warranted
- The Massart inequality seems ideal to use in testing
Final Thoughts

- The `distr` package is an object-oriented implementation of distributions
- It facilitates operations on distributions such as convolutions
- It uses sampling for calculation of moments and distribution functions
- The package `VarianceGamma` has been designed and implemented using these ideas
- Implementing the logarithm options of the dpq functions is quite difficult for the distributions in which we are interested