Journal of Statistical Software

February 2016, Volume 69, Issue 3. *doi: 10.18637/jss.v069.i03*

PoweR: A Reproducible Research Tool to Ease Monte Carlo Power Simulation Studies for Goodness-of-fit Tests in R

Pierre Lafaye de Micheaux
Université de Montréal, CREST - ENSAI

Viet Anh Tran
Université de Montréal

Abstract

The **PoweR** package aims to help obtain or verify empirical power studies for goodness-of-fit tests for independent and identically distributed data. The current version of our package is only valid for simple null hypotheses or for pivotal test statistics for which the set of critical values does not depend on a particular choice of a null distribution (and on nuisance parameters) under the non-simple null case. We also assume that the distribution of the test statistic is continuous. As a reproducible research computational tool it can be viewed as helping to simply reproducing (or detecting errors in) simulation results already published in the literature. Using our package helps also in designing new simulation studies. The empirical levels and powers for many statistical test statistics under a wide variety of alternative distributions can be obtained quickly and accurately using a C/C++ and R environment. The **parallel** package can be used to parallelize computations when a multicore processor is available. The results can be displayed using LaTeX tables or specialized graphs, which can be directly incorporated into a report. This article gives an overview of the main design aims and principles of our package, as well as strategies for adaptation and extension. Hands-on illustrations are presented to help new users in getting started.

Keywords: reproducible research, Monte Carlo, goodness-of-fit test, power study, R.

1. Introduction

Reproducible research, a philosophy of research first promoted by Claerbout and Karrenbach (1992), is a way of providing the reader of a document the possibility of reproducing all of its graphical and numerical content. This implies access to all the data, codes and software to run it, as well as instructions to use it properly. Note that this means that users can easily produce similar results from their own data. In the specific field of research on goodness-of-fit tests, most published papers contain a section including Monte Carlo simulation results

showing performance of several competing tests in terms of size and power. These results are usually presented more or less in the form of voluminous tables. To obtain or reproduce such results *is* a fastidious and time-consuming operation, as one usually has to program in one's favorite language all the test statistics involved, devise a plan for the simulations (including all the chosen alternative distributions), run those simulations and integrate the results in a LaTeX table. These operations can take weeks (see e.g., Romão, Delgado, and Costa (2010) for a comprehensive study of non-normality tests) and errors are difficult to avoid. To circumvent these problems, we propose the following guidelines for reproducible research:

1. always provide an explicit (mathematical) formula or procedure to compute the test statistic that has been developed;
2. apply the test on a small real (or simulated) data set and provide the data, the value of the test statistic and the p value;
3. if possible, give a pseudocode description of the algorithm used to compute the test statistic, its critical values and p values (see Kreher and Stinson (2005) for an appropriate LaTeX package);
4. give clear indications or references for the competing tests and alternative distributions used in the simulations, including the values of all the parameters involved;
5. assuming familiarity with the R and C/C++ languages, integrate the code of your functions (with comments) into our package **PoweR** and use it to perform the simulations, then in your publication give the set of instructions used.

As per these guidelines, we advocate the use of our new package **PoweR**. This is designed to help obtain or verify empirical power studies for (goodness-of-fit) testing of a null hypothesis that the (independent and identically distributed) simulated data comes from some specified distribution. Its main characteristics are:

- generation of values from many probability distributions;
- computation of several goodness-of-fit test statistics;
- Monte Carlo computation of p values;
- functions to compute the empirical size and power of several hypothesis tests under various distributions;
- various plot functions, described later, to ease graphical comparisons between tests;
- C/C++ implementation of several parts of the code for faster computations;
- optimized management of the random generated sampled values;
- possibility to use parallel computations, using the **parallel** package (R Core Team 2015), if your computer is equipped with several CPUs or with multicore processors;
- output of LaTeX tables that can be directly incorporated into your document;
- graphical user interface (GUI).

However, we warn a potential user that the current version of our package is only valid for simple null hypotheses or for pivotal test statistics for which the set of critical values does not depend on a particular choice of a null distribution (and on nuisance parameters) under the non-simple null case (but see Remark 2.2).

In the next section, we recall some results of computational theory of various statistical quantities using Monte Carlo simulations, and recall the terminology used when one performs a goodness-of-fit test simulation study. We also mention the probability distributions and the hypothesis tests that are already included in the package. In Section 3, we show how to use our package – via a script or via the GUI – to (re)produce a simulated comparison of powers for already existing tests. In Section 4, we show how to extend the package to add a new law or a newly created test. We then conclude the paper by giving future avenues of development. Note that hereon, we will suppose that our package **PoweR** has been loaded into R memory using the instruction `library("PoweR")`. All the computations presented in this paper have been performed on a laptop under the Linux Debian 8 operating system, equipped with a 64 bit eight-cores Intel(R) Core(TM) i7 CPU 960 at 3.20GHz.

2. Monte Carlo simulations for goodness-of-fit tests

2.1. Theoretical background on hypothesis tests

Suppose that we have a sample $X_1, \ldots, X_n$ of random variables for which the distribution is assumed to belong to some parametric family $\mathcal{F} = \{\mathcal{P}(\boldsymbol{\theta}); \mathcal{P} \in \mathcal{D}, \boldsymbol{\theta} \in \Theta_{\mathcal{P}}\}$, where $\mathcal{D}$ is some subset of all the probability distributions and $\Theta_{\mathcal{P}} \subset \mathbb{R}^{k_{\mathcal{P}}}$, $k_{\mathcal{P}} \in \mathbb{N}$. Goodness-of-fit tests are procedures designed to evaluate the following statistical hypotheses involving the true probability distribution $\mathcal{L}(X_i)$ of X_i:

$$\mathcal{H}_0 : \mathcal{L}(X_i) \in \mathcal{A} = \{\mathcal{P}_0(\boldsymbol{\theta}); \boldsymbol{\theta} \in \Theta_0 \subset \Theta_{\mathcal{P}_0}\} \quad \text{versus} \quad \mathcal{H}_1 : \mathcal{L}(X_i) \in \mathcal{F} \backslash \mathcal{A},$$

where $\mathcal{P}_0(\boldsymbol{\theta}) \in \mathcal{F}$ is called a *null distribution*. If the true distribution of the sample does not belong to $\mathcal{A}$, we call it an *alternative distribution*. For example, one might be interested in testing non-normality, in which case $\mathcal{P}_0(\boldsymbol{\theta}) = \mathcal{N}(\mu, \sigma^2)$, the Gaussian distribution with parameters $\boldsymbol{\theta} = (\mu, \sigma^2)$. The truthfulness of hypothesis $\mathcal{H}_0$ is questioned, at a pre-specified *significance level* α, using a test statistic $T_n = T(X_1, \ldots, X_n)$ built so as to reflect a discrepancy between the null hypothesis and the information brought by the data. Its observed value t_{obs} is usually compared with one (c_α) or two ($c_L(\alpha)$ and $c_R(\alpha)$) *critical values* (also called *percentage points*) defining the so-called rejection (of $\mathcal{H}_0$) region $\mathcal{R}_\alpha$. That is we decide to reject $\mathcal{H}_0$ (declare it is false) if and only if $T_n \in \mathcal{R}_\alpha$. In this paper, we shall only consider rejection regions $\mathcal{R}_\alpha$ of the form $\{T_n > c_\alpha\}$, $\{T_n < c_\alpha\}$ or $\{T_n < c_L(\alpha)\} \cup \{T_n > c_R(\alpha)\}$.

If $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$, where $\mathcal{F}_1$ and $\mathcal{F}_2$ are two nonempty disjoint families of distributions (e.g., sub-Gaussian and super-Gaussian distributions), and if the alternative hypothesis is $\mathcal{H}_1 : \mathcal{L}(X_i) \in \mathcal{F}_1 \backslash \mathcal{A}$ (instead of $\mathcal{H}_1 : \mathcal{L}(X_i) \in \mathcal{F} \backslash \mathcal{A}$), the test will be termed *one-tailed* or *one-sided*; otherwise, it will be called *two-tailed* or *two-sided*.

Two types of errors are commonly associated with any test procedure. A Type-I error occurs if we wrongly reject a true null hypothesis and a Type-II error if we fail to reject a false null hypothesis. A test procedure is designed to control, with some threshold level (the *significance level* α), the probability of committing a Type-I error. A critical value(s) is (are) needed to

define the rejection region, and is (are) chosen so that $\mathsf{P}_{\mathcal{H}_0}[T_n \in \mathcal{R}_\alpha] \leq \alpha$ (but as close to α as possible). The probability of a Type-I error $\mathsf{P}_{\mathcal{H}_0}[T_n \in \mathcal{R}_\alpha]$ is called the *size* of the test and it will be important to check if it is really less than α. Note that an evaluation of this size will be possible numerically using our package; the value obtained being called the *empirical size* (or sometimes *empirical level*) of the test. Note also that sometimes the critical values defining the rejection region are computed based on the asymptotic distribution of T_n. Therefore, if T_∞ denotes any random variable following this asymptotic distribution, we choose critical values such that $\mathsf{P}_{\mathcal{H}_0}[T_\infty \in \mathcal{R}_\alpha] \leq \alpha$. These *asymptotic critical values* will be denoted c_α^a, $c_L^a(\alpha)$ and $c_R^a(\alpha)$.

Several non-normality tests are available in the **nortest** package (Gross and bug fixes by Uwe Ligges 2012). A question naturally arises for any user of this package: which test should they use? The answer depends on the power $1-\beta(\boldsymbol{\theta})$ of the test statistic $T_n = T(X_1, \ldots, X_n)$ considered, which is given by

$$1 - \beta(\boldsymbol{\theta}) = \mathsf{P}_{\mathcal{H}_1}[T_n \in \mathcal{R}_\alpha],$$

where $\beta(\boldsymbol{\theta})$ is the probability of committing a Type-II error and $\mathcal{R}_\alpha$ is the α-rejection region of the test procedure, determined beforehand as described above. For a given significance level, the test with the greatest power (for a given value of $\boldsymbol{\theta}$) should be used. At this point, one should note that given real data, the power of any test procedure is unknown because it depends on the unknown true distribution of the sample at hand. This is why it is important to perform simulations to compare numerically several hypothesis tests (designed to test the same null hypothesis) numerically under a wide variety of common alternative distributions. This is when our package **PoweR** becomes handy, as it offers a fast and automated way to perform such computations using Monte Carlo simulations.

Another important characteristic attached to the application of a hypothesis test on real or simulated data is the so-called *p value*, informally defined as the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. Note that *two-sided statistical p values* (those computed when $\mathcal{R}_\alpha = \{T_n < c_L(\alpha)\} \cup \{T_n > c_R(\alpha)\}$) are well defined (as $2\mathsf{P}_{\mathcal{H}_0}[T_n > |t_{obs}|]$) only when the test statistic considered has a symmetric distribution. For non-symmetric distributions, several proposals have been made (Kulinskaya 2008) but none of them led to a consensus. In this paper, we shall only consider the Fisher's doubled p value (Yates 1984, p. 444), even given the evident drawback of this doubling rule that it may result in a p value greater than 1 (The rule doubles the one-sided p value coming from the tail corresponding to the observed direction, which is taken to be the lower one if $t_{obs} < q_2$ and the upper one otherwise, where q_2 is the median of T_n or of T_∞). Fisher's motivation was an equal prior weight of departure in either direction. Future versions of our package might propose other approaches.

2.2. Monte Carlo computations

All the characteristics defined so far (critical values, size and power of the test, p value), can be estimated numerically using Monte Carlo simulations. A typical algorithm is given below.

Require: *n (Sample size)*
Require: *M (Number of Monte Carlo samples)*
Require: $\mathcal{P}$ *(A probability distribution from which to generate numbers)*
Require: $\boldsymbol{\theta}$ *(Vector of parameters of* $\mathcal{P}$*)*

for $m = 1, \ldots, M$ **do**

- generate independently a sample $\boldsymbol{x}_m = (x_{1,m}, \ldots, x_{n,m})$ of size n drawn at random from a distribution $\mathcal{P}(\boldsymbol{\theta}) \in \mathcal{F}$ for some fixed value $\boldsymbol{\theta}$ of the parameter vector;
- compute the observed value of the test statistic $t_{n,m} = T(x_{1,m}, \ldots, x_{n,m})$.

end for
return

We can then use $t_{n,1}, \ldots, t_{n,M}$ to estimate quantities of interest. Let us consider a two-sided test that rejects for either large or small values of the test statistic. If $\mathcal{P} = \mathcal{P}_0$ (we are under the null), we could compute (using our function `many.crit()`) the critical values

$$\hat{c}_L(\alpha) = t_{n,(\lceil M\alpha/2 \rceil)} \quad \text{and} \quad \hat{c}_R(\alpha) = t_{n,(\lceil M(1-\alpha/2) \rceil)}$$

which are respectively the $(\alpha/2)^{th}$ and $(1-\alpha/2)^{th}$ empirical quantiles, where $\lceil x \rceil$ denotes the smallest integer not less than x. Note that in the current version of our package, we have assumed a rejection of the null with equal probability for large or small values of the test statistic. The power of the test under some alternative distribution $\mathcal{P}(\boldsymbol{\theta})$ can then be estimated by

$$1 - \hat{\beta}(\boldsymbol{\theta}) = \frac{\#\Big\{t_{n,m}; 1 \le m \le M, t_{n,m} < \hat{c}_L(\alpha) \text{ or } t_{n,m} > \hat{c}_R(\alpha)\Big\}}{M},$$

where the $t_{n,m}$ are here computed under the alternative distribution specified by $\boldsymbol{\theta}$. If $\mathcal{P} = \mathcal{P}_0$ and we have given values of $c_L(\alpha)$ and $c_R(\alpha)$, for example the (theoretical) asymptotical critical values $c_L^a(\alpha)$ and $c_R^a(\alpha)$, we can compute the empirical size of the test by

$$\frac{\#\Big\{t_{n,m}; 1 \le m \le M, t_{n,m} < c_L(\alpha) \text{ or } t_{n,m} > c_R(\alpha)\Big\}}{M}.$$

Suppose one has an observed value t_{obs} of some test statistic T_n used to evaluate some goodness-of-fit hypothesis $\mathcal{H}_0$. The p value can be approximated (by $\hat{p}$ given in Equation 1) using the following procedure. First, randomly generate M samples under the null from which you compute M values of the test statistic $t_{n,1}, \ldots, t_{n,M}$. Next compute

$$\hat{p} = \begin{cases} 2 \times \frac{\#\{t_{n,m}; 1 \le m \le M, t_{n,m} > t_{obs}\}}{M} & \text{if } t_{obs} > \hat{q}_{0.5}; \\ 2 \times \frac{\#\{t_{n,m}; 1 \le m \le M, t_{n,m} \le t_{obs}\}}{M} & \text{otherwise,} \end{cases} \tag{1}$$

where $\hat{q}_{0.5}$ is the empirical median of $t_{n,1}, \ldots, t_{n,M}$.

Remark 2.1. Let $p \in (0,1)$ and let $\{t_{(n,1)}, \ldots, t_{(n,M)}\}$ be a set of i.i.d. generated realizations of some continuous test statistic T_n arranged in ascending order. An asymptotic (in terms of the number M of Monte Carlo samples) $(1-\alpha)$-confidence interval for the p-th quantile $x_{n,p}$ of T_n (i.e., $\mathsf{P}(T_n \leq x_{n,p}) = p$) is given by

$$\left[t_{(n,i_M)}, t_{(n,j_M)}\right]$$

where $i_M = \lfloor \frac{1}{2} + Mp - \sqrt{M} z_{1-\alpha/2} \sqrt{p(1-p)} \rfloor$ and $j_M = \lfloor \frac{1}{2} + Mp + \sqrt{M} z_{1-\alpha/2} \sqrt{p(1-p)} \rfloor$, $1 \leq i_M, j_M \leq M$, and where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$-th quantile of a $\mathcal{N}(0,1)$ distribution. This approach can be used to obtain confidence intervals for the Monte Carlo estimators of the critical values presented above (i.e., for $\hat{c}_L$ and $\hat{c}_R$).

Similarly, a $(1-\alpha)$-confidence interval for the estimated power $1-\hat{\beta}(\boldsymbol{\theta})$ is given by (normal approximation)

$$\left[1 - \hat{\beta}(\boldsymbol{\theta}) \pm z_{1-\alpha/2} \sqrt{\frac{\hat{\beta}(\boldsymbol{\theta})(1-\hat{\beta}(\boldsymbol{\theta}))}{M}}\right]$$

or by (Wilson 1927's improvement)

$$\frac{1}{1+\frac{z_{1-\alpha/2}^2}{M}} \left[1 - \hat{\beta}(\boldsymbol{\theta}) + \frac{z_{1-\alpha/2}^2}{2M} \pm z_{1-\alpha/2} \sqrt{\frac{\hat{\beta}(\boldsymbol{\theta})(1-\hat{\beta}(\boldsymbol{\theta}))}{M} + \frac{z_{1-\alpha/2}^2}{4M^2}}\right].$$

2.3. Presenting results

Presenting Monte Carlo results to show evidence of the finite-sample properties of hypothesis testing procedures usually relies on tables of the (empirical) size and power of these hypothesis tests. One can also use simple techniques for the graphical display of simulation evidence. Our package **PoweR** render these tasks easily. The reader is encouraged to read Ehrenberg (1977) which contains some precepts for improving data presentations. Davidson and MacKinnon (1998) describe three types of figures, called *p value plots*, *p value discrepancy plots* and *size-power curves* that convey much more information, in a more easily assimilated form, than tables; see also Koziol (1989), Lieber (1990), Schweder and Spjøtvoll (1982) and Wilk and Gnanadesikan (1968).

All of these graphs, presented below, are based on the empirical distribution function (EDF) of the p values of the test statistic considered. If we have a generated sample $p_1, \ldots, p_N$ of p values derived from observed values of a certain test statistic T_n (under a given distribution), then the associated EDF is computed using the following formula:

$$\hat{F}(x) = \frac{1}{N} \sum_{\ell=1}^{N} \mathbb{1}\{p_\ell \leq x\} \quad \forall x \in (0,1), \tag{2}$$

where $\mathbb{1}\{C\}$ equals 1 if condition C is satisfied, and 0 otherwise. This EDF can be evaluated at J points $x_j, j = 1, \ldots, J$ which should be chosen in advance so as to provide a reasonable snapshot of the $(0,1)$ interval, or of that part of it which is of interest. A quite parsimonious way to choose the x_j is

$$x_j = 0.001,\ 0.002, \ldots,\ 0.010,\ 0.015, \ldots,\ 0.990,\ 0.991, \ldots,\ 0.999 \quad (J = 215). \tag{3}$$

Extra points can be added near 0 and 1 to ensure that we do not miss any unusual behavior in the tails.

Two plots designed to evaluate (and compare) the size of one or several test statistics under the null hypothesis are:

• **The p value plot:** this is a plot of $\hat{F}(x_j)$ against x_j, $j = 1, \ldots, J$. If the distribution of T_n used to compute the p_i's is correct, each of the p_i should be the realization of a uniform distribution on $(0, 1)$. Therefore, when $\hat{F}(x_j)$ is plotted against x_j, the resulting graph should be close to the 45° line. We can then easily distinguish at a glance between test statistics that systematically over-reject, under-reject, or reject in roughly the right proportion of times.

• **The p value discrepancy plot:** this is a plot of $\hat{F}(x_j) - x_j$ against x_j. This plot conveys a lot more information than p value plots for test statistics that are well behaved. However, some of this information is spurious, simply reflecting experimental randomness. Davidson and MacKinnon (1998, Section 5) therefore discuss semi-parametric methods for smoothing them. Moreover, because there is no natural scale for the vertical axis, p value discrepancy plots can be harder to interpret than p value plots.

The nominal (i.e., approximated) size of a test is often different from the true size $\mathsf{P}_{\mathcal{H}_0}[T_n \in \mathcal{R}_\alpha]$, for example when it is computed using some approximation (e.g., asymptotic) of the (finite sample) null-distribution of T_n. Because the previous graphs are used to evaluate the correctness of the size, it is important to note that the p values under the null should be calculated using this approximate distribution.

If we want to compare the power of competing tests, we can use the following graph which has the advantage of being useable even if all the tests do not have the correct size. In this case, the p values under the null can be calculated using a Monte Carlo approximation of the null distribution.

• **The size-power curves:** the points $(\hat{F}(x_j), \hat{F}^*(x_j))$, $j = 1, \ldots, J$, including the points $(0, 0)$ and $(1, 1)$, where $\hat{F}(x)$ and $\hat{F}^*(x)$ are respectively the EDF of the p values under the null and under the alternative distribution; see Wilk and Gnanadesikan (1968) for a description of such a plot and Bhattacharya and Habtzghi (2002) for a definition of the p value as a random variable.

We define three other quantities, given a table of powers p_{ik} $(1 \leq i \leq I, 1 \leq k \leq K)$ of test statistics $T_1, \ldots, T_K$ against I alternative distributions. For a given sample size n and significance level α, the *average power* for test statistic T_k is defined as

$$\frac{1}{I} \sum_{i=1}^{I} p_{ik},$$

the *average gap power* is defined as

$$\frac{1}{I} \sum_{i=1}^{I} \left| p_{ik} - \max_{1 \leq k \leq K} (p_{ik}) \right|$$

and the *worst gap power* is defined as

$$\max_{1 \leq i \leq I} \left| p_{ik} - \max_{1 \leq k \leq K} (p_{ik}) \right|.$$

Remark 2.2. Unless the null is a simple one, there will be an infinite number of data generating processes (DGPs) that satisfy it. If the test statistic is pivotal, it does not matter which one we use to generate $\hat{F}(x)$. However, if it is not pivotal (such as the standard Kolmogorov-Smirnov test when the parameters are estimated), the choice of which DGP to use can matter greatly. Davidson and MacKinnon (1996) argue that a reasonable choice is the pseudo-true null, which is the DGP that satisfies the null hypothesis and is as close as possible, in the sense of the Kullback-Leibler information criterion, to the DGP used to generate $\hat{F}^*(x)$. This approach has not been implemented in our package and the choice of which DGP to use is left to the responsibility of the reader.

3. Using PoweR

3.1. Preliminaries

The instruction `help(package = "PoweR")` returns a list of all the functions available in package **PoweR**. These are listed in Table 1 below.

3.2. Distributions in PoweR

The **PoweR** package contains a large set of probability distributions (39 at the moment of writing this paper) from which to generate random numbers. These are detailed in Section A in the Appendix. Note that this information is also available from within R, as illustrated below.

```
R> head(getindex()$mat.laws)
```

```
  Index                Law Nbparams Default1 Default2 Default3 Default4
1     1      Laplace(mu,b)        2        0        1       NA       NA
2     2   Normal(mu,sigma)        2        0        1       NA       NA
3     3   Cauchy(mu,sigma)        2        0        1       NA       NA
4     4 Logistic(mu,sigma)        2        0        1       NA       NA
5     5  Gamma(shape,rate)        2        2        1       NA       NA
6     6           Beta(a,b)        2        1        1       NA       NA
```

The columns `Default1` to `Default4` display the default values of the parameters of each corresponding probability distribution in the `Law` column (e.g., `mu` $= 0$ and `b` $= 1$ for `Laplace(mu, b)`). In this context, the useful function `help.law()` enables one to obtain various information about a given law. For example, try `help.law(6)` to display documentation about the law whose index is 6, namely the Beta(a,b) distribution.

The function `gensample()` enables one to generate a sample from any distribution in the package. Try this to generate a random sample of size 10 from a $\mathcal{N}(\mu, \sigma)$ distribution with $\mu = 1$ and $\sigma = 2$:

```
R> gensample(law.index = 2, n = 10, law.pars = c(1, 2))
```

Name	Description
`calcFx()`	Empirical distribution function of p values
`checklaw()`	Check proper behavior of a random generator
`compquant()`	Computation of the quantile values only for one test statistic
`create.alter()`	Return automatically a named list of alter values (type) for test statistics
`gensample()`	Generate random samples from a law added in the package
`getindex()`	Retrieve indices of laws and test statistics
`getnbparlaws()`	Retrieve the default number of parameters of some laws
`getnbparstats()`	Get numbers of parameters of test statistics
`graph()`	p value plots, p value discrepancy plots and size-power curves
`help.law()`	Open documentation for a given law using its index
`help.stat()`	Open documentation for a test using its index
`law.cstr()`	Gives information about a given law
`many.crit()`	Compute critical values for several test statistics, sample sizes and significance levels, for a given law
`many.pval()`	Computation of p values for several test statistics
`plot.discrepancy()`	p value discrepancy plots
`plot.pvalue()`	p value plots
`plot.sizepower()`	Size-power curves
`powcomp.easy()`	Computation of power and level tables for hypothesis tests ("slow" but easy to use version)
`powcomp.fast()`	Computation of power and level tables for hypothesis tests, for several sample sizes and significance levels ("fast" version)
`power.gui()`	Graphical user interfaces (GUI)
`print.critvalues()`	Transform the critical values given by function `many.crit()` into LaTeX code to build a table of critical values
`print.power()`	Transform the power values given by function `powcomp.fast()` into LaTeX code to build a table of power values
`pvalueMC()`	Computation of one p value for one test statistic by means of Monte Carlo simulations
`stat.cstr()`	Gives information about a test statistic
`statcompute()`	Perform one of the hypothesis tests available in the package

Table 1: Functions from package **PoweR**.

```
$sample
 [1] -0.2529076  1.3672866 -0.6712572  4.1905616  1.6590155 -0.6409368
 [7]  1.9748581  2.4766494  2.1515627  0.3892232

$law
[1] "Normal(mu,sigma)"

$law.pars
[1] 1 2
```

3.3. Goodness-of-fit test statistics in PoweR

The **PoweR** package contains many functions to test non-normality, non-uniformity and non-laplacity. These are given in Tables 4, 6 and 5 of the Appendix respectively.

The instruction `getindex()$mat.stats` returns a `data.frame` with four columns listing these test statistics. The first one (`Index`) contains the indices of all the test statistics available in **PoweR**. The second one (`Stat`) contains the corresponding names of these test statistics. The third one (`Alter`) gives the type of test:

$$\texttt{Alter} = \begin{cases} 0 & \text{for a two-sided test with } \mathcal{R}_\alpha = \{T_n < c_L(\alpha)\} \cup \{T_n > c_R(\alpha)\}; \\ 1 & \text{for a one-sided test with } \mathcal{R}_\alpha = \{T_n < c_\alpha\}; \\ 2 & \text{for a one-sided test with } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 3 & \text{for a two-sided test with } \mathcal{R}_\alpha = \{T_n > c_\alpha\}; \\ 4 & \text{for a two-sided test with } \mathcal{R}_\alpha = \{T_n < c_\alpha\}. \end{cases}$$

If `Alter` $= 0$, Fisher's doubled p value will be computed. The fourth argument (`Nbparams`) gives the number of parameters for the corresponding test statistic. This is illustrated below.

```
R> head(getindex()$mat.stats)
```

```
  Index Stat Alter Nbparams
1     1  K-S     3        0
2     2 AD^*     3        0
3     3  Z_C     3        0
4     4  Z_A     3        0
5     5  P_S     3        0
6     6  K^2     3        0
```

Note that the function `getindex()` can be used to obtain this information for a subset of all test statistics using its argument `stat.indices`:

```
R> getindex(stat.indices = c(17, 23, 78))
```

```
   Index                 Stat Alter Nbparams
17    17                  T_w 0,1,2        0
23    23            tilde{W}     4        0
78    78 D_{n,m}(phi_lambda)     3        2
```

The function `help.stat()` displays some information about a given test statistic. Try for example `help.stat(21)` to display the documentation for the test whose index is 21, namely the Shapiro-Wilk test for non-normality.

3.4. Perform a test on an observed data sample

The function `statcompute()` enables one to perform a test on a given sample of observed (or simulated) data. This is illustrated by showing that it gives identical results to the `shapiro.test()` function from **stats** package (R Core Team 2015).

```
R> set.seed(1)
R> x <- rnorm(10)
R> shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.93828, p-value = 0.534

R> statcompute(stat.index = 21, data = x, level = 0.05)

$statistic
[1] 0.9382803

$pvalue
[1] 0.5340414
```

The arguments of this function are:

- `stat.index`: this should be a single integer-valued index.
- `data`: sample from which to compute the test statistic.
- `levels`: vector giving the desired significance levels for the test.
- `critvalL`: if not `NULL`, a vector of left critival values;

  `critvalR`: if not `NULL`, a vector of right critival values.

  The length of each vector of critical values should be the same as the length of the argument `levels`. A convenient way to fill these values is either to use the quantiles of the asymptotic null distribution (e.g., `qchisq(0.95,2)` for a χ^2_2 distribution), or to use the function `compquant()` to compute them using a Monte Carlo simulation with M repetitions:

  ```
  R> crit <- compquant(n = 10, law.index = 2, stat.index = 21, M = 10^5)
  R> crit$quant

       2.5%        5%       10%       90%       95%     97.5%
  0.8186749 0.8443032 0.8702271 0.9714234 0.9782153 0.9830226
  ```

  Note that if both `critvalL` and `critvalR` are `NULL`, then the decisions to reject $\mathcal{H}_0$ (1 is the code for a rejection) are taken by comparing the p value to each element of `levels`. The p value is, most of the time, computed using the asymptotic (or tabulated[1]) distribution of the test statistic under the null. If this computation is not possible, the `NA` value is returned. Nevertheless, the function `pvalueMC()`, described later on, can always be used.

[1]These tables can be found in the original articles where the test statistic has been published. An advanced user could also look into the `pvalue*.cpp` files in folder `src/law-stats/stats/pvalues/` (e.g., `pvalue42.cpp`).

- `alter`: a single integer value in $\{0, 1, 2, 3, 4\}$, explained at the beginning of Section 3.3.
- `stat.pars`: A vector of parameters. If `NULL`, the default parameter values for this test statistic will be used. See Tables 4, 6 and 5 in the Appendix for a list of these parameters.

3.5. Monte Carlo p values

The function `pvalueMC()` can be used to compute a Monte Carlo empirical p value as described in Equation 1. Its arguments are

- `data`: sample from which to compute the p value.
- `stat.index`: this should be a single integer-valued index of the test statistic considered.
- `null.law.index`: index of the law under the null hypothesis.
- `M`: number of Monte Carlo repetitions; default value is 10^5.
- `alter`: integer value in $\{0, 1, 2, 3, 4\}$ giving the type of test as described above.
- `null.law.pars`: if not `NULL`, vector of the parameter values for the law specified by `law.index`.
- `stat.pars`: if not `NULL`, vector of parameter values for the test.
- `list.stat`: if not `NULL`, a vector of M statistic values computed under the null.
- `method`: not used. only the doubled p value of Fisher is available for the moment.

```
R> x <- rnorm(100)
R> statcompute(1, x, levels = 0.05, alter = 3)$pvalue

[1] 0.4342264

R> pvalueMC(x, stat.index = 1, null.law.index = 2, M = 10^5, alter = 3)

[1] 0.43143
```

3.6. Perform a simulation study using a script

In this section, we briefly present the main functions of our package by showing how to obtain the simulation results from an article already published. Puig and Stephens (2000) present several simulation results for tests of the Laplace distribution. Using our package **PoweR**, we can easily retrieve the same simulation results with a few command lines.

Critical values

Let us start by reproducing their Table 1, containing the percentage points (critical values) of the Cramér-von Mises statistic W^2 (index: 43). In the following, we consider $n = 1,000$

as being Infinity, the value 1 for `law.index` stands for the Laplace distribution, `M` denotes the number of Monte Carlo repetitions, `vectn` is the vector of sample sizes and `levels` is the vector of significance levels.

```
R> system.time({
+    law.index <- 1
+    M <- 10^5
+    vectn <- c(10, 15, 20, 35, 50, 75, 100, 1000)
+    levels <- c(0.50, 0.25, 0.10, 0.05, 0.025, 0.01)
+    table1 <- many.crit(law.index, stat.indices = 43, M, vectn, levels,
+      alter = list(stat43 = 3))
+  })

   user  system elapsed
 46.432   0.026  46.597

R> print(table1, digits = 3)

        n    0.5   0.25    0.1  0.05 0.025  0.01
[1,]   10 0.0485 0.0695 0.0968 0.118 0.140 0.171
[2,]   15 0.0539 0.0792 0.1137 0.142 0.171 0.210
[3,]   20 0.0509 0.0744 0.1060 0.131 0.157 0.191
[4,]   35 0.0539 0.0799 0.1156 0.144 0.173 0.212
[5,]   50 0.0528 0.0776 0.1113 0.138 0.166 0.206
[6,]   75 0.0542 0.0802 0.1157 0.144 0.173 0.212
[7,]  100 0.0534 0.0791 0.1142 0.142 0.171 0.209
[8,] 1000 0.0540 0.0800 0.1154 0.144 0.173 0.214
```

The function `many.crit()` computes critical values for several test statistics and significance levels. It outputs an object of class `critvalues`. Its arguments are

1. `law.index`: law index as displayed by function `getindex()`.
2. `stat.indices`: vector of the indices of statistics as displayed by function `getindex()`.
3. `M`: number of Monte Carlo repetitions to use.
4. `vectn`: vector of numbers of observations for the samples to be generated.
5. `levels`: vector of required level values.
6. `alter`: named-list with type of test for each statistical test: `alter[["stat`j`"]]` = 0, 1, 2, 3 or 4.
7. `law.pars`: if not `NULL`, a vector of length at most 4 containing parameters of the law from which to generate random values.
8. `parstats`: named-list of parameter values for each statistic to simulate. The names of the list should be `stat`j, j taken in `stat.indices`. If `NULL`, the default parameter values for the corresponding statistic will be used.

	Significance level (α)					
Sample size (n)	0.5	0.25	0.1	0.05	0.025	0.01
10	0.049	0.069	0.097	0.118	0.140	0.171
15	0.054	0.079	0.114	0.142	0.171	0.210
20	0.051	0.074	0.106	0.131	0.157	0.191
35	0.054	0.080	0.116	0.144	0.173	0.212
50	0.053	0.078	0.111	0.138	0.166	0.206
75	0.054	0.080	0.116	0.144	0.173	0.212
100	0.053	0.079	0.114	0.142	0.171	0.209
1000	0.054	0.080	0.115	0.144	0.173	0.214

Table 2: Critical values of W^2 test.

9. `model`: `NULL`. Not implemented yet.

10. `Rlaw` : If `law.index` is set to 0, then `Rlaw` should be a (random generating) function.

11. `Rstats`: A list of the same length as `stat.indices`. If a value of the vector `stat.indices` is set to 0, the corresponding component of the list `Rstats` should be an R function that outputs a list with components `statistic` (value of the test statistic), `pvalue` (p-value of the test; if not computable should be set to 0), `decision` (vector of decisions for each value in `levels`; 1 if we reject the null, 0 otherwise), `alter` (see above), `stat.pars` (see above), `pvalcomp` (`1L` if the p-value can be computed, `0L` otherwise), `nbparstat` (length of `stat.pars`). A user can thus provide its own R functions to perform statistical tests. Note: If a value of `stat.indices` is not 0, then the corresponding component of `Rstats` should be set to `NULL`. The input arguments of this R function should be `data`, `levels`, `usecrit`=0 (will be set to 1 upon call if critical values are to be used to take the decisions), `critvalL`=0 and `critvalR`=0. Depending on the value of `alter`, only `critvalL` or only `critvalR` or both will be used.

The function `print.critvalues()` allows us to display the simulation results of critical values for the Cramér-von Mises W^2 statistic. Manipulating its arguments correctly yields the required display:

1. `x`: critical values given by function `many.crit()`.

2. `digits`: integer indicating the number of decimal places to be used.

3. `latex.output`: logical. If `TRUE`, we output LaTeX code for the table of critical values. If `FALSE`, we display a table in the R console.

If `latex.output = TRUE`, a LaTeX code is returned. After compilation, with `pdflatex` say, this gives directly Table 2 below, which is virtually identical to Table 1 in Puig and Stephens (2000). Note that the function `print.critvalues()` automatically adapts its result (to produce one or several tables) depending on the number of test statistics.

Note that the same procedure can be used to obtain in one shot the critical values of Tables 1-4 in Puig and Stephens (2000) for the Watson U^2, Anderson-Darling A^2, Kolmogorov $\sqrt{n}D$ and Kuiper V statistics.

Power

These critical values being obtained, we can study the empirical power of these tests using the instructions below. The alternative distributions we have considered are the Normal, Cauchy, Logistic symmetric and Generalized Extreme Value distributions, for which the indices (given in `law.indices`) are 2, 3, 4 and 28, respectively. As before, the value 1 for `law.index` stands for the Laplace distribution, and `alter` is a list defining the type of each test. Note that the default values that we used for the alternative distributions can be obtained using the function `law.cstr()`.

```
R> system.time({
+    law.index <- 1
+    M <- 10^5
+    vectn <- c(10, 15, 20, 35, 50, 75, 100)
+    levels <- 0.05
+    stat.indices <- c(43, 44, 42, 45, 46)
+    law.indices <- c(2, 3, 4, 28)
+    alter <- create.alter(stat.indices, rep(3, 5))
+    critval <- many.crit(law.index, stat.indices, M, vectn, levels, alter)
+    table6 <- powcomp.fast(law.indices, stat.indices, vectn, M, levels,
+      critval = critval, alter)
+  })

   user  system elapsed
152.362   0.041 152.822
```

Remark 3.1. If you have at your disposal a multicore processor, you can use the argument `nbclus=`p, where p is the number of cores on which you want to perform the computations.

The function `print.power()` works similarly to `print.critvalues()`. Table 3 below is obtained using the instruction `print(table6,digits=3,latex.output=TRUE)`. The results we have obtained are very similar to those presented in Table 6 from Puig and Stephens (2000). Note that our function adds the average power, the average gap power and the worst gap power to this table of powers (see Section 2.3 for a definition of these quantities).

The arguments of the function `powcomp.fast()` are:

1. `law.indices`: vector of law indices as displayed by function `getindex()`.
2. `stat.indices`: vector of indices of statistics as given by function `getindex()`.
3. `vectn`: vector of numbers of observations for the samples to be generated.
4. `M`: number of Monte Carlo repetitions to use.
5. `levels`: vector of required level values.

			Goodness-of-fit tests				
Alternative	n	α	W^2	U^2	A^2	$\sqrt{n}D$	V
Normal(mu=0,sigma=1)	10	0.05	4.985	5.52	4.589	4.65	5.407
	15	0.05	6.906	7.546	6.572	7.762	7.574
	20	0.05	7.468	11.116	6.747	8.61	10.566
	35	0.05	12.058	21.932	10.542	14.563	19.298
	50	0.05	17.209	33.531	14.917	19.74	29.343
	75	0.05	27.635	52.554	24.272	28.907	45.134
	100	0.05	40.934	68.833	36.258	37.895	59.987
Cauchy(mu=0,sigma=1)	10	0.05	33.748	39.232	35.658	32.399	38.036
	15	0.05	41.529	52.576	44.067	39.771	50.648
	20	0.05	53.839	63.26	55.911	51.114	61.017
	35	0.05	73.236	82.613	74.666	69.888	80.347
	50	0.05	86.347	92.122	87.236	83.201	90.572
	75	0.05	95.382	97.977	95.851	93.481	97.259
	100	0.05	98.68	99.527	98.811	97.75	99.287
Logistic(mu=0,sigma=1)	10	0.05	4.672	4.695	4.344	4.387	4.698
	15	0.05	5.983	5.543	5.606	6.327	5.655
	20	0.05	5.919	7.161	5.377	6.37	7.068
	35	0.05	8.155	11.832	7.096	9.382	10.773
	50	0.05	9.983	17.101	8.485	11.697	15.515
	75	0.05	13.921	26.13	11.465	15.92	23.043
	100	0.05	18.739	35.818	15.16	20.019	31.021
GEV(mu=0,sigma=1,xi=0)	10	0.05	7.437	6.871	7.715	6.436	6.645
	15	0.05	9.808	9.009	10.911	9.154	8.648
	20	0.05	12.007	12.431	13.993	10.626	11.337
	35	0.05	19.062	23.288	24.077	16.281	19.62
	50	0.05	28.544	35.614	38.068	22.724	29.631
	75	0.05	43.922	54.677	59.274	33.957	45.658
	100	0.05	61.426	71.114	78.618	46.877	60.424
Average power	n	α	W^2	U^2	A^2	$\sqrt{n}D$	V
	10	0.05	12.710	14.079	13.077	11.968	13.697
	15	0.05	16.056	18.669	16.789	15.754	18.131
	20	0.05	19.808	23.492	20.507	19.180	22.497
	35	0.05	28.128	34.916	29.095	27.529	32.509
	50	0.05	35.521	44.592	37.176	34.340	41.265
	75	0.05	45.215	57.834	47.715	43.066	52.773
	100	0.05	54.945	68.823	57.212	50.635	62.680
Average gap	n	α	W^2	U^2	A^2	$\sqrt{n}D$	V
	10	0.05	1.581	0.212	1.215	2.323	0.595
	15	0.05	3.337	0.725	2.605	3.640	1.263
	20	0.05	4.074	0.391	3.375	4.703	1.385
	35	0.05	6.986	0.197	6.018	7.585	2.604
	50	0.05	9.685	0.614	8.029	10.865	3.940
	75	0.05	13.769	1.149	11.268	15.918	6.210
	100	0.05	15.754	1.876	13.487	20.064	8.019
Worst gap	n	α	W^2	U^2	A^2	$\sqrt{n}D$	V
	10	0.05	5.484	0.844	3.574	6.833	1.196
	15	0.05	11.047	1.902	8.509	12.805	2.263
	20	0.05	9.421	1.562	7.349	12.146	2.656
	35	0.05	9.874	0.789	11.390	12.725	4.457
	50	0.05	16.322	2.454	18.614	15.344	8.437
	75	0.05	24.919	4.597	28.282	25.317	13.616
	100	0.05	27.899	7.504	32.575	31.741	18.194

Table 3: Power of W^2, U^2, A^2, $\sqrt{n}D$ and V tests.

6. `critval`: named-list of critical values for each test statistic. The names of the list should be `stat`j, j taken in `stat.indices`.

7. `alter`: named-list with type of test for each statistical test: `alter[["stat`j`"]]`=0, 1, 2, 3 or 4.

8. `parlaws`: named-list of parameter values for each law to simulate. The names of the list should be `law`j, j taken in `law.indices`. The length of vector `law`j should not be greater than 4.

9. `parstats`: named-list of parameter values for each statistic to simulate. The names of the list should be `stat`j, j taken in `stat.indices`. If `NULL`, the default parameter values for the corresponding statistics will be used.

10. `nbclus`: number of slaves to use for computation on a cluster.

11. `model`: `NULL`. Not implemented yet.

12. `Rlaws`: When some law indices in `law.indices` are equal to 0, this means that you will be using some R random generators. In that case, you should provide the names of the random generation functions in the corresponding components of `Rlaws` list, the other components should be set to `NULL`.

13. `Rstats`: A list of the same length as `stat.indices`. If a value of the vector `stat.indices` is set to 0, the corresponding component of the list `Rstats` should be an R function that outputs a list with components `statistic` (value of the test statistic), `pvalue` (p-value of the test; if not computable should be set to 0), `decision` (vector of decisions for each value in `levels`; 1 if we reject the null, 0 otherwise), `alter` (see above), `stat.pars` (see above), `pvalcomp` (`1L` if the p-value can be computed, `0L` otherwise), `nbparstat` (length of `stat.pars`). A user can thus provide its own R functions to perform statistical tests. Note: If a value of `stat.indices` is not 0, then the corresponding component of `Rstats` should be set to `NULL`. The input arguments of this R function should be `data`, `levels`, `usecrit`=0 (will be set to 1 upon call if critical values are to be used to take the decisions), `critvalL`=0 and `critvalR`=0. Depending on the value of `alter`, only `critvalL` or only `critvalR` or both will be used.

Plots

Our package **PoweR** provides not only the LaTeX tables of critical values and powers, but also plots based on p values as explained in Section 2.3. These are obtained (except for statistics 45 and 46 for which no general theory exists to find their asymptotic distribution) using the R instructions given below, and are shown in Figures 1, 2 and 3. Note that the results of Puig and Stephens (2000) are not that easy nor quick to obtain because the asymptotic distribution is an infinite sum of weighted chi-squared random variables. Unfortunately, the weights λ are not provided in their paper and computing them is not an easy task and involves optimization procedures that can be time consuming. This illustrates pretty well the reproducibility problem.

```
R> stind <- c(43, 44, 42)
R> alter <- create.alter(stind, rep(3, 3))
```

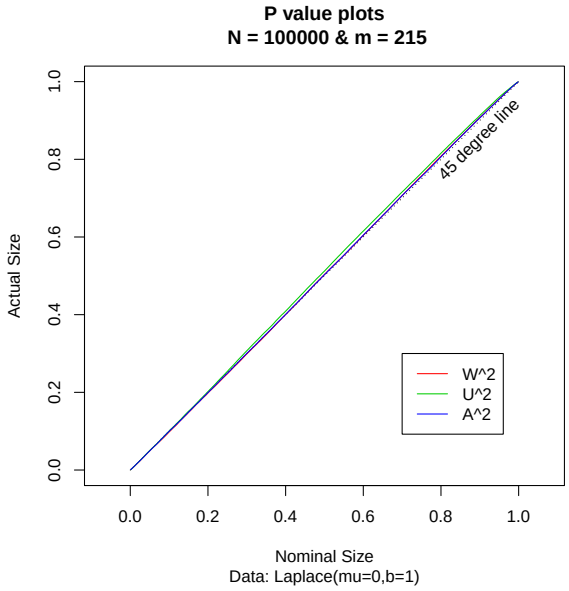

Figure 1: p value plot.

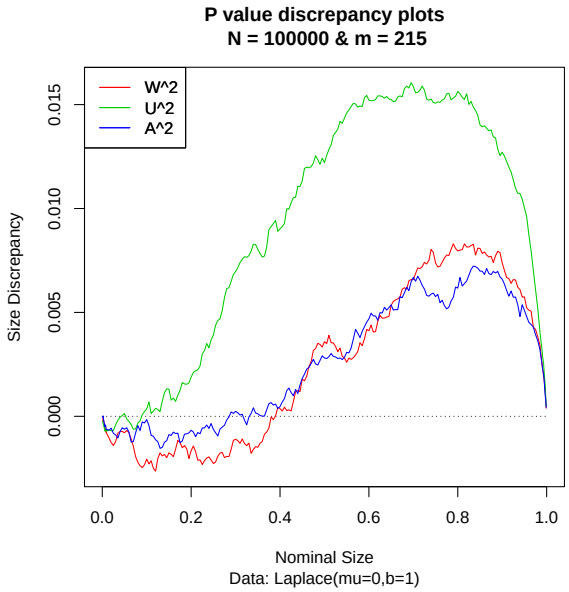

Figure 2: p value discrepancy plot.

```
R> system.time({
+  ptmpnull <- many.pval(stat.indices = stind, law.index = 1,
+    n = 100, M = 10^5, N = 10^5, alter = alter, null.dist = 2,
+    method = "direct")$pvals
+  })

     user    system   elapsed
31084.243     0.906 31170.972
```

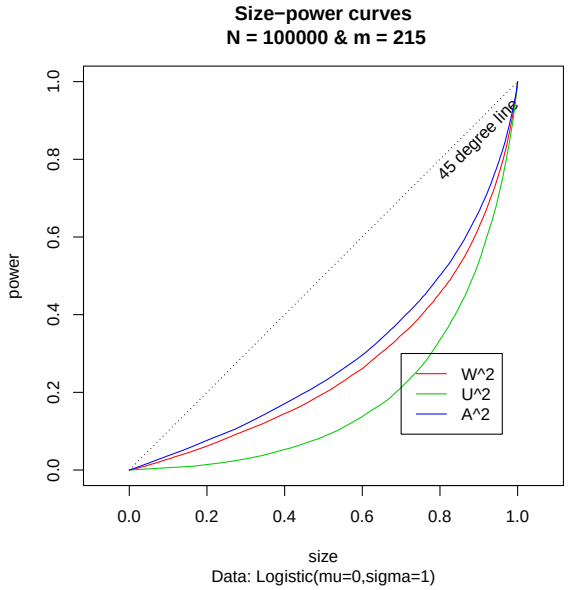

Figure 3: Size-power curves.

```
R> Fxnull <- calcFx(ptmpnull)
```

The call to the `many.pval()` function above, where `stind` contains the indices of the hypothesis tests and `alter` contains the type of each test, produces a $N \times 3$ matrix for which the k-th column is an N-dimensional vector of p values associated with the k-th test statistic ($1 \leq k \leq 3$). These p values are calculated via a direct method, namely using the true null distribution (if it is known), the asymptotic null distribution or any other theoretical approximation to it. Next, the function `calcFx()` is applied columnwise on this matrix of p values to compute $\hat{F}(x_j)$, using Equation 2 for the J points given in Equation 3.

Now we can produce the first two graphs (Figures 1 and 2):

```
R> plot.pvalue(Fxnull)
R> plot.discrepancy(Fxnull, legend.pos = "topleft")
```

To produce the size-power curves, we first need to use the following instructions, where now a Monte Carlo approach is employed to compute the p values.

```
R> system.time({
+  ptmpnull <- many.pval(stat.indices = stind, law.index = 1, n = 100,
+    M = 10^5, N = 10^5, alter = alter, null.dist = 2, method = "MC")$pvals
+  })

    user   system  elapsed
13279.27     0.34 13321.65

R> Fxnull <- calcFx(ptmpnull)
R> system.time({
```

```
+  ptmp <- many.pval(stat.indices = stind, law.index = 4, n = 100, M = 10^5,
+    N = 10^5, alter = alter, null.dist = 2, method = "MC")$pvals
+  })

     user    system   elapsed
13054.455     0.548 13100.285

R> Fx <- calcFx(ptmp)
```

This last instruction will give the third graph (Figure 3):

```
R> plot.sizepower(Fx, Fxnull)
```

3.7. Using user-defined densities and tests coded in R

It is possible to avoid the burden of a C++ implementation of your test or of the random generation procedure for a new density not yet included in the package. This is easily done through the `Rstats` and `Rlaw` (or `Rlaws`) arguments, as will be described below. Note however that this can lead to an increase in the computing time. Also, we think that one should consider implementing these in C++ (see Section 4) before publication in order to follow the guidelines presented in the Introduction (in particular step 5).

We present below a situation where we want to evaluate the power of the Shapiro-Wilks test for normality against the Benini alternative distribution. Random values of this distribution (not included in our package) can be obtained thanks to the `rbenini()` function in the **VGAM** package (Yee 2014). For pedagogic purpose, we show how to use both (and simultaneously) the Shapiro-Wilks test implemented (in C++) in our package (`Index`=21) and an R user-defined function of the same test called `my.shapiro()`.

Note that any R user-defined test function should follow the same pattern as the one below. Its input and output arguments are described in page 17. The body of this function contains a call to the base R function `shapiro.test()`. Obviously, if you develop a new goodness-of-fit test, you will replace it by a call to your own R procedure.

```
R> my.shapiro <- function(data, levels, usecrit = 0, critvalL = 0,
+    critvalR = 0)
+  {
+    res <- shapiro.test(data)
+    decisions <- rep(0, length(levels))
+    for (i in 1:length(levels)) {
+      if (usecrit == 0) {
+        decisions[i] <- if (res$p.value < levels[i]) 1 else 0
+      } else {
+        decisions[i] <- if (res$statistic < critvalL[i]) 1 else 0
+      }
+    }
+    return(list(statistic = res$statistic, pvalue = res$p.value,
+      decision = decisions, alter = 4, stat.pars = NULL, pvalcomp = 1L,
+      nbparstat = 0))
+  }
```

The following code computes critical values. Note that in order to use an R random generating function (`rnorm()` here), one has to set `law.index` to 0, specify the name of that R function via the `Rlaw` argument and specify the values of the parameters of the distribution through the `law.pars` argument. Similarly, to use an R function for your test (`my.shapiro()` here), you have to set the value of `stat.indices` to 0 and specify the name of that function via the `Rstats` argument. The other components of `Rstats`, corresponding to non-zero values of `stat.indices`, should be set to `NULL`.

```
R> M <- 10^5
R> levels <- 0.05
R> vectn <- c(10, 15, 100)
R> stind <- c(21, 0)
R> alter <- create.alter(stind, rep(4, 2))
R> (critval <- many.crit(law.index = 0, stat.indices = stind, M, vectn,
+    levels, alter, Rlaw = list(rnorm), law.pars = c(0, 1),
+    Rstats = list(NULL, my.shapiro)))

    n level critL.stat21 critL.stat0
1  10  0.05    0.8442145   0.8442145
2  15  0.05    0.8816759   0.8816759
3 100  0.05    0.9746649   0.9746649
```

Now we compute the empirical power using the above set of critical values. The functioning of the `Rlaws` and `Rstats` arguments are similar as above. We recall that `Index`=2 stands for the Gaussian distribution. Below, we generate observations from the standard Gaussian distribution and from the Benini$(1, e^1)$ distribution.

```
R> library("VGAM")
R> law.indices <- c(2, 0)
R> (powcomp.fast(law.indices, stind, vectn, M, levels, critval = critval,
+     alter, Rlaws = list(NULL,rbenini), parlaws = list(law2 = c(0, 1),
+     law0 = c(1, exp(1))), Rstats = list(NULL, my.shapiro)))

                           law   n level      W  stat0
1        Normal(mu=0,sigma=1)  10  0.05  4.815  4.815
2                              15  0.05  4.718  4.718
3                             100  0.05  4.946  4.946
4 rbenini(1,2.71828182845905)  10  0.05 25.288 25.288
5                              15  0.05 40.898 40.898
6                             100  0.05 99.983 99.983
                n level      W  stat0
Average power  10  0.05 15.052 15.052
2              15  0.05 22.808 22.808
3             100  0.05 52.465 52.465
              n level W stat0
Average gap  10  0.05 0     0
2            15  0.05 0     0
```

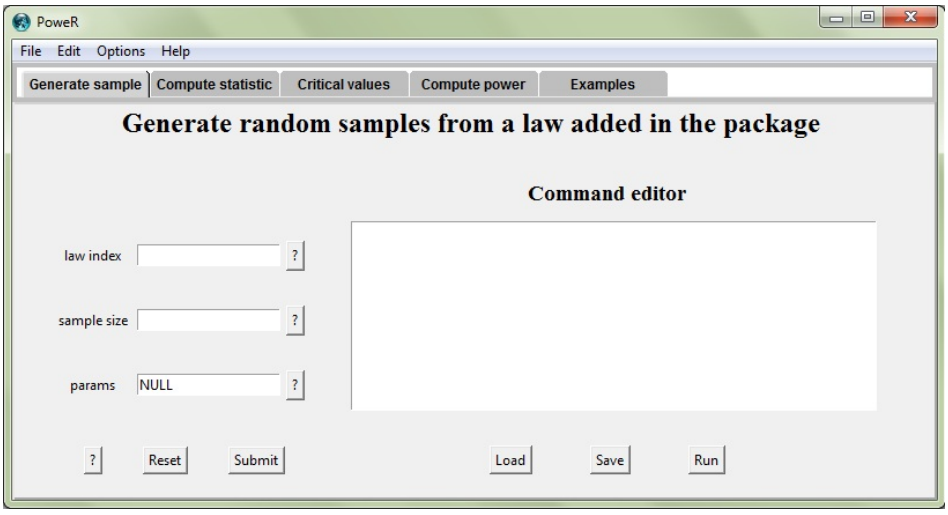

Figure 4: **PoweR** GUI.

```
3            100  0.05 0     0
             n level W stat0
Worst gap   10  0.05 0     0
2           15  0.05 0     0
3          100  0.05 0     0
```

3.8. The graphical user interface (GUI)

To aid in using the package, we have created a GUI. To start it, simply type `power.gui()` in the R console (see the 'Details' section in `help(power.gui)` concerning potential problems with `Iwidgets`). You should see the window shown in Figure 4.

This interface consists of five tabs:

- `Generate sample`: generate random samples from a law added in the package.
- `Compute statistic`: compute value of the test statistic for a given test index.
- `Critical values`: compute critical values of several test statistics.
- `Compute power`: compute power for hypothesis tests.
- `Examples`: reproduce simulation results already published in the literature.

The lower portion of the window – a different one corresponding to each tab – is divided into two parts: the left and the right sides.

- **Left side:** this contains required fields (e.g., indices of laws, tests, or parameters, etc.), a `Reset` button used to reset the fields to their original values and a `Submit` button used to send commands to a text editor on the right side of the window. In addition, `?` buttons instruct us on how to fill in the fields, as depicted in Figure 5.

  The `?` button at the bottom left sends us directly to the documentation of the function corresponding to this tab.

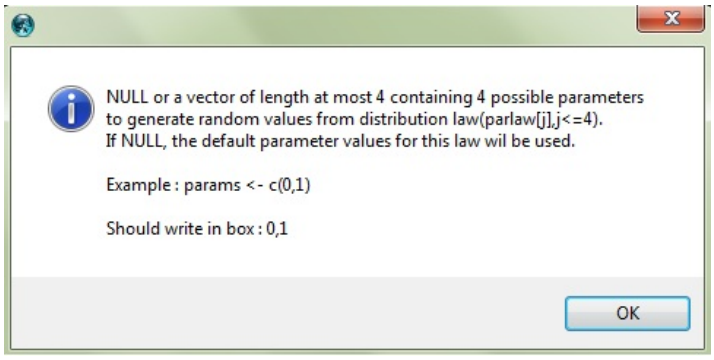

Figure 5: Help window `?`.

- **Right side:** this contains a text editor that receives commands from the left. It is easy to edit, then to save with the `Save` button, or to load a file already created by the `Load` button. We can execute commands via the `Run` button.

You will find in the `Examples` tab around 50 code examples which allow us to regenerate simulation results from many articles already published in the literature.

4. Extending PoweR

When a researcher plans a new simulation study to evaluate the performance, in terms of power, of a newly developed procedure, he or she might need to proceed as described below. Note that to help in integrating new functions into the package, we employed a mechanism described in Temple Lang (2001) to register our C/C++ routines. The developper then only has to add his own law and/or hypothesis test files in the appropriate subfolders.

4.1. Adding a law

It is possible to add a new law to our package **PoweR**. These new laws will then be used to generate random values in order to offer other choices of alternative distributions for power comparisons. We assume that you have already downloaded our package source code from the CRAN website. Here are a few steps to guide you through the process of implementing a new law in the package:

1. Add the definition of the new law in the two C++ files `def-laws-stats.cpp` and `register.cpp`, following the instructions given in those files. These two files are located in the subfolder `Power/src/laws-stats/`.

2. Create a new C++ file `law`j`.cpp` in subfolder `Power/src/laws-stats/laws/` that will contain the source code to generate observations from the new distribution, where j is an available index for the new law. Note that j should be taken as the value returned by the following instruction: `nrow(getindex()$mat.laws)+1`. Look at the other files in that directory to see how to write your own file. All files (should) have the same pattern which is succinctly described below. The function prototype (all arguments are pointers) is:

```
void lawj (int *xlen, double *x, char **name, int *getname,
                    double *params, int *nbparams, int *setseed);
```

where `xlen[0]` will receive from R the length of the data sample to be generated, `x` will be used for these generated data to be passed back to R, `name` will be a 50-length pointer to char pointers each one of length 1, the purpose being to output the name of the distribution, `getname[0]` will contain 1 or 0 depending if the distribution name should be retrieved or not (in the former case, no data will be generated), `params` (of length `nbparams[0]`) will receive the values of the parameters of the distribution, and finally the input parameter `seed[0]` will contain 1 if one wants to read in `.Random.seed` and write it out after use; otherwise it must be set to 0.

3. Create a help file for your new law in subfolder `Power/man/`. Follow the pattern of the other *law* help files in that same directory. You should also update the file `Distributions.Rd`.

4. Recompile package **PoweR** (Windows: `R CMD INSTALL -build PoweR`; Linux: `R CMD build PoweR`).

Moreover, it is advisable to program (in R language) the new law density function and to add it to the file `PoweR/inst/laws/densities.R` as well as the computation of the expectation and variance and add them to the file `PoweR/inst/laws/moments.R`. You should next update the files `PoweR/man/densities.Rd` and `PoweR/man/moments.Rd` by adding an alias to the new density and moments functions respectively.

After that you can use the function `gensample()` to verify whether the new added law behaves correctly. First of all, having a sufficient number of generated random numbers from the new law allows us to compute their empirical expectation and variance. We can then compare them with the theoretical values. This is exemplified below for a Weibull(10,1) distribution.

```
R> x <- gensample(law.index = 11, n = 10^5, c(10, 1))$sample
R> mean(x)

[1] 0.9511258

R> var(x)

[1] 0.01324107

R> moments11(10, 1)

$expectation
[1] 0.9513508

$variance
[1] 0.01310046
```

Next, using the function `checklaw()`, you can check if the histogram of the generated random values matches the corresponding density curve. This is illustrated below.

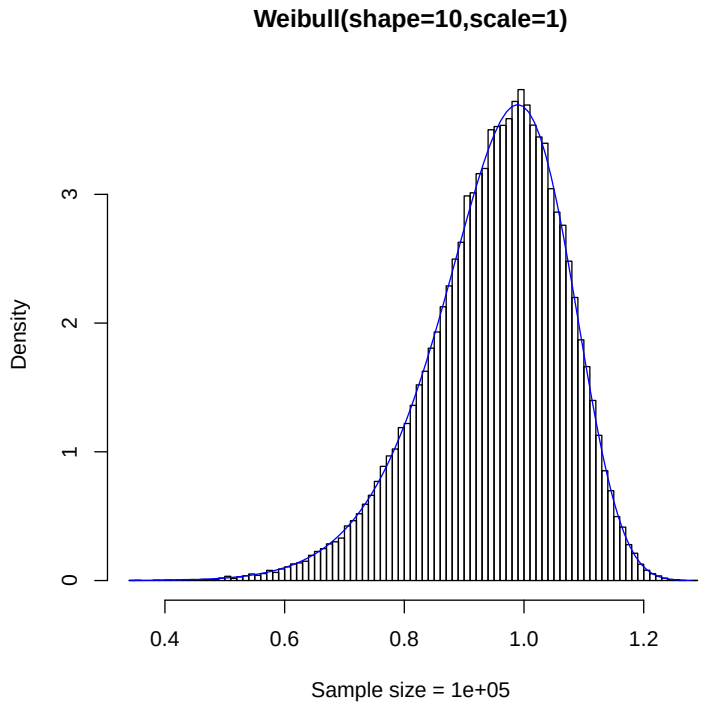

Figure 6: Graphical check for histogram and density from Weibull law.

```
R> checklaw(law.index = 11, sample.size = 10^5, law.pars = c(10, 1),
+    density = dlaw11)
```

Note that this last figure can also be obtained using the `dweibull` function instead of `dlaw11`.

4.2. Adding a test statistic

Once a new goodness-of-fit test has been developed, a researcher might want to compare it with other existing tests in the literature in terms of power. The idea is to add it to our package and use the function `powcomp.fast()` to perform this kind of comparison. We present in this section the steps to follow to implement a new test statistic in the package. We assume that you have already downloaded the source code of our package from the CRAN website.

1. Add the definition of the new test in the two C++ files `def-laws-stats.cpp` and `register.cpp`. These two files are located in subfolder `Power/src/laws-stats/`.

2. Create a new C++ file `stat`j`.cpp` in subfolder `Power/src/laws-stats/stats/` that contains the source code of this new test statistic, where j is an available index for the new test statistic. It is also important to create another C++ file named `pvalue`j`.cpp` in subfolder `Power/src/laws-stats/stats/pvalues/` which contains the code to compute the p value of the test number j. Note that j should be taken equal to the value returned by the following instruction: `nrow(getindex()$mat.stats)+1`. Look at the

other files in that directory to see how to write your own file. All files (should) have the same pattern which is described in the files `statj.cpp` and `pvaluej.cpp`. The function prototype (all arguments are pointers) is:

```
void statj(double *x, int *xlen, double *level, int *nblevel, char **name
           int *getname, double *statistic, int *pvalcomp, double *pvalue,
           double *critvalL, double *critvalR, int *usecrit, int *alter,
           int *decision, double *paramstat, int *nbparamstat);
```

where `x` will receive the data sample (of length `xlen[0]`), `level` will contain the `nblevel[0]` values of levels to be considered, `name` and `getname` are arguments similar to the ones used in the previous subsection (but for the test statistic name), `statistic[0]` will contain an output of the test statistic value, `pvalcomp[0]` is 1 or 0 depending on whether a computation of the p-value is required (the portion of C++ code for the computation of this p-value will be in the file `pvaluej.cpp` and called via `#include "pvalues/pvaluej.cpp"`), `pvalue[0]` will contain an output of the p-value (or `(int)0` if this computation is not possible), `critvalL` and `critvalR` will respectively contain an input of the vectors (of length `nblevel[0]`) of left and right critical values of the test, `usecrit[0]` is 1 if these critical values have to be used to take the decision to reject (`decision[i]=1`, $i = 0, \ldots,$`nblevel[0]`-1) or not (`decision[i]=0`) the null or if the p-value has to be used instead, `alter[0]` is an input integer value in $\{0, 1, 2, 3, 4\}$ described in Section 3.3, and finally `paramstat` and `nbparamstat` are similar to the arguments `params` and `nbparams` described in the previous section but here used for parameters of the test statistic.

3. Create a help file for your new test statistic in subfolder `Power/man/`. Follow the pattern of the other *stat* help files in that same directory. You should also consider updating one of the files `Laplace.tests.Rd`, `Normality.tests.Rd`, `Uniformity.tests.Rd`, or creating a new one, depending on the kind of test you want to include.

Now we give two pieces of advice to verify if your new goodness-of-fit test has been correctly coded. Firstly, if you already have a R code of your test statistic, simply use the function `statcompute()` described in Section 3 to compute the test statistic value, and compare it with the one obtained from your R code, as we have already shown in Section 3.3. Secondly, if you know how to compute your test p value, use the function `pvalueMC()`, which computes the value given in Equation 1, to compute a Monte Carlo empirical p value, and then compare it with the one computed inside the C++ file `pvaluej.cpp`. Make sure that for both the same vector of data is used to compute the results. This is illustrated below where we find two p values which are almost identical.

```
R> x <- rnorm(100)
R> pvalueMC(x, stat.index = 21, null.law.index = 2, M = 10^5, alter = 4)

[1] 0.16367

R> shapiro.test(x)$p.value

[1] 0.1656738
```

4.3. Adding an example in the GUI

To make simulation results from some published paper available in the 'Examples' tab of the GUI, one only has to create a file in the folder `PoweR/inst/examples/`, following the pattern of those already present. This will increase the reproducibility of simulation results published in the literature.

5. Conclusion

The package **PoweR**, available on the CRAN website, relieves in a reproducible manner the heavy work associated with producing Monte Carlo simulations for goodness-of-fit tests for i.i.d. data. Ideally, once a new goodness-of-fit test has been published it should be included (with a reference to the publication) in our package, which we think could become a very useful tool for our community. The authors of a new test published in the literature could send to the maintainer of the **PoweR** package their files `law`j`.cpp`, `law`j`.R`, `densities.R`, `moments.R`, `stat`j`.cpp`, `pvalue`j`.cpp`, `stat`j`.R` (and `example.txt` for inclusion in the GUI examples) as explained in Section 4.

The current version of our package is currently restricted to the case of simple null hypotheses or for pivotal test statistics for which the set of critical values does not depend on a particular choice of a null distribution (and on nuisance parameters) under the non-simple null case. When this is not the case, a classical Monte Carlo approach is not valid anymore because of the presence of nuisance parameters. As such, the `pvalueMC()` and `many.crit()` functions are potentially dangerous. These problems could manifest if users want to add new tests to the package. For example, if they want to test a distribution with an unknown shape parameter, then the Anderson-Darling test statistic would depend on that parameter. This problem can also be seen in the Algorithm on page 5. Indeed, when $\mathcal{H}_0$ is a family of distributions, each $\boldsymbol{\theta}_0 \in \Theta_0$ would lead to a different (pair of) critical value(s). One should then compute critical values for all $\boldsymbol{\theta}_0 \in \Theta_0$ and take the supremum (resp. infimum) of these right (resp. left) critical values in order to build a test procedure controlling the (maximal) type-I error rate. Similarly, to calculate p-values, one should take something of a supremum over the parameter space Θ_0. The problem here is that the cardinal of Θ_0 could be infinite. Several approaches can be considered to try to circumvent this problem: a) Using a (finite) grid of points over Θ_0, hopping that this will be sufficient; b) choose a value $\boldsymbol{\theta}$ such that $\mathcal{P}_0(\boldsymbol{\theta}) \in \mathcal{A}$ (i.e., corresponds to a null distribution) and such that the Kullback-Leibler (KL) distance between the distribution $\mathcal{P}_0(\boldsymbol{\theta})$ and the alternative distribution (with density $g(x)$ say) is minimum (see Remark 2.1). Note that, for a power computation, the alternative distribution is chosen by the user and should be entirely specified. For example, $g(x)$ could be the $U[0,1]$ uniform density and $\mathcal{P}_0(x, \boldsymbol{\theta})$ the density of the $\mathcal{N}(\mu, \sigma^2)$ Gaussian distribution, where $\boldsymbol{\theta} = (\mu, \sigma^2)$. The objective would then be to find a $\boldsymbol{\theta}$ such that

$$\int_{-\infty}^{+\infty} \mathcal{P}_0(x, \boldsymbol{\theta}) \log \frac{\mathcal{P}_0(x, \boldsymbol{\theta})}{g(x)} dx$$

is minimum. This can be achieved by numerical integration and optimization. The intuition is that we want to be able to measure the capability (power) of some test to detect departures from a given set of distributions corresponding to the null hypothesis towards some entirely specified alternative. This will be more difficult if the chosen null distribution is close to

the alternative, which is the choice made using this KL approach. Note that to compute a p-value for an observed sample, one could start by replacing $g(x)$ by some density estimate based on the sample; c) add an option for parametric bootstrapping to generate observations under the null when the distribution of the test statistic depends about nuisance parameters of the null distributions. These nuisance parameters would be estimated using the sample simulated under the alternative (or the observed sample), the intuition being to choose a null distribution closest in some sense to the alternative (or true) distribution.

Some other future avenues of development include:

- to treat the case of test statistics having a discrete distribution, for example using the empirical quantiles of Ma, Genton, and Parzen (2011);
- to compute empirical critical values with a non-equal probability of rejection of the null in the tails (for bilateral tests);
- to perform simulations for goodness-of-fit tests for the errors of parametric models;
- to use a cluster of workstations; and
- to provide other templates for the LaTeX output of critical values and powers, with an alias in files `print.critvalues.Rd` and `print.power.Rd`).

Other ideas in which to use this package could also emerge. For example, one can use it with students for pedagogic purposes to investigate the robustness of the Student t-test (test index: 83) to non-normality of the data.

Acknowledgments

This research has been funded by the NSERC of Canada. We thank the anonymous reviewer and an editor for their helpful comments that improved the quality of this work.

References

Anderson TW, Darling DA (1954). "A Test of Goodness of Fit." *Journal of the American Statistical Association*, **49**, 765–769. doi:10.1080/01621459.1954.10501232.

Atkinson AC (1982). "The Simulation of Generalized Inverse Gaussian and Hyperbolic Random Variables." *Society for Industrial and Applied Mathematics. Journal on Scientific and Statistical Computing*, **3**(4), 502–515. doi:10.1137/0903033.

Azzalini A (2005). "The Skew-Normal Distribution and Related Multivariate Families." *Scandinavian Journal of Statistics. Theory and Applications*, **32**(2), 159–200. doi:10.1111/j.1467-9469.2005.00426.x.

Bates GE (1955). "Joint Distributions of Time Intervals for the Occurrence of Successive Accidents in a Generalized Pólya Scheme." *The Annals of Mathematical Statistics*, **26**, 705–720. doi:10.1214/aoms/1177728429.

Bhattacharya B, Habtzghi D (2002). "Median of the p Value Under the Alternative Hypothesis." *The American Statistician*, **56**(3), 202–206. doi:10.1198/000313002146.

Bonett DG, Seier E (2002). "A Test of Normality With High Uniform Power." *Computational Statistics & Data Analysis*, **40**(3), 435–445. doi:10.1016/s0167-9473(02)00074-9.

Bontemps C, Meddahi N (2005). "Testing Normality: A GMM Approach." *Journal of Econometrics*, **124**(1), 149–186. doi:10.1016/j.jeconom.2004.02.014.

Brunk HD (1962). "On the Range of the Difference Between Hypothetical Distribution Function and Pyke's Modified Empirical Distribution Function." *The Annals of Mathematical Statistics*, **33**, 525–532. doi:10.1214/aoms/1177704578.

Brys G, Hubert M, Struyf A (2008). "Goodness-of-Fit Tests Based on a Robust Measure of Skewness." *Computational Statistics*, **23**(3), 429–442. doi:10.1007/s00180-007-0083-7.

Cabaña A, Cabaña EM (1994). "Goodness-of-Fit and Comparison Tests of the Kolmogorov-Smirnov Type for Bivariate Populations." *The Annals of Statistics*, **22**(3), 1447–1459.

Castillo E, Hadi AS, Balakrishnan N, Sarabia JM (2005). *Extreme Value and Related Models With Applications in Engineering and Science*. Wiley Series in Probability and Statistics. John Wiley & Sons, Hoboken, NJ.

Chambers JM, Mallows CL, Stuck BW (1976). "A Method for Simulating Stable Random Variables." *Journal of the American Statistical Association*, **71**(354), 340–344. doi:10.1080/01621459.1976.10480344.

Chen L, Shapiro SS (1995). "An Alternative Test for Normality Based on Normalized Spacings." *Journal of Statistical Computation and Simulation*, **53**, 269–288. doi:10.1080/00949659508811711.

Choi B, Kim K (2006). "Testing Goodness-of-Fit for Laplace Distribution Based on Maximum Entropy." *Statistics*, **40**(6), 517–531. doi:10.1080/02331880600822473.

Claerbout J, Karrenbach M (1992). "Electronic Documents Give Reproducible Research a New Meaning." In *Proceedings of the 62nd Annual International Meeting of the Society of Exploration Geophysics*, pp. 601–604.

Coin D (2008). "A Goodness-of-Fit Test for Normality Based on Polynomial Regression." *Computational Statistics & Data Analysis*, **52**(4), 2185–2198. doi:10.1016/j.csda.2007.07.012.

Coles S (2001). *An Introduction to Statistical Modeling of Extreme Values*. Springer Series in Statistics. Springer-Verlag, London. ISBN 1-85233-459-2. doi:10.1007/978-1-4471-3675-0.

Cressie N (1978). "Power Results for Tests Based on High Order Gaps." *Biometrika*, **65**(1), 214–218. doi:10.1093/biomet/65.1.214.

Cressie N (1979). "An Optimal Statistic Based on Higher Order Gaps." *Biometrika*, **66**(3), 619–627.

D'Agostino RB (1971). "An Omnibus Test of Normality for Moderate and Large Size Samples." *Biometrika*, **58**, 341–348. doi:10.1093/biomet/58.2.341.

D'Agostino RB, Pearson ES (1973). "Tests for Departure From Normality. Empirical Results for the Distributions of b_2 and $\sqrt{b_1}$." *Biometrika*, **60**, 613–622. doi:10.1093/biomet/60.3.613.

D'Agostino RB, Pearson ES (1974). "Correction To: "Tests for Departure From Normality. Empirical Results for the Distributions of b_2 and $\sqrt{b_1}$" (Biometrika **60** (1973), 613–622)." *Biometrika*, **61**, 647. doi:10.1093/biomet/60.3.613.

D'Agostino RB, Stephens MA (1986). *Goodness-of-Fit Techniques*. Marcel Dekker, New York.

Davidson R, MacKinnon JG (1996). "The Power of Bootstrap Tests." *Technical report*, Queen's Institute for Economic Research Discussion Paper 937.

Davidson R, MacKinnon JG (1998). "Graphical Methods for Investigating the Size and Power of Hypothesis Tests." *The Manchester School*, **66**(1), 1–26. doi:10.1111/1467-9957.00086.

del Barrio E, Cuesta-Albertos J, Matran C, Rodriguez-Rodriguez J (1999). "Tests of Goodness-of-Fit Based on the L_2-Wasserstein Distance." *The Annals of Statistics*, **27**, 1230–1239. doi:10.1214/aos/1017938923.

Desgagné A, Angers JF (2005). "Importance Sampling With the Generalized Exponential Power Density." *Statistics and Computing*, **15**(3), 189–196. doi:10.1007/s11222-005-1308-7.

Desgagné A, Lafaye de Micheaux P (2016). "Tests of Normality Based on 2nd-Power Skewness and Kurtosis." Submitted.

Desgagné A, Lafaye de Micheaux P, Leblanc A (2009). "P-Kurtosis and Goodness-of-Fit Tests for Normality." Unpublished.

Desgagné A, Lafaye de Micheaux P, Leblanc A (2013). "Test of Normality Against Generalized Exponential Power Alternatives." *Communications in Statistics - Theory and Methods*, **42**(1), 164–190. doi:10.1080/03610926.2011.577548.

Desgagné A, Lafaye de Micheaux P, Leblanc A (2014). "Goodness-of-Fit Tests for the Laplace Distribution." Unpublished.

Doornik JA, Hansen H (2008). "An Omnibus Test for Univariate and Multivariate Normality." *Oxford Bulletin of Economics and Statistics*, **70**, 927–939. doi:10.1111/j.1468-0084.2008.00537.x.

Durbin J (1969). "Tests for Serial Correlation in Regression Analysis Based on the Periodogram of Least-Squares Residuals." *Biometrika*, **56**, 1–15. doi:10.1093/biomet/56.1.1.

Ehrenberg ASC (1977). "Rudiments of Numeracy." *Journal of the Royal Statistical Society A*, **140**(3), 277–297. doi:10.2307/2344922.

Epps TW, Pulley LB (1983). "A Test for Normality Based on the Empirical Characteristic Function." *Biometrika*, **70**(3), 723–726.

Feller W (1968). *An Introduction to Probability Theory and Its Applications. Vol. I.* Third edition. John Wiley & Sons, New York.

Feller W (1971). *An Introduction to Probability Theory and Its Applications. Vol. II.* Second edition. John Wiley & Sons, New York.

Filliben JJ (1975). "The Probability Plot Correlation Coefficient Test for Normality." *Technometrics*, **17**(1), 111–117. doi:10.1080/00401706.1975.10489279.

Gel YR (2010). "Test of Fit for a Laplace Distribution Against Heavier Tailed Alternatives." *Computational Statistics & Data Analysis*, **54**(4), 958–965. doi:10.1016/j.csda.2009.10.008.

Gel YR, Gastwirth JL (2008). "A Robust Modification of the Jarque-Bera Test of Normality." *Economics Letters*, **99**(1), 30–32. doi:10.1016/j.econlet.2007.05.022.

Gel YR, Miao W, Gastwirth JL (2007). "Robust Directed Tests of Normality Against Heavy-Tailed Alternatives." *Computational Statistics & Data Analysis*, **51**(5), 2734–2746. doi:10.1016/j.csda.2006.08.022.

Glen AG, Leemis LM, Barr DR (2001). "Order Statistics in Goodness-of-Fit Testing." *IEEE Transactions on Reliability*, **50**(2), 209–213. doi:10.1109/24.963129.

Greenwood M (1946). "The Statistical Study of Infectious Diseases." *Journal of Royal Statistical Society A*, **109**, 85–110. doi:10.2307/2981176.

Gross J, bug fixes by Uwe Ligges (2012). ***nortest**: Tests for Normality*. R package version 1.0-2, URL https://CRAN.R-project.org/package=nortest.

Gulati S (2011). "Goodness of Fit Test for the Rayleigh and the Laplace Distributions." *International Journal of Applied Mathematics & Statistics*, **24**(SI-11A), 74–85.

Hegazy YAS, Green JR (1975). "Some New Goodness-of-Fit Tests Using Order Statistics." *Applied Statistics*, **24**, 299–308. doi:10.2307/2347090.

Hosking JRM (1990). "*L*-Moments: Analysis and Estimation of Distributions Using Linear Combinations of Order Statistics." *Journal of the Royal Statistical Society B*, **52**(1), 105–124.

Jarque CM, Bera AK (1987). "A Test for Normality of Observations and Regression Residuals." *International Statistical Review. Revue International de Statistique*, **55**(2), 163–172. doi:10.2307/1403192.

Johnson NL (1949). "Systems of Frequency Curves Generated by Methods of Translation." *Biometrika*, **36**, 149–176. doi:10.1093/biomet/36.1-2.149.

Kallenberg WCM, Ledwina T (1997). "Data Driven Smooth Tests for Composite Hypotheses: Comparison of Powers." *Journal of Statistical Computation and Simulation*, **59**(2), 101–121. doi:10.1080/00949659708811850.

Kolmogorov AN (1933). "Sulla Determinazione Empirica Di Una Legge Di Distibuziane." *Giornale dell'Istituta Italiano degli Attuari*, **4**, 83–91.

Kotz S, Kozubowski TJ, Podgórski K (2001). *The Laplace Distribution and Generalizations*. Birkhäuser Boston Inc., Boston, MA. A revisit with applications to communications, economics, engineering, and finance.

Koziol JA (1989). "A Note on Plots of P-Values to Evaluate Many Tests Simultaneously." *Biometrical Journal*, **31**(8), 969–972. doi:10.1002/bimj.4710310813.

Kreher DL, Stinson DR (2005). *A Pseudocode: A LaTeX Style File for Displaying Algorithms*. Department of Mathematical Sciences, Michigan Technological University, Houghton, MI 49931. URL http://mirrors.ctan.org/macros/latex/contrib/pseudocode/pseudocode.pdf.

Kulinskaya E (2008). "On Two-Sided P-Values for Non-Symmetric Distributions." arXiv:0810.2124.

Kundu D (2005). "Discriminating Between Normal and Laplace Distributions." In *Advances in Ranking and Selection, Multiple Comparisons, and Reliability*, Stat. Ind. Technol., pp. 65–79. Birkhäuser Boston, Boston, MA.

Langholz B, Kronmal RA (1991). "Tests of Distributional Hypotheses With Nuisance Parameters Using Fourier Series Methods." *Journal of the American Statistical Association*, **86**(416), 1077–1084. doi:10.1080/01621459.1991.10475154.

Leone FC, Nelson LS, Nottingham RB (1961). "The Folded Normal Distribution." *Technometrics*, **3**, 543–550. doi:10.1080/00401706.1961.10489974.

Lieber RL (1990). "Statistical Significance and Statistical Power in Hypothesis Testing." *Journal of Orthopaedic Research*. doi:10.1002/jor.1100080221.

Lilliefors H (1967). "On the Kolmogorov-Smirnov Test for Normality With Mean and Variance Unknown." *Journal of the American Statistical Association*, **62**, 399–402. doi:10.1080/01621459.1967.10482916.

Ma Y, Genton MG, Parzen E (2011). "Asymptotic Properties of Sample Quantiles of Discrete Distributions." *Annals of the Institute of Statistical Mathematics*, **63**, 227–243. doi:10.1007/s10463-008-0215-z.

Marhuenda MA, Marhuenda Y, Morales D (2005). "Uniformity Tests Under Quantile Categorization." *Kybernetes*, **34**(6), 888–901. doi:10.1108/03684920510595553.

Martinez J, Iglewicz B (1981). "A Test for Departure From Normality Based on a Biweight Estimator of Scale." *Biometrika*, **68**(1), 331–333.

Meintanis SG (2004). "A Class of Omnibus Tests for the Laplace Distribution Based on the Empirical Characteristic Function." *Communications in Statistics. Theory and Methods*, **33**(4), 925–948. doi:10.1081/sta-120028735.

Morales D, Pardo L, Pardo MC, Vajda I (2003). "Limit Laws for Disparities of Spacings." *Journal of Nonparametric Statistics*, **15**(3), 325–342. doi:10.1080/1048525031000120206.

Moran PAP (1951). "The Random Division of an Interval. II." *Journal of the Royal Statistical Society B*, **13**, 147–150.

Nadarajah S (2005). "A Generalized Normal Distribution." *Journal of Applied Statistics*, **32**(7), 685–694. doi:10.1080/02664760500079464.

Pardo MC (2003). "A Test for Uniformity Based on Informational Energy." *Statistical Papers*, **44**(4), 521–534. doi:10.1007/bf02926008.

Puig P, Stephens MA (2000). "Tests of Fit for the Laplace Distribution, With Applications." *Technometrics*, **42**(4), 417–424. doi:10.2307/1270952.

Quesenberry CP, Miller FLJ (1977). "Power Studies of Some Tests for Uniformity." *Journal of Statistical Computation and Simulation*, **5**, 169–191. doi:10.1080/00949657708810150.

Rahman MM, Govindarajulu Z (1997). "A Modification of the Test of Shapiro and Wilk for Normality." *Journal of Applied Statistics*, **24**(2), 219–235. doi:10.1080/02664769723828.

Rayner JCW, Best DJ (1989). *Smooth Tests of Goodness of Fit*. The Clarendon Press Oxford University Press, New York. ISBN 0-19-505610-8.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Read TRC, Cressie NAC (1988). *Goodness-of-Fit Statistics for Discrete Multivariate Data*. Springer Series in Statistics. Springer-Verlag, New York. ISBN 0-387-96682-X. doi:10.1007/978-1-4612-4578-0.

Romão X, Delgado R, Costa A (2010). "An Empirical Power Comparison of Univariate Goodness-of-Fit Tests for Normality." *Journal of Statistical Computation and Simulation*, **80**(5-6), 545–591. doi:10.1080/00949650902740824.

Schweder T, Spjøtvoll E (1982). "Plots of P-Values to Evaluate Many Tests Simultaneously." *Biometrika*, **69**(3), 493–502.

Shapiro SS, Francia R (1972). "An Approximation Analysis of Variance Test for Normality." *Journal of the American Statistical Association*, **67**, 215–216. doi:10.1080/01621459.1972.10481232.

Shapiro SS, Wilk MB (1965). "An Analysis of Variance Test for Normality: Complete Samples." *Biometrika*, **52**, 591–611. doi:10.1093/biomet/52.3-4.591.

Spiegelhalter DJ (1977). "A Test for Normality Against Symmetric Alternatives." *Biometrika*, **64**(2), 415–418. doi:10.1093/biomet/64.2.415.

Swartz T (1992). "Goodness-of-Fit Tests Using Kullback-Leibler Information." *Communications in Statistics. Simulation and Computation*, **21**(3), 711–729. doi:10.1080/03610919208813046.

Temple Lang D (2001). "In Search of C/C++ & Fortran Routines." *R News*, **1**(3), 20–23.

Vasicek O (1976). "A Test for Normality Based on Sample Entropy." *Journal of the Royal Statistical Society B*, **38**(1), 54–59.

Wilk MB, Gnanadesikan R (1968). "Probability Plotting Methods for the Analysis of Data." *Biometrika*, **55**(1), pp. 1–17. doi:10.1093/biomet/55.1.1.

Wilson EB (1927). "Probable Inference, the Law of Succession, and Statistical Inference." *Journal of the American Statistical Association*, **22**, 209–212. doi:10.2307/2276774.

Yates F (1984). "Tests of Significance for 2 × 2 Contingency Tables." *Journal of the Royal Statistical Society A*, **147**(3), 426–463. doi:10.2307/2981577.

Yee TW (2014). ***VGAM**: Vector Generalized Linear and Additive Models*. R package version 0.9-5, URL https://CRAN.R-project.org/package=VGAM.

Yen VC, Moore AH (1988). "Modified Goodness-of-Fit Test for the Laplace Distribution." *Communications in Statistics - Simulation and Computation*, **17**(1), 275–281. doi:10.1080/03610918808812661.

Zhang J (2002). "Powerful Goodness-of-Fit Tests Based on the Likelihood Ratio." *Journal of the Royal Statistical Society B*, **64**(2), 281–294. doi:10.1111/1467-9868.00337.

Zhang J, Wu Y (2005). "Likelihood-Ratio Tests for Normality." *Computational Statistics & Data Analysis*, **49**(3), 709–721. doi:10.1016/j.csda.2004.05.034.

Zhang P (1999). "Omnibus Test of Normality Using the Q Statistic." *Journal of Applied Statistics*, **26**(4), 519–528. doi:10.1080/02664769922395.

A. Distributions

This section contains a description (name, notation, density function, generation procedure, theoretical expectation and variance) of all the probability distributions available in the package from which a user can generate observations.

1. Laplace: $Lp(\mu, b)$.
   Density: $\frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$.
   Generation: $\mu - b.sgn\{U - \frac{1}{2}\} \ln(1 - 2|U - \frac{1}{2}|)$.
   Expectation: μ.
   Variance: $2b^2$.

2. Normal: $N(\mu, \sigma)$.
   Density: $(\sqrt{2\pi}\sigma)^{-1} \exp^{-\frac{x^2}{2\sigma^2}}$.
   Generation: $\sigma Z + \mu$.
   Expectation: μ.
   Variance: σ^2.

3. Cauchy: $Cauchy(l, s)$.
   Density: $\frac{1}{\pi s(1+(\frac{x-l}{s})^2)}$.
   Generation: `rcauchy`(l, s).
   Expectation: undefined.
   Variance: undefined.

4. Logistic: $Lg(\mu, s)$.
   Density: $\frac{1}{s} e^{-\frac{x-\mu}{s}} (1 + e^{-\frac{x-\mu}{s}})^{-2}$.
   Generation: $\mu + s \ln\left(\frac{U}{1-U}\right)$.
   Expectation: μ.
   Variance: $\frac{\pi^2}{3} s^2$.

5. Gamma: $Gamma(a, b)$.
   Density: $\frac{1}{(1/b)^a \Gamma(a)} x^{a-1} e^{-xb}, X \geq 0,\ a, b > 0$.
   Generation: `rgamma`$(a, 1/b)$.
   Expectation: $\frac{a}{b}$.
   Variance: $\frac{a}{b^2}$.

6. Beta: $Beta(\alpha, \beta)$.
   Density: $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1},\ 0 \leq x \leq 1$.
   Generation: `rbeta`(α, β).
   Expectation: $\frac{\alpha}{\alpha+\beta}$.
   Variance: $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$.

7. Uniform: $U(a, b)$.
   Density: $(b-a)^{-1}, a \leq x \leq b$.
   Generation: $\frac{U}{(b-a)} + a$.
   Expectation: $\frac{a+b}{2}$.
   Variance: $\frac{(b-a)^2}{12}$.

8. Student: *Student-t*(k).
   Density: $(\sqrt{k\pi})^{-1}\frac{\Gamma\left(\frac{k+1}{2}\right)}{\Gamma\left(\frac{k}{2}\right)}\left(1+\frac{x^2}{k}\right)^{-\frac{k+1}{2}}, k>0.$
   Generation: $\frac{Z}{\sqrt{\chi^2(k)/k}}$.
   Expectation: $k=1$: undefined, $k>1$: 0.
   Variance: $k\leq 2$: ∞, $k>2$: $\frac{k}{k-2}$.

9. Chi-squared: $\chi^2(k)$.
   Density: $2^{-k/2}\Gamma(k/2)^{-1}x^{k/2-1}e^{-x/2}, x>0.$
   Generation: $\sum_{i=1}^{k} Z_i^2$.
   Expectation: k.
   Variance: $2k$.

10. Log normal: $LN(\mu,\sigma)$.
    Density: $\frac{1}{x\sigma\sqrt{2\pi}}e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$.
    Generation: $\exp\{N(\mu,\sigma^2)\}$.
    Expectation: $e^{\mu+\sigma^2/2}$.
    Variance: $(e^{\sigma^2}-1)e^{2\mu+\sigma^2}$.

11. Weibull: $W(\lambda,k)$.
    Density: $\frac{\lambda}{k}\left(\frac{x}{k}\right)^{\lambda-1}e^{-(x/k)^{\lambda}}$.
    Generation: $k(-\ln(U))^{1/\lambda}$.
    Expectation: $\mu=k\Gamma\left(1+\lambda^{-1}\right)$.
    Variance: $k^2\Gamma\left(1+\frac{2}{\lambda}\right)-\mu^2$.

12. Shifted exponential: $SE(l,b)$.
    Density: $b\exp\{-(x-l)b\}, x\geq l$.
    Generation: $\frac{-\ln U}{b}+l$.
    Expectation: $l+\frac{1}{b}$.
    Variance: $\frac{1}{b^2}$.

13. Power uniform: U^{1+j}.
    Density: $\frac{1}{1+j}x^{-\frac{j}{j+1}}$.
    Generation: U^{1+j}.
    Expectation: $\frac{1}{j+2}$.
    Variance: $\frac{1}{2j+3}\frac{(j+1)^2}{(j+2)^2}$.

14. Average uniform: *AveUnif*(k,a,b).
    Density: $\frac{k^k}{(k-1)!}\sum_{j=0}^{\lfloor k\frac{x-a}{b-a}\rfloor}(-1)^j\binom{k}{j}(\frac{x-a}{b-a}-\frac{j}{k})^{k-1}$ for $a\leq x\leq b$.
    Generation: `mean(runif`(k,a,b)`)`.
    Expectation: $\frac{1}{2}(a+b)$.
    Variance: $\frac{1}{12k}(b-a)^2$.

15. UUniform: *UUnif*(j).
    Density: $(2(1+j))^{-1}(x^{-j/(1+j)}+(1-x)^{-j/(1+j)})$.
    Generation: $SU^{j+1}+(1-S)(1-U^{j+1})$.

Expectation: $\frac{1}{2}$.
Variance: $\frac{2j^2+3j+2}{2(2j+3)(2j+4)}$.

16. VUniform: $VUnif(j)$.
Density: $f_{14}(x-\frac{1}{2})\mathbb{1}\{x<1\} + f_{14}(x+\frac{1}{2})\mathbb{1}\{x\geq 0\}$ where f_{14} is $AveUnif(j+1,0,1)$.
Generation: if $Z_{j+1} < 0.5$: $Z_{j+1}+0.5$, else: $Z_{j+1}-0.5$, with $Z_{j+1} = AveUnif(j+1)$.
Expectation: $\frac{1}{2}$.
Variance: see Remark A.

17. Johnson SU: $JSU(\mu,\sigma,\nu,\tau)$.
Density: $\frac{1}{c\sigma\frac{1}{\tau}}\frac{1}{\sqrt{z^2+1}}\frac{1}{\sqrt{2\pi}}e^{-r^2/2}$.
Generation: $\mu + c\sigma\sqrt{w}\sinh(\omega) + c\sigma\sinh\left(\frac{1}{\tau}(Z+\nu)\right)$; $r = -\nu + \tau\sinh^{-1}(z)$; $z = \frac{x-(\mu+c\sigma\sqrt{w}\sinh(\omega))}{c\sigma}$; $c = ((w-1)(w\cosh(2\omega)+1)/2)^{-1/2}$; $w = e^{(\frac{1}{\tau})^2}$ and $\omega = -\nu\frac{1}{\tau}$.
Expectation: μ.
Variance: σ^2.

18. Symmetrical Tukey: $TU(l)$.
Density: undefined.
Generation: $\frac{U^l-(1-U)^l}{l}, -1 \leq X \leq 1$.
Expectation: 0.
Variance: $\frac{2}{l^2}\left(\frac{1}{2l+1} - \frac{\Gamma^2(l+1)}{\Gamma(2l+2)}\right)$.

19. Location contaminated: $LoConN(p,m)$.
Density: $\frac{1}{\sqrt{2\pi}}\left[pe^{-\frac{(x-m)^2}{2}} + (1-p)e^{-\frac{x^2}{2}}\right]$.
Generation: $U = \texttt{runif}(0,1)$; if$(U<p)$ $x = \texttt{rnorm}(m,1)$, otherwise $x = \texttt{rnorm}(0,1)$.
Expectation: pm.
Variance: $1-(pm)^2+pm^2$.

20. Johnson SB: $JSB(g,d)$.
Density: $\frac{d}{\sqrt{2\pi}}\frac{1}{x(1-x)}e^{-\frac{1}{2}\left(g+d\ln\frac{x}{1-x}\right)^2}$, $d>0$.
Generation: $\left(1+e^{-\frac{Z-g}{d}}\right)^{-1}, 0<X<1$.
Expectation: undefined.
Variance: undefined.

21. Skew normal: $SkewN(\xi,\omega,\alpha)$.
Density: $\left(\frac{2}{\omega}\right)\phi\left(\frac{x-\xi}{\omega}\right)\Phi\left(\alpha\left(\frac{x-\xi}{\omega}\right)\right), \omega>0$.
Generation: $\xi+\omega Y$; if$(U_0\geq 0)$ $Y = U_1$; otherwise$Y = -U_1$; U_0, V independent of $N(0,1)$; $U_1 = \delta U_0 + \sqrt{1-\delta^2}V$; $\delta = \alpha/\sqrt{1+\alpha^2}$.
Expectation: $\xi+\omega\sqrt{2/\pi}\delta$.
Variance: $\omega^2(1-2\delta^2/\pi)$.

22. Scale contaminated: $ScConN(p,d)$.
Density: $\frac{1}{\sqrt{2\pi}}\left[\frac{p}{d}e^{-\frac{x^2}{2d^2}} + (1-p)e^{-\frac{x^2}{2}}\right]$.
Generation: $U = \texttt{runif}(0,1)$; if$(U<p)$ $x = \texttt{rnorm}(0,d)$; otherwise $x = \texttt{rnorm}(0,1)$.

Expectation: 0.
Variance: $pd^2 + 1 - p$.

23. Generalized Pareto: $GP(\mu, \sigma, \xi)$.
Density: if $\xi > 0$: $\mathbb{1}[0 \leq x \leq \mu + \frac{\sigma}{\xi}]\frac{1}{\sigma}(1 - \xi\frac{x-\mu}{\sigma})^{(1-\xi)/\xi}$; else $\mathbb{1}[-\mu \leq x \leq \infty]\frac{1}{\sigma}(1 - \xi\frac{x-\mu}{\sigma})^{(1-\xi)/\xi}$.
Generation: $\mu - \frac{\sigma(U^{\xi}-1)}{\xi}$.
Expectation: $\mu + \frac{\sigma}{1+\xi}$ $(\xi < 1)$.
Variance: $\frac{\sigma^2}{(1+\xi)^2(1+2\xi)}$ $(\xi < 1/2)$.

24. Generalized error distribution: $GED(\mu, \sigma, p)$.
Density: $\frac{p}{2\sigma\Gamma(1/p)}e^{-(|x-\mu|/\sigma)^p}$.
Generation: $\mu + \sigma\left(\frac{G_p}{p}\right)^{(1/p)} sign(U - 1/2)$.
Expectation: μ.
Variance: $\frac{\sigma^2\Gamma(3/p)}{\Gamma(1/p)}$.

25. Stable: $S(\alpha, \beta, c, \mu)$.
Density: undefined, $0 < \alpha \leq 2$, $-1 \leq \beta \leq 1$, $c > 0$, $\mu \in \mathbb{R}$.
Generation: if ($\alpha = 1$ and $\beta = 0$) `tmp` = `rcauchy`$(0, 1)$, $x =$ `tmp` $\cdot\, c + \mu$; else see function `rstable()` in package **stabledist**.
Expectation: μ if $\alpha > 1$, undefined otherwise.
Variance: $2c^2$ if $\alpha = 2$, ∞ otherwise.

26. Gumbel: $Gumbel(\mu, \sigma)$.
Density: $\frac{1}{\sigma}\exp\left\{-\exp\left[-\left(\frac{x-\mu}{\sigma}\right)\right] - \left(\frac{x-\mu}{\sigma}\right)\right\}$.
Generation: $\mu - \sigma\ln(E)$.
Expectation: $\mu + \sigma(-\Gamma'(1))$.
Variance: $\frac{\pi^2}{6}\sigma^2$.

27. Frechet: $Frechet(\mu, \sigma, \alpha)$.
Density: $\frac{\alpha}{\sigma}\left(\frac{x-\mu}{\sigma}\right)_+^{-\alpha-1}\exp\left\{-\left(\frac{x-\mu}{\sigma}\right)^{-\alpha}\right\}$.
Generation: $\mu + \sigma E^{-1/\alpha}$.
Expectation: if $\alpha > 1$: $\mu + \sigma\Gamma(1 - \frac{1}{\alpha})$; else ∞.
Variance: if $\alpha > 2$: $\sigma^2(\Gamma(1 - \frac{2}{\alpha}) - (\Gamma(1 - \frac{1}{\alpha}))^2)$; else ∞.

28. Generalized extreme value: $GEV(\mu, \sigma, \xi)$.
Density: $\xi \neq 0$: $[1 + z]^{-\frac{1}{\xi}-1}\exp\left\{-[1 + z]^{-\frac{1}{\xi}}\right\}/\sigma$ with $z = \xi\frac{x-\mu}{\sigma}$, for $1 + z > 0$; $\xi = 0$: Gumbel.
Generation: if $\xi = 0$: $\mu - \sigma\ln(E)$; else: $\mu + \sigma(E^{-\xi} - 1)/\xi$.
Expectation: if $\xi \neq 0, \xi < 1$: $\mu + \sigma\frac{\Gamma(1-\xi)-1}{\xi}$; $\mu + \sigma\gamma$ if $\xi = 0$; ∞ if $\xi \geq 1$; γ: Euler constant.
Variance: if $\xi \neq 0, \xi < \frac{1}{2}$: $\sigma^2\frac{(g_2-g_1^2)}{\xi^2}$; $\sigma^2\frac{\pi^2}{6}$ if $\xi = 0$; ∞ if $\xi \geq \frac{1}{2}$; $g_k = \Gamma(1 - k\xi)$.

29. Generalized arcsine: $GArcSine(\alpha)$.
Density: $\frac{\sin(\pi\alpha)}{\pi}x^{-\alpha}(1 - x)^{\alpha-1}$ for $0 \leq x \leq 1$ and $0 < \alpha < 1$.
Generation: `rbeta`$(1 - \alpha, \alpha)$.

Expectation: $1-\alpha$.
Variance: $(1-\alpha)\alpha/2$.

30. Folded normal: $FoldN(\mu,\sigma)$.
Density: $\texttt{dnorm}(x,\mu,\sigma)+\texttt{dnorm}(-x,\mu,\sigma)$ for $x\geq 0$.
Generation: $|N(\mu,\sigma^2)|$.
Expectation: $\sigma\sqrt{\frac{2}{\pi}}e^{-\frac{\mu^2}{2\sigma^2}}+\mu[1-2\Phi(-\frac{\mu}{\sigma})]$.
Variance: $\mu^2+\sigma^2-\left\{\sigma\sqrt{\frac{2}{\pi}}e^{-\frac{\mu^2}{2\sigma^2}}+\mu[1-2\Phi(-\frac{\mu}{\sigma})]\right\}^2$.

31. Mixture normal: $MixN(p,m,d)$.
Density: $p\cdot\texttt{dnorm}(x,m,d)+(1-p)\cdot\texttt{dnorm}(x)$.
Generation: $U=\texttt{runif}(0,1)$; if$(U<p)$ $x=\texttt{rnorm}(m,d)$; else $x=\texttt{rnorm}(0,1)$.
Expectation: mp.
Variance: $(1-p)(1+pm^2)+pd^2$.

32. Truncated normal: $TruncN(a,b)$.
Density: $\frac{\exp(-x^2/2)}{\sqrt{2\pi}(\Phi(b)-\Phi(a))}\mathbb{1}[a\leq x\leq b]$.
Generation: $Z=\texttt{rnorm}(0,1)$; while$(Z<a$ or $Z>b)$ $Z=\texttt{rnorm}(0,1)$; $x=Z$.
Expectation: $\frac{\phi(a)-\phi(b)}{\Phi(b)-\Phi(a)}$.
Variance: $1+\frac{a\phi(a)-b\phi(b)}{\Phi(b)-\Phi(a)}-\left(\frac{\phi(a)-\phi(b)}{\Phi(b)-\Phi(a)}\right)^2$.

33. Normal with outliers: $Nout(a)$.
Density: undefined.
Generation: $a\in\{1,2,3,4,5\}$; $x=\texttt{rnorm}(0,1)$ with a outliers.
Expectation: 0.
Variance: 1.

34. Generalized exponential power: $GEP(t1,t2,t3)$.
Density: if $|x|\geq z_0$: $p(x;\gamma,\delta,\alpha,\beta,z_0)\propto e^{-}\delta|x|^{\gamma}|x|^{-\alpha}(log|x|)^{-\beta}$; if $|x|<z_0$: $p(x;\gamma,\delta,\alpha,\beta,z_0)$.
Generation: see function `law34.cpp` in **PoweR**.
Expectation: undefined.
Variance: undefined.

35. Exponential : $Exp(\lambda)$.
Density: $f(x)=\lambda e^{-\lambda x}$ for $x\geq 0$.
Generation: $\texttt{rexp}(\frac{1}{\lambda})$.
Expectation: $\frac{1}{\lambda}$.
Variance: $\frac{1}{\lambda^2}$.

36. Asymmetric Laplace: $ALp(\mu,b,k)$.
Density: for $x\leq\mu$: $f(x)=\frac{\sqrt{2}}{b}\frac{k}{1+k^2}\exp\left(-\frac{\sqrt{2}}{bk}|x-\mu|\right)$; for $x>\mu$: $f(x)=\frac{\sqrt{2}}{b}\frac{k}{1+k^2}\exp\left(-\frac{\sqrt{2}k}{b}|x-\mu|\right)$.
Generation: $\mu+b\log\left(\frac{\texttt{runif}(n)^k}{\texttt{runif}(n)^{1/k}}\right)/\sqrt{2}$.

Expectation: $\mu + b \cdot \frac{(\frac{1}{k} - k)}{\sqrt{2}}$.

Variance: $b^2 \frac{1 + k^4}{2k^2}$.

37. Normal-inverse Gaussian: $NIG(\alpha, \beta, \delta, \mu)$.
Density: $\frac{\alpha\delta K_1(\alpha\sqrt{\delta^2+(x-\mu)^2})}{\pi\sqrt{\delta^2+(x-\mu)^2}} e^{\delta\gamma+\beta(x-\mu)}$; $\gamma = \sqrt{\alpha^2 - \beta^2}$; K_1: Bessel function of the second kind.
Generation: see `rnig()` in package **fBasics**. Expectation: $\mu + \frac{\beta\delta}{\gamma}$. Variance: $\frac{\delta\alpha^2}{\gamma^3}$.

38. Asymmetric power distribution: $APD(\theta, \phi, \alpha, \lambda)$.
Density: `dens`.
Generation: `gen`.
Expectation: $\theta + \frac{\Gamma(\frac{2}{\lambda})}{\Gamma(\frac{1}{\lambda})}(1 - 2\alpha)\delta^{-\frac{1}{\lambda}}$; $\delta = \frac{2\alpha^\lambda(1-\alpha)^\lambda}{\alpha^\lambda+(1-\alpha)^\lambda}$.
Variance: $\phi^2[\Gamma\left(\frac{3}{\lambda}\right)\Gamma\left(\frac{1}{\lambda}\right)(1 - 3\alpha + 3\alpha^2) - \Gamma^2\left(\frac{2}{\lambda}\right)(1 - 2\alpha)^2]/[\Gamma^2\left(\frac{1}{\lambda}\right)\delta^{\frac{2}{\lambda}}]$.

39. Modified asymmetric power distribution: $modAPD(\theta_1, \theta_2, \mu, \sigma)$.
Density:

$$g(x) = \sigma^{-1} \frac{(\delta_{\boldsymbol{\theta}}/2)^{1/\theta_2}}{\Gamma(1 + 1/\theta_2)} \exp\left[- \left(\frac{2\sigma^{-1}(\delta_{\boldsymbol{\theta}}/2)^{1/\theta_2}}{1 + sign(x - \mu)(1 - 2\theta_1)} |x - \mu| \right)^{\theta_2} \right]$$

where $\boldsymbol{\theta} = (\theta_1, \theta_2)^T$ is the vector of parameters, $0 < \theta_1 < 1$, $\theta_2 > 0$ and

$$\delta_{\boldsymbol{\theta}} = \frac{2(\theta_1)^{\theta_2}(1 - \theta_1)^{\theta_2}}{(\theta_1)^{\theta_2} + (1 - \theta_1)^{\theta_2}}.$$

Generation: $\mu + \sigma 2^{1/\theta_2} \left[(\mathbb{1}\{U > \theta_1\} - \theta_1)(G_{\theta_2,1}/\delta)^{1/\theta_2} \right]$.
Expectation: $\mu + 2^{1.0/\theta_2}\sigma\Gamma(2/\theta_2)(1 - 2\theta_1)\delta^{-1/\theta_2}/\Gamma(1/\theta_2)$
Variance: $2^{2/\theta_2}\sigma^2(\Gamma(3/\theta_2)\Gamma(1/\theta_2)(1-3\theta_1+3\theta_1^2) - (\Gamma(2/\theta_2))^2(1-2\theta_1)^2)\delta^{-2/\theta_2}/(\Gamma(1/\theta_2))^2$.

Remark A.1.

- These are the references associated with some indices of statistics. 12: Kallenberg and Ledwina (1997), 13: Quesenberry and Miller (1977), 14: Quesenberry and Miller (1977) Bates (1955), 15,16: Quesenberry and Miller (1977), 17: Johnson (1949), 18: Kallenberg and Ledwina (1997), 19: Kallenberg and Ledwina (1997), 20: Kallenberg and Ledwina (1997), 21: Azzalini (2005), 22: Kallenberg and Ledwina (1997), 23: Coles (2001), 24: Nadarajah (2005), 25: Chambers, Mallows, and Stuck (1976), 26: Coles (2001), 27: Castillo, Hadi, Balakrishnan, and Sarabia (2005), 28: Coles (2001), 29: Feller (1968, 1971), 30: Leone, Nelson, and Nottingham (1961), 31,32,33: Romão *et al.* (2010), 34: Desgagné and Angers (2005), 36: Kotz, Kozubowski, and Podgórski (2001), 37: Atkinson (1982), 39: Desgagné and Lafaye de Micheaux (2016).

- U means a $Uniform(0, 1)$ distribution, E is an *Exponential* distribution, U and E are independent.

- S is for a law independent from U such that $\mathsf{P}[S = 0] = \mathsf{P}[S = 1] = 1/2$.

- Z stands for the *Gaussian* law and G_p represents the $Gamma(1/p, p)$ law while $G_{p,1}$ stands for the $Gamma(1/p, 1)$ law.

- *Average Uniform* law is also called $Bates(k, a, b)$. In Quesenberry and Miller (1977), it is $AveUnif(k + 1, 0, 1)$.

- We go from *Generalized Pareto*(μ, σ, ξ) to *Pareto*(a, k) by letting $\mu = k$, $\xi = a^{-1}$ and $\sigma = ka^{-1}$.

- We go from *Generalized Pareto*(μ, σ, ξ) to a shifted Pareto by letting $\mu = 0$, $\xi = 1/2$ and $\sigma = 1/2$.

- We go from $JSU(\mu, \sigma, \nu, \tau)$ to $JSB(g, d)$ by letting $\tau = d$, $\nu = -g$, $\sigma = c^{-1}$ $= \left[(e^{d^2} - 1)(e^{d^2}\cosh(2g/d) + 1)/2\right]^{-1/2}$ and $\mu = -\sqrt{e^{d^2}}\sinh(g/d)$.

- We go from $GED(\mu, \sigma, p)$ to $GED(\lambda)$ by letting $\mu = 0$, $p = \lambda$ and $\sigma = \dfrac{1}{\lambda^{1/\lambda}\sigma}$ with $C_\lambda = \sqrt{\Gamma(3\lambda^{-1})/\Gamma(\lambda^{-1})}$.

- Variance of $VUnif(j)$ is given by:

$$\mathsf{VAR}(Y_j) = \frac{1}{12(j+1)} - \frac{1}{4} + \frac{1}{(j+1)!}\sum_{k=0}^{j+1}(-1)^k\binom{j+1}{k}\cdot$$

$$\left\{(-1)^{j+1}\frac{k^{j+2}}{(j+1)(j+2)} - \text{sign}(k - \frac{j+1}{2})\left(\frac{j+1}{2} - k\right)^{(j+1)}\left[\frac{1}{j+2}\left(\frac{j+1}{2} - k\right) + \frac{k}{j+1}\right]\right\}.$$

  where $sign(0) = -1$.

B. Tests

Table 4 contains a list of non-normality tests included in the package. The order of presentation is chronological. It also contains references to definitions of these tests.

Table 5 contains a list of uniformity tests as well as associated references.

Table 6 contains a list of tests for the Laplace distribution. It also contains references to definitions of these tests.

	Test	Reference
1	Lilliefors $K - S$	Lilliefors (1967)
2	Anderson-Darling AD^*	D'Agostino and Stephens (1986)
3	Zhang-Wu Z_C	Zhang and Wu (2005)
4	Zhang-Wu Z_A	Zhang and Wu (2005)
5	Glen-Leemis-Barr P_S	Glen, Leemis, and Barr (2001)
6	D'Agostino-Pearson K^2	D'Agostino and Pearson (1973, 1974)
7	Jarque-Bera JB	Jarque and Bera (1987)
8	Doornik-Hansen DH	Doornik and Hansen (2008)
9	Gel-Gastwirth RJB	Gel and Gastwirth (2008)
10	Hosking T_{Lmom}	Hosking (1990)
11	Hosking $T^{(1)}_{Lmom}$	Hosking (1990)
12	Hosking $T^{(2)}_{Lmom}$	Hosking (1990)
13	Hosking $T^{(3)}_{Lmom}$	Hosking (1990)
14	Bontemps-Meddahi BM_{3-4}	Bontemps and Meddahi (2005)
15	Bontemps-Meddahi BM_{3-6}	Bontemps and Meddahi (2005)
16	Brys-Hubert-Struyf T_{MC-LR}	Brys, Hubert, and Struyf (2008)
17	Bonett-Seier T_w	Bonett and Seier (2002)
18	Combination of T_{MC-LR} & T_w	Brys *et al.* (2008), Bonett and Seier (2002)
19	Cabana-Cabana $T_{S,l}$	Cabaña and Cabaña (1994)
20	Cabana-Cabana $T_{K,l}$	Cabaña and Cabaña (1994)
21	Shapiro-Wilk W	Shapiro and Wilk (1965)
22	Shapiro-Francia W'	Shapiro and Francia (1972)
23	modified Shapiro-Wilk $\tilde{W}$	Rahman and Govindarajulu (1997)
24	D'Agostino D	D'Agostino (1971)
25	Filliben r	Filliben (1975)
26	Chen-Shapiro CS	Chen and Shapiro (1995)
27	Zhang Q	Zhang (1999)
28	Zhang $Q - Q^*$	Zhang (1999)
29	Barrio-Cuesta Albertos-Matran-Rodriguez $BCMR$	del Barrio *et al.* (1999)
30	Coin β_3^2	Coin (2008)
31	Epps-Pulley $T^*(\alpha)$	Epps and Pulley (1983)
32	Martinez-Iglewicz I_n	Martinez and Iglewicz (1981)
33	Gel-Miao-Gastwirth R_{sJ}	Gel, Miao, and Gastwirth (2007)
34	Zhang Q^*	Zhang (1999)
35	Desgagné-Lafaye de Micheaux-Leblanc P_1	Desgagné *et al.* (2009)
36	Desgagné-Lafaye de Micheaux-Leblanc P_2	Desgagné *et al.* (2009)
37	Desgagné-Lafaye de Micheaux-Leblanc S_2	Desgagné and Lafaye de Micheaux (2016)
38	Desgagné-Lafaye de Micheaux-Leblanc K_2	Desgagné and Lafaye de Micheaux (2016)
39	Desgagné-Lafaye de Micheaux-Leblanc X_{APD}	Desgagné and Lafaye de Micheaux (2016)
40	Desgagné-Lafaye de Micheaux-Leblanc	Desgagné *et al.* (2013)
41	Spiegelhalter S	Spiegelhalter (1977)

Table 4: List of non-normality tests.

	Test	Reference
63	Kolmogorov D_n	Kolmogorov (1933)
64	Cramér-von Mises W_n^2	Anderson and Darling (1954)
65	Anderson-Darling A_n^2	Anderson and Darling (1954)
66	Durbin C_n	Durbin (1969)
67	Kuiper K_n	Brunk (1962)
68	Hegazy-Green T_1	Hegazy and Green (1975)
69	Hegazy-Green T_2	Hegazy and Green (1975)
70	Greenwood $G(n)$	Greenwood (1946)
71	Quesenberry-Miller Q	Quesenberry and Miller (1977)
72	Read-Cressie $2nI^\lambda$	Read and Cressie (1988)
73	Moran $M(n)$	Moran (1951)
74	Cressie $L_n^{(m)}$	Cressie (1978)
75	Cressie $S_n^{(m)}$	Cressie (1979)
76	Vasicek $H(m, n)$	Vasicek (1976)
77	Swartz $A^*(n)$	Swartz (1992)
78	Morales $D_{n,m}(\phi_\lambda)$	Morales, Pardo, Pardo, and Vajda (2003)
79	Pardo $E_{m,n}$	Pardo (2003)
80	Marhuenda $T_{n,m}^\lambda$	Marhuenda, Marhuenda, and Morales (2005)
81	Zhang Z_A	Zhang (2002)
82	Zhang Z_C	Zhang (2002)

Table 5: List of uniformity tests.

	Test	Reference
42	Anderson-Darling A^2	Yen and Moore (1988)
43	Cramér-von Mises W^2	Yen and Moore (1988)
44	Watson U^2	Puig and Stephens (2000)
45	Kolmogorov-Smirnov $\sqrt{n}D$	Puig and Stephens (2000)
46	Kuiper V	Puig and Stephens (2000)
47	Meintanis $T_{n,a}^{(1)}$ - MO	Meintanis (2004)
48	Meintanis $T_{n,a}^{(1)}$ - ML	Meintanis (2004)
49	Meintanis $T_{n,a}^{(2)}$ - MO	Meintanis (2004)
50	Meintanis $T_{n,a}^{(2)}$ - ML	Meintanis (2004)
51	Choi-Kim $T_{m,n}^V$	Choi and Kim (2006)
52	Choi-Kim $T_{m,n}^E$	Choi and Kim (2006)
53	Choi-Kim $T_{m,n}^C$	Choi and Kim (2006)
54	Desgagné-Lafaye de Micheaux-Leblanc $\hat{G}_n$	Desgagné, Lafaye de Micheaux, and Leblanc (2014)
55	Rayner-Best V_3	Rayner and Best (1989)
56	Rayner-Best V_4	Rayner and Best (1989)
57	Langholz-Kronmal K_1	Langholz and Kronmal (1991)
58	Kundu T	Kundu (2005)
59	Gulati Z	Gulati (2011)
60	Gel K	Gel (2010)
61	Lafaye de Micheaux LM	Desgagné *et al.* (2014)

Table 6: List of tests for a Laplace distribution.

Affiliation:

Pierre Lafaye de Micheaux
Departement de Mathématiques et Statistique
Université de Montréal
Montreal, Canada
E-mail: lafaye@dms.umontreal.ca
URL: http://biostatisticien.eu/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
February 2016, Volume 69, Issue 3 *Submitted:* 2013-08-27
doi:10.18637/jss.v069.i03 *Accepted:* 2015-01-27