The xbar Package: Object-Oriented Distributions and Parameter Estimation in R

Ioannis Oikonomidis
National and Kapodistrian
University of Athens

Samis Trevezas
National and Kapodistrian
University of Athens

Abstract

The **xbar** package provides a comprehensive set of features for probabilities and mathematical statistics. It extends the range of available distribution families and facilitates the computation of key parametric quantities, such as moments and information-theoretic measures. The main focus of the package is parameter estimation through maximum likelihood and moment-based methods under an intuitive and efficient framework. All package features are available both in a **stats**-like syntax for entry-level users, and in an S4 object-oriented programming system for more experienced ones. The common `d`, `p`, `q`, `r` function family of each distribution, e.g. `dnorm()`, `pnorm()`, `qnorm()`, `rnorm()`, is enriched with the `ll` counterpart, e.g. `llnorm()`, that calculates the log-likelihood, the `e` counterpart, e.g. `enorm()`, that performs parameter estimation, and the `v` counterpart, e.g. `vnorm()`, that calculates the asymptotic variance-covariance matrix of an estimator. Furthermore, an S4-class distribution system is developed, so that these functions can be used generically. Moments and other parametric functions (mean, median, mode, variance, standard deviation, skewness, kurtosis, entropy, and Fisher information) are also included in the package. New distribution families, such as Dirichlet and multivariate gamma, not included in the **stats** package are made available. Parameter estimation is performed analytically if possible, while numerical optimization of the MLE (whenever required, e.g. the beta and gamma distribution families) is performed with computational efficiency, taking advantage of the score equation system to reduce the dimensionality of the optimization problem. Finally, the package includes functions to compute and plot the asymptotic variance and common finite-sample metrics (bias, variance, and root mean square error) of estimators, allowing their study and comparison. Overall, **xbar** addresses several limitations in the state of the art R packages in an attempt to provide some of the most fundamental methods of statistics in a unified package.

Keywords: Parameter estimation, Maximum likelihood estimation, Moment estimation, Score-adjusted moment estimation, Distribution, Entropy, Fisher information, S4, OOP, R.

1. Introduction

Package **xbar** is designed to cover a broad collection of distribution families, extending the functionalities of the **stats** package to support new families, parametric quantity computation and parameter estimation. All package features are available both in a **stats**-like syntax for entry-level users, and in an S4 object-oriented programming system for more experienced ones. This section introduces the state-of-the-art R packages in this direction, highlighting both the advantages and the shortcomings of each package.

1.1. Literature Review

The stats package

The **stats** package includes four functions for each distribution:

- The `d<name>()` function that calculates the density function f,
- the `p<name>()` function that calculates the distribution function F,
- the `q<name>()` function that calculates the (generalized) inverse distribution function F^{-1}, and
- the `r<name>()` function that simulates observations from the distribution.

This set of functions forms the foundation of statistical computing in R. They are implemented in C and Fortran, offering significantly higher computational efficiency compared to native R code, and are designed to be intuitive and accessible to new users. However, a notable limitation is their non-generic nature, as the distribution must be explicitly specified in the function name. This design is usually intuitive for new users, and is indeed a rational approach since the R community is composed of individuals with diverse backgrounds, many of which are not programmers. However, it can pose challenges when developing functions that accept a distribution and its parameters as input arguments. The straightforward implementation usually comes down to long `if - else if - else` statements or the equivalent `switch()` calls, such as the following:

```
densfun <- switch(distname,
                  "beta" = dbeta,
                  "cauchy" = dcauchy,
                  "chi-squared" = dchisq,
                  "exponential" = dexp,
                  "f" = df,
                  "gamma" = dgamma,
                  "geometric" = dgeom,
                  "log-normal" = dlnorm,
                  "lognormal" = dlnorm,
                  "logistic" = dlogis,
                  "negative binomial" = dnbinom,
                  "normal" = dnorm,
                  "poisson" = dpois,
                  "t" = mydt,
                  "weibull" = dweibull,
                  NULL)
```

Such implementations can be found in popular, important R packages, like **MASS**, from which the above code snippet is taken, or **smm**, a great package that implements semi-Markov modeling. Obviously, this problem can easily be tackled by object-oriented programming (OOP). Other programming languages commonly used in data analysis, like Python and Julia, do adopt such approaches.

The distr package ecosystem

The R package most similar to **xbar** is the **distr** package, which started as an S4 distribution system (Ruckdeschel, Kohl, Stabla, and Camphausen 2006; Ruckdeschel and Kohl 2014). Since then, a few other complementary packages have been developed to offer new capabilities, including **distrEx** for moment computation and **distrMod** for parameter estimation. This package ecosystem is similar to **xbar** in a number of ways, therefore their differences-and the thereof necessity for a new package-are explained in detail.

The **distr** framework does not strictly adhere to S4 conventions, as the `d()-p()-q()-r()` function family is generated *alongside* the random variable object. This design can lead to bugs upon modifying an object's parameter slots, as illustrated by the following example. Given a distribution object `D` from the $N(10, 1)$ distribution, a user wants to alter its mean to -10. This can be done in two different ways, using the **distr** function `mean<-` and accessing the parameter slot directly. In the second way, the `r()` function is not affected, since it was already created along with `D`.

```
library(distr)
set.seed(1)

D <- Norm(10, 1)
mean(D) <- -10
x <- r(D)(100) # r simulates from N(-10, 1)
mean(x) # -9.891113

D <- Norm(10, 1)
D@param@mean <- - 10
x <- r(D)(100) # r simulates from N(10, 1)
mean(x) # 9.962192
```

> **Note**
>
> **xbar** actually started as a short extension of the **distr** package ecosystem. The initial purpose was to code the score-adjusted moment estimators (SAME) developed by the authors in *cite* and study their properties. However, upon examining the S4 implementation, realizing the extra complexity accompanying the definition of new distribution families, and eventually falling in the aforementioned bug, it was decided that it was preferable to create a new S4 system that could accommodate the needs of the research team.

The distr6 package

The second alternative for a OOP distribution system in R is the **distr6** package, based on the R6 class system (Sonabend and Kiraly 2025). Before analyzing its capabilities, it seems fitting to quote the **distr6** development team:

> **What is distr6?**
>
> **distr6** is a unified and clean interface to organize the probability distributions implemented in R into one R6 object-oriented package, as well as adding distributions yet to implemented

> in R [...]. **distr6** extends the work of Peter Ruckdeschel, Matthias Kohl et al. who created the first object-oriented (OO) interface for distributions using S4. Their **distr** package is currently the gold-standard in R for OO distribution handling.

At the time of writing, **distr6** is not available on CRAN, and according to the authors *"it will not be for the foreseeable future"*, although it can be easily downloaded from github:

```
devtools::install_github("xoopR/distr6")
```

It is important to underline that the `R6` system handles memory more efficiently in comparison with S3 and S4, avoiding the creation of object copies, as detailed in *citation*. However, many R users are not familiarized with the R6 system, since the majority of R functions are implemented in the `S3` system. This is the main reason the S4 system was chosen for the development of **xbar**. The **distr6** package indeed includes a great number of distribution families. Unfortunately, it does not cover parameter estimation, which is the main reason behind the development of the **xbar** package.

Parameter Estimation in R packages

Parameter estimation in the iid (independent and identically distributed) framework is implemented in some R packages, although not in **stats**. The two most notable mentions are the the `MCEstimator()` function of the **distrMod** R package (part of the **distr** ecosystem), and the `fitdistr()` function of the **MASS** package. Other packages that can perform maximum likelihood estimation given a log-likelihood function also exists, such as **stats4** and **bbmle**. However, the purpose of **xbar** is to provide readily estimates without the need for the user to manually supply optimization functions or starting points.

`MCEstimator()` computes parameter estimates by minimizing a user-defined criterion, such as the negative log-likelihood for maximum likelihood estimation or distance measures for minimum distance estimation.

`fitdistr()` supports, at the moment of writing, the MLE for 14 common univariate distributions: beta, Cauchy, $\mathcal{X}^2$, exponential, Fisher's F, Gamma, Geometric, log-normal, logistic, negative binomial, normal, Poisson, Student's t, and Weibull. MLEs that can be explicitly derived are directly coded in the function, while non-explicit ones are numerically computed using `optim()`. The function optimized in these distributions is the log-likelihood in its general form, i.e. the sum of the log-densities, without simplifications with respect to the sufficient statistics, or the removal of constant terms. A similar approach is adopted by the `fitdist()` function of the **fitdistrplus** package, which covers maximum likelihood, quantile matching, maximum goodness-of-fit (minimum distance), and moment estimation. The advantages of **xbar** in this direction are extensively discussed in Subsection 3.2.

1.2. Purpose and Innovation

The **xbar** package provides a comprehensive set of features for mathematical and asymptotic statistics. It extends the range of available distribution families and facilitates the computation of key parametric quantities, such as moments and information-theoretic measures. The main focus of the package is parameter estimation through maximum likelihood and moment-based methods under an intuitive and efficient framework. Its goal is to encapsulate

Distribution	Class Name	Distribution	Class Name
Bernoulli	`Bern`	Laplace	`Laplace`
Beta	`Beta`	Log-Normal	`Lnorm`
Binomial	`Binom`	Multivariate Gamma	`Multigam`
Categorical	`Cat`	Multinomial	`Multinom`
Cauchy	`Cauchy`	Negative Binomial	`Nbinom`
Chi-Square	`Chisq`	Normal	`Norm`
Dirichlet	`Dir`	Poisson	`Pois`
Fisher	`Fisher`	Student	`Stud`
Gamma	`Gam`	Uniform	`Unif`
Geometric	`Geom`	Weibull	`Weib`

Table 1: Overview of the distributions implemented in the **xbar** package, along with their respective class names.

the distribution richness of the **distr6** package and enhance the capabilities of the **distr** package ecosystem in a simple framework. All package features are available both in a **stats**-like syntax for the entry-level users, and in a OOP system for more experienced ones. Overall, **xbar** addresses several limitations in the state of the art R packages in an attempt to provide the most fundamental methods of statistics in a unified package. The remainder of this paper introduces the main functionalities of the package.

2. The estim S4 Distribution System

2.1. Probability Distributions

In the **xbar** OOP system each distribution has a respective S4 class, all of which are subclasses of the `Distribution` S4 class. Table 1 shows the distributions available in the package, along with their class names.

Defining an object from the desired distribution class is straightforward, as seen in the following example. The parameter names, which are generally identical to the ones defined in the **stats** package, can be omitted.

```
R> D <- Beta(shape1 = 1, shape2 = 2)
R> D <- Beta(1, 2)
```

Having defined the distribution object `D`, the `d()-p()-q()-r()` functions can be used, as shown in the following example, comparing against the **stats** syntax.

```
R> d(D, 0.5) ; dbeta(0.5, shape1, shape2)
R> p(D, 0.5) ; pbeta(0.5, shape1, shape2)
R> qn(D, 0.75) ; qbeta(0.75, shape1, shape2)
R> r(D, 2) ; rbeta(2, shape1, shape2)
```

Alternatively, if only the distribution argument is supplied, the methods behave as functionals (i.e. they return a function). This behavior offers enhanced functionality such as:

```
R> F <- p(D) ; F(0.5)
```

Technical Detail

The quantile function is called `qn()` rather than the more intuitive `q()`. The reason behind this choice lies in the RStudio IDE (Integrated Development Environment), which overrides the method selection process of the **base** function `q()` used to quit an R session, i.e. a function named `q()` always ends the session. In order to avoid this unpleasant behavior, the name `qn()` was chosen instead.

2.2. Parametric Quantities of Interest

The **xbar** package contains a set of methods that calculate the theoretical moments (expectation, variance and standard deviation, skewness, excess kurtosis) and other important parametric functions (median, mode, entropy, Fisher information) of a distribution. Alternatively, the `moments()` function automatically finds the available methods for a given distribution and returns all of the results in a list.

```
R> mean(D)
R> median(D)
R> mode(D)
R> var(D)
R> sd(D)
R> skew(D)
R> kurt(D)
R> entro(D)
R> finf(D)
R> moments(D)
```

Technical Detail

Only the function-distribution combinations that are theoretically defined are available; for example, while `var()` is available for all distributions, `sd()` is available only for the univariate ones. In case the result is not unique, a predetermined value is returned with a warning. The following example illustrates this in the case of $\mathcal{B}(1, 1)$, i.e. a uniform distribution for which every value in the $[0, 1]$ interval is a mode.

```
R> mode(Beta(1, 1))

[1] 0.5
Warning message:
In mode(Beta(1, 1)) :
  In Beta(1, 1), all elements in the [0, 1] interval are modes.
  0.5 is returned by default.
```

3. Parameter Estimation

The **xbar** package includes a number of options when it comes to parameter estimation. In order to illustrate these alternatives, a random sample is generated from the Beta distribution.

```
R> set.seed(1)
R> shape1 <- 1
R> shape2 <- 2
R> D <- Beta(shape1, shape2)
R> x <- r(D, 100)
```

3.1. Estimation Methods

The **xbar** package covers three major estimation methods: maximum likelihood estimation (MLE), moment estimation (ME), and score-adjusted estimation (SAME).

In order to perform parameter estimation, a new `e<name>()` member is added to the `d()-p()-q()-r()` family, following the standard **stats** name convention. These `e<name>()` functions take two arguments, the observations `x` (an atomic vector for univariate or a matrix for multivariate distibutions) and the `type` of estimation method to use (a character with possible values `"mle"`, `"me"`, and `"same"`.)

```
R> ebeta(x, type = "mle")

$shape1
[1] 1.066968

$shape2
[1] 2.466715

R> ebeta(x, type = "me")

$shape1
[1] 1.074511

$shape2
[1] 2.469756

R> ebeta(x, type = "same")

$shape1
[1] 1.067768

$shape2
[1] 2.454257
```

Point estimation functions are available in two versions, the distribution specific one, e.g. `ebeta()`, and the S4 generic ones, namely `mle()`, `me()`, and `same()`. A general function called `e()` is also implemented, covering all distributions and estimators.

```
R> mle(D, x)
R> me(D, x)
R> same(D, x)
R> e(D, x, type = "mle")
```

Technical Detail

It is important to note that the S4 methods also accept a character for the distribution. The name should be the same as the S4 distribution generator, case ignored (i.e. "Beta", "beta", or "bEtA").

```
R> mle("beta", x)
R> mle("bEtA", x)
R> e("Beta", x, type = "mle")
```

3.2. Log-likelihood

Log-likelihood functions are also available in two versions, the distribution specific one, e.g. `llbeta()`, and the `ll()` S4 generic one.

```
R> llbeta(x, shape1, shape2)

[1] 26.56269

R> ll(D, x)

[1] 26.56269
```

In some distribution families like beta and gamma, the MLE cannot be explicitly derived and numerical optimization algorithms have to be employed. Even in "good" scenarios, with plenty of observations and a smooth optimization function, numerical algorithms should not be viewed as panacea, and extra care should be taken to ensure a fast and right convergence if possible. Two important steps are taken in **xbar** in this direction:

1. The log-likelihood function is analytically calculated for each distribution family, so that constant terms with respect to the parameters can be removed, leaving only the sufficient statistics as a requirement for the function evaluation.
2. Multidimensional problems are reduced to unidimensional ones by utilizing the score equations.

An illustrative example for the Beta distribution is shown below. Let f denote the probability density function of $X \sim \mathcal{B}(\alpha, \beta)$:

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad 0 < x < 1,$$

where Γ is the Gamma function. Then, the log-likelihood function, divided by the sample size n, takes the form:

$$\ell(\alpha, \beta) = (\alpha - 1)\overline{\log X} + (\beta - 1)\overline{\log(1 - X)} - \log\Gamma(\alpha) - \log\Gamma(\beta) + \log\Gamma(\alpha + \beta).$$

The score equation for α is:

$$\frac{\partial \ell}{\partial \alpha}(\alpha, \beta) = \overline{\log X} - \psi(\alpha) + \psi(\alpha + \beta) = 0.$$

The score equation for β is:

$$\frac{\partial \ell}{\partial \beta}(\alpha, \beta) = \overline{\log(1 - X)} - \psi(\beta) + \psi(\alpha + \beta) = 0.$$

These two nonlinear equations must be solved numerically. However, instead of solving the above two-dimensional problem, one can see that by denoting $c := \alpha + \beta$, the two score equations can be rewritten as:

$$\alpha = \psi^{-1}\left[\psi(c) + \overline{\log X}\right] \quad \beta = \psi^{-1}\left[\psi(c) + \overline{\log(1 - X)}\right],$$

i.e. restricted to the score equation system solution space, both parameters can be expressed as a function of their sum c, and therefore the log-likelihood function can be optimized with respect to c:

$$\ell^{\star}(c) = [\alpha(c) - 1]\,\overline{\log X} + [\beta(c) - 1]\,\overline{\log(1 - X)} - \log\Gamma\left[\alpha(c)\right] - \log\Gamma\left[\beta(c)\right] + \log\Gamma(c).$$

Technical Detail

It would perhaps be more intuitive to use the score equations to express α as a function of β or vice versa. However, the above method can be directly generalized to the Dirichlet case and reduce the initial k-dimensional problem to a unidimensional one. The same technique can be utilized for the gamma and multivariate gamma distribution families, also reducing the dimension to unity, from 2 and $k + 1$ respectively.

In **xbar**, the resulting function that is inserted in the optimization algorithm is called `lloptim()`, and is not to be confused with the actual log-likelihood function `ll()`. The corresponding derivative is called `dlloptim()`. Therefore, whenever numerical computation of the MLE is required, **xbar** calls the `optim()` function with the following arguments:

- `lloptim()`, an efficient function to be optimized,
- `dlloptim()`, its analytically-computed derivate,
- the ME or SAME as the starting point (user's choice),
- the L-BFGS-U optimization algorithm, with lower and upper limits defined by default as the parameter space boundary.

3.3. Asymptotic Variance - Covariance Matrix

The asymptotic variance (or variance - covariance matrix for multidimensional parameters) of the estimators are also covered in the package. As with point estimation, the implementation is twofold, distribution specific (`vbeta()`) and S4 generic. In the first case, the `type` argument can be used to specify the estimator type. The general function `avar()` covers all distributions and estimators.

```
R> vbeta(shape1, shape2, type = "mle")

         shape1   shape2
shape1 1.597168 2.523104
shape2 2.523104 7.985838

R> vbeta(shape1, shape2, type = "me")

       shape1 shape2
shape1    2.1    3.3
shape2    3.3    9.3

R> vbeta(shape1, shape2, type = "same")

         shape1   shape2
shape1 1.644934 2.539868
shape2 2.539868 8.079736

R> avar_mle(D)
R> avar_me(D)
R> avar_same(D)
R> avar(D, type = "mle")
```

4. Estimation Metrics and Comparison

The different estimators of a parameter can be compared based on both finite sample and asymptotic properties. The package includes two functions named `small_metrics()` and `large_metrics()`, where small and large refers to the "small sample" and "large sample" terms that are often used for the two cases. The former estimates the bias, variance and root mean square error (RMSE) of the estimator with Monte Carlo simulations, while the latter calculates the asymptotic variance - covariance matrix (as derived by the `avar` functions). The resulting data frames can be plotted with the functions `plot_small_metrics()` and `plot_large_metrics()`, respectively.

To illustrate the function's design, consider the following example from the beta distribution: We are interested to calculate the metrics (bias, variance, and RMSE) of the α parameter estimators (MLE, ME, and SAME), for sample sizes 20 and 50. Specifically, we want to illustrate how these metrics change for $\alpha \in [1, 5]$, and $\beta = 2$ (constant). The following code can do that:

```
R> D <- Beta(1, 2)
R> prm <- list(name = "shape1",
+              val = seq(1, 5, by = 0.5))
R> x <- small_metrics(D, prm,
+               obs = c(20, 50),
+               est = c("mle", "same", "me"),
+               sam = 1e3,
+               seed = 1)
R> class(x)

[1] "SmallMetrics"
attr(,"package")
[1] "estim"

R> head(x@df)

  Parameter Observations Estimator Metric     Value
1       1.0           20       mle   Bias 0.1322510
2       1.5           20       mle   Bias 0.2486026
3       2.0           20       mle   Bias 0.3276922
4       2.5           20       mle   Bias 0.5215973
5       3.0           20       mle   Bias 0.5517199
6       3.5           20       mle   Bias 0.5381523
```

The `small_metrics()` function takes the following arguments:

- `D`, the distribution object of interest,
- `prm`, a list that specifies how the `shape1` parameter values should change,
- `obs`, a numeric vector holding the sample sizes,
- `est`, a character vector specifying the estimators under comparison,
- `sam`, the Monte Carlo sample size to use for the metrics estimation,
- `seed`, a seed to be passed to `set.seed()` for replicability.

The resulting data frame can be passed to `plot()` to see the results. This `plot()` method depends on **ggplot2** to provide a highly-customizable graph.

```
plot(x)
```

Note that in some distribution families the parameter is a vector, as is the case with the Dirichlet distribution (a multivariate generalization of beta), which holds a single parameter vector `alpha`. In these cases, the `prm` list can include a third element, `pos`, specifying which parameter of the vector should change:

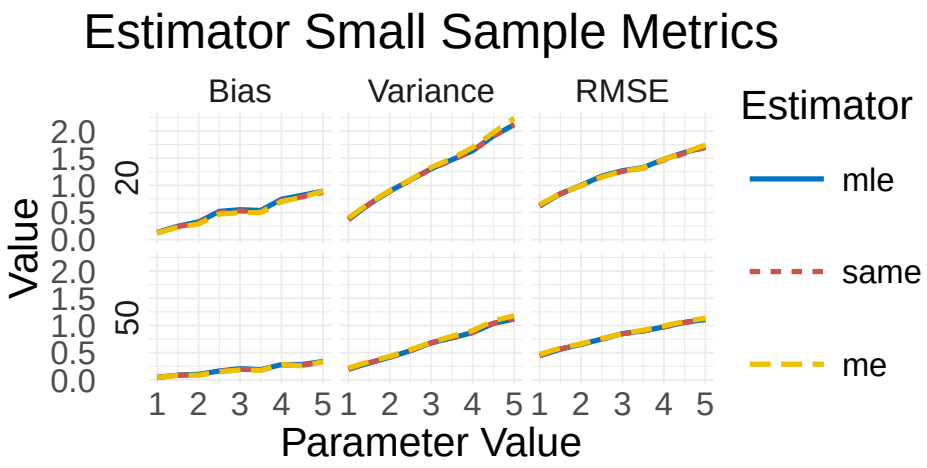

Figure 1: Small-sample metrics comparison for MLE, ME, and SAME of the beta distribution α parameter.

```
R> D <- Dir(alpha = 1:4)
R> prm <- list(name = "alpha",
+              pos = 1,
+              val = seq(1, 5, by = 0.5))
R> x <- small_metrics(D, prm,
+                     obs = c(20, 50),
+                     est = c("mle", "same", "me"),
+                     sam = 1e3,
+                     seed = 1)
R> class(x)

[1] "SmallMetrics"
attr(,"package")
[1] "estim"

R> head(x@df)

  Parameter Observations Estimator Metric      Value
1       1.0           20       mle   Bias 0.07759434
2       1.5           20       mle   Bias 0.11916042
3       2.0           20       mle   Bias 0.17488071
4       2.5           20       mle   Bias 0.20523185
5       3.0           20       mle   Bias 0.32907679
6       3.5           20       mle   Bias 0.32915767
```

The `large_metrics()` function design is almost identical, except that no `obs`, `sam`, and `seed` arguments are needed here. The following example illustrates the large sample metrics for the beta distribution shape α estimators. Again, the resulting data frame can be passed to `plot()`.

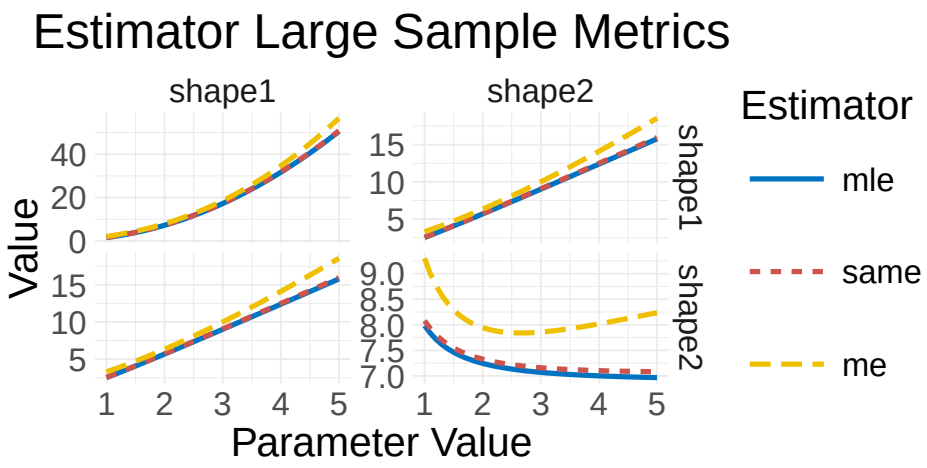

Figure 2: Large-sample metrics comparison for MLE, ME, and SAME of the beta distribution α parameter.

```
plot(x)
```

```
R> D <- Beta(1, 2)
R> prm <- list(name = "shape1",
+               val = seq(1, 5, by = 0.1))
R> x <- large_metrics(D, prm,
+                     est = c("mle", "same", "me"))
R> class(x)
```

```
[1] "LargeMetrics"
attr(,"package")
[1] "estim"
```

```
R> head(x@df)
```

```
     Row    Col Parameter Estimator    Value
1 shape1 shape1       1.0       mle 1.597168
2 shape2 shape1       1.0       mle 2.523104
3 shape1 shape2       1.0       mle 2.523104
4 shape2 shape2       1.0       mle 7.985838
5 shape1 shape1       1.1       mle 1.969699
6 shape2 shape1       1.1       mle 2.826906
```

5. Documentation and Checks

5.1. Documentation

5.2. Testing

The R package is rigorously tested to ensure reliability, correctness, and stability. More than 1,000 automated tests have been implemented using the **testthat** package, a widely used framework for unit testing in R. These tests cover a broad range of functionalities, including edge cases, error handling, and performance checks, to verify that every function behaves as expected under various conditions. Continuous testing helps detect potential regressions early, maintaining the integrity of the package as it evolves. By leveraging **testthat**, we ensure that all updates and modifications uphold the expected behavior and performance standards.

6. Defining New Classes and Methods

Of course, it is possible to be interested in a distribution family not included in the package. It is straightforward for users to define their own S4 class and methods. Since this paper is addressed to both novice and experienced R users, the beta distribution paradigm is explained in detail below:

6.1. Defining the Class

The `setClass()` function defines a new S4 class, i.e. the distribution of interest. The `slots` argument defines the parameters and their respective class (usually numeric, but it can also be a matrix in distributions like the multivariate normal and the Wishart). The optional argument `prototype` can be used to define the default parameter values in case they are not specified by the user.

```
setClass("Beta",
  contains = "Distribution",
  slots = c(shape1 = "numeric", shape2 = "numeric"),
  prototype = list(shape1 = 1, shape2 = 1))
```

6.2. Defining a Generator

Now that the class is defined, one can type `D <- new("Beta", shape1 = shape1, shape2 = shape2)` to create a new object of class `Beta`. However, this is not so intuitive, and a wrapper function with the class name can be used instead. This function, often called a "generator", can be used to simply code `D <- Beta(1, 2)` and define a new object from the $\mathcal{B}(1, 2)$ distribution. The parameter slots can be accessed with the `@` sign, as shown above.

```
Beta <- function(shape1 = 1, shape2 = 1) {
  new("Beta", shape1 = shape1, shape2 = shape2)
}

D <- Beta(1, 2)
D@shape1 ; D@shape2
```

6.3. Defining Validity Checks

This step is optional but rather essential. So far, a user could type `D <- Beta(-1, 2)` without any errors, even though the beta parameters are defined in $\mathbb{R}_+$. To prevent such behaviors (that will probably end in bugs further down the road), the developer is advised to create a `setValidity()` function, including all the necessary restrictions posed by the parameter space.

```
setValidity("Beta", function(object) {
  if(length(object@shape1) != 1) {
    stop("shape1 has to be a numeric of length 1")
  }
  if(object@shape1 <= 0) {
    stop("shape1 has to be positive")
  }
  if(length(object@shape2) != 1) {
    stop("shape2 has to be a numeric of length 1")
  }
  if(object@shape2 <= 0) {
    stop("shape2 has to be positive")
  }
  TRUE
})
```

6.4. Defining the Class Methods

Now that everything is set, it is time to define methods for the new class. Creating functions and S4 methods in R are two very similar processes, except the latter wraps the function in `setMethod()` and specifies a signature class, as shown above. The package source code can be used to easily define all methods of interest for the new distribution class.

```
# probability density function
setMethod("d", signature = c(distr = "Beta", x = "numeric"),
          function(distr, x) {
            dbeta(x, shape1 = distr@shape1, shape2 = distr@shape2)
          })

# (theoretical) expectation
setMethod("mean",
          signature  = c(x = "Beta"),
          definition = function(x) {

  x@shape1 / (x@shape1 + x@shape2)

})

# moment estimator
```

```
setMethod("me",
          signature  = c(distr = "Beta", x = "numeric"),
          definition = function(distr, x) {

  m  <- mean(x)
  m2 <- mean(x ^ 2)
  d  <- (m - m2) / (m2 - m ^ 2)

  c(shape1 = d * m, shape2 = d * (1 - m))

})
```

7. Discussion

7.1. Advantages

7.2. Limitations

7.3. Perspectives

8. Conclusion

Computational details

The results in this paper were obtained using R 4.4.1 with the **xbar** 0.11.2 package. R itself and all packages used are available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgments

The authors would like to thank the editor and the anonymous reviewers for their valuable suggestions. Ioannis Oikonomidis would like to thank the Fanourakis Foundation for supporting his PhD studies.

References

Ruckdeschel P, Kohl M (2014). "General Purpose Convolution Algorithm in S4 Classes by Means of FFT." *Journal of Statistical Software*, **59**(4), 1–25. doi:10.18637/jss.v059.i04.

Ruckdeschel P, Kohl M, Stabla T, Camphausen F (2006). "S4 Classes for Distributions." *R News*, **6**(2), 2–6.

Sonabend R, Kiraly F (2025). *distr6: The Complete R6 Probability Distributions Interface*. R package version 1.8.4, commit a642cd312c51fba7ede7336036f907dbe279889e, URL https://github.com/xoopR/distr6.

Affiliation:

Ioannis Oikonomidis
Department of Mathematics
National and Kapodistrian University of Athens
15874 Zografos, Greece E-mail: goikon@math.uoa.gr
URL: users.uoa.gr/~goikon