Package 'KScorrect'

July 3, 2019

Type Package

Title Lilliefors-Corrected Kolmogorov-Smirnov Goodness-of-Fit Tests

Version 1.4.0

Depends R (>= 3.6.0)

Imports MASS (>= 7.3.0), doParallel (>= 1.0.14), foreach (>= 1.4.4), iterators (>= 1.0.10), parallel (>= 3.6.0), mclust (>= 5.4)

Date 2019-06-30

Author Phil Novack-Gottshall, Steve C. Wang

Maintainer Phil Novack-Gottshall <pnovack-gottshall@ben.edu>

Description Implements the Lilliefors-corrected Kolmogorov-Smirnov test for use in goodness-of-fit tests, suitable when population parameters are unknown and must be estimated by sample statistics. P-values are estimated by simulation. Can be used with a variety of continuous distributions, including normal, lognormal, univariate mixtures of normals, uniform, loguniform, exponential, gamma, and Weibull distributions. Functions to generate random numbers and calculate density, distribution, and quantile functions are provided for use with the log uniform and mixture distributions.

License CC0

URL https://github.com/pnovack-gottshall/KScorrect

BugReports https://github.com/pnovack-gottshall/KScorrect/issues

LazyData TRUE

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2019-07-03 19:30:03 UTC

R topics documented:

KScorrect-package *KScorrect: Lilliefors-Corrected Kolmogorov-Smirnov Goodness-of-Fit Tests*

Description

Implements the Lilliefors-corrected Kolmogorov-Smirnov test for use in goodness-of-fit tests.

Details

KScorrect implements the Lilliefors-corrected Kolmogorov-Smirnov test for use in goodness-of-fit tests, suitable when population parameters are unknown and must be estimated by sample statistics. *P*-values are estimated by simulation. Coded to complement `ks.test`, it can be used with a variety of continuous distributions, including normal, lognormal, univariate mixtures of normals, uniform, loguniform, exponential, gamma, and Weibull distributions.

Functions to generate random numbers and calculate density, distribution, and quantile functions are provided for use with the loguniform and mixture distributions.

Author(s)

Phil Novack-Gottshall <pnovack-gottshall@ben.edu>

Steve C. Wang <scwang@swarthmore.edu>

Examples

```
# Get the package version and citation of KScorrect
packageVersion("KScorrect")
citation("KScorrect")

x <- runif(200)
Lc <- LcKS(x, cdf="pnorm", nreps=999)
hist(Lc$D.sim)
abline(v = Lc$D.obs, lty = 2)
print(Lc, max=50)  # Print first 50 simulated statistics
# Approximate p-value (usually) << 0.05

# Confirmation uncorrected version has increased Type II error rate when
#   using sample statistics to estimate parameters:
ks.test(x, "pnorm", mean(x), sd(x))   # p-value always larger, (usually) > 0.05
```

```
x <- rlunif(200, min=exp(1), max=exp(10)) # random loguniform sample
Lc <- LcKS(x, cdf="plnorm")
Lc$p.value      # Approximate p-value: (usually) << 0.05
```

---

dlunif *The Log Uniform Distribution*

Description

Density, distribution function, quantile function and random generation for the log uniform distribution in the interval from `min` to `max`. Parameters must be raw values (not log-transformed) and will be log-transformed using specified `base`.

Usage

```
dlunif(x, min, max, base = exp(1))

plunif(q, min, max, base = exp(1))

qlunif(p, min, max, base = exp(1))

rlunif(n, min, max, base = exp(1))
```

Arguments

`x`	Vector of quantiles.
`min`	Lower limit of the distribution, in raw (not log-transformed) values. Negative values will give warning.
`max`	Upper limit of the distribution, in raw (not log-transformed) values. Negative values will give warning.
`base`	The base to which logarithms are computed. Defaults to `e=exp(1)`. Must be a positive number.
`q`	Vector of quantiles.
`p`	Vector of probabilities.
`n`	Number of observations.

Details

A log uniform (or loguniform or log-uniform) random variable has a uniform distribution when log-transformed.

Value

`dlunif` gives the density, `plunif` gives the distribution function, `qlunif` gives the quantile function, and `rlunif` generates random numbers.

Note

Parameters `min, max` must be provided as raw (not log-transformed) values and will be log-transformed using `base`. In other words, when log-transformed, a log uniform random variable with parameters `min=a` and `max=b` is uniform over the interval from `log(a)` to `log(b)`.

Author(s)

Steve Wang <scwang@swarthmore.edu>

See Also

`Distributions` for other standard distributions

Examples

```
plot(1:100, dlunif(1:100, exp(1), exp(10)), type="l", main="Loguniform density")
plot(log(1:100), dlunif(log(1:100), log(1), log(10)), type="l",
     main="Loguniform density")

plot(1:100, plunif(1:100, exp(1), exp(10)), type="l", main="Loguniform cumulative")
plot(qlunif(ppoints(100), exp(1), exp(10)), type="l", main="Loguniform quantile")

hist(rlunif(1000, exp(1), exp(10)), main="random loguniform sample")
hist(log(rlunif(10000, exp(1), exp(10))), main="random loguniform sample")
hist(log(rlunif(10000, exp(1), exp(10), base=10), base=10), main="random loguniform sample")
```

`dmixnorm` *The Normal Mixture Distribution*

Description

Density, distribution function, quantile function, and random generation for a univariate (one-dimensional) distribution composed of a mixture of normal distributions with means equal to `mean`, standard deviations equal to `sd`, and mixing proportion of the components equal to `pro`.

Usage

```
dmixnorm(x, mean, sd, pro)

pmixnorm(q, mean, sd, pro)

qmixnorm(p, mean, sd, pro, expand = 1)

rmixnorm(n, mean, sd, pro)
```

Arguments

x	Vector of quantiles.
mean	Vector of means, one for each component.
sd	Vector of standard deviations, one for each component. If a single value is provided, an equal-variance mixture model is implemented. Must be non-negative.
pro	Vector of mixing proportions, one for each component. If missing, an equal-proportion model is implemented, with a warning. If proportions do not sum to unity, they are rescaled to do so. Must be non-negative.
q	Vector of quantiles.
p	Vector of probabilities.
expand	Value to expand the range of probabilities for quantile approximation. Default = 1.0. See details below.
n	Number of observations.

Details

These functions use, modify, and wrap around those from the mclust package, especially dens, and sim. Functions are slightly faster than the corresponding mclust functions when used with univariate distributions.

Unlike mclust, which primarily focuses on parameter estimation based on mixture samples, the functions here are modified to calculate PDFs, CDFs, approximate quantiles, and random numbers for mixture distributions with user-specified parameters. The functions are written to emulate the syntax of other R distribution functions (e.g., Normal).

The number of mixture components (argument G in mclust) is specified from the length of the mean vector. If a single sd value is provided, an equal-variance mixture model (modelNames="E" in mclust) is implemented; if multiple values are provided, a variable-variance model (modelNames="V" in mclust) is implemented. If mixing proportion pro is missing, all components are assigned equal mixing proportions, with a warning. Mixing proportions are rescaled to sum to unity. If the lengths of supplied means, standard deviations, and mixing proportions conflict, an error is called.

Analytical solutions are not available to calculate a quantile function for all combinations of mixture parameters. qmixnorm approximates the quantile function using a spline function calculated from cumulative density functions for the specified mixture distribution. Quantile values for probabilities near zero and one are approximated by taking a randomly generated sample (with sample size equal to the product of 1000 and the number of mixture components), and expanding that range positively and negatively by a multiple (specified by (default) expand = 1) of the observed range in the random sample. In cases where the distribution range is large (such as when mixture components are discrete or there are large distances between components), resulting extreme probability values will be very close to zero or one and can result in non-calculable (NaN) quantiles (and a warning). Use of other expand values (especially expand < 1.0 that expand the ranges by smaller multiples) often will yield improved approximations. Note that expand values equal to or close to 0 may result in inaccurate approximation of extreme quantiles. In situations requiring extreme quantile values, it is recommended that the largest expand value that does not result in a non-calculable quantile (i.e., no warning called) be used. See examples for confirmation that approximations are accurate, comparing the approximate quantiles from a single 'mixture' distribution to those calculated for the same distribution using qnorm, and demonstrating cases in which using non-default expand values will allow correct approximation of quantiles.

Value

dmixnorm gives the density, pmixnorm gives the distribution function, qmixnorm approximates the quantile function, and rmixnorm generates random numbers.

Author(s)

Phil Novack-Gottshall <pnovack-gottshall@ben.edu> and Steve Wang <scwang@swarthmore.edu>, based on functions written by Luca Scrucca.

See Also

Distributions for other standard distributions, and mclust::dens, sim, and cdfMclust for alternative density, quantile, and random number functions for multivariate mixture distributions.

Examples

```
# Mixture of two normal distributions
mean <- c(3, 6)
pro <- c(.25, .75)
sd <- c(.5, 1)
x <- rmixnorm(n=5000, mean=mean, pro=pro, sd=sd)
hist(x, n=20, main="random bimodal sample")

## Not run:
# Requires functions from the 'mclust' package
require(mclust)
# Confirm 'rmixnorm' above produced specified model
mod <- mclust::Mclust(x)
mod             # Best model (correctly) has two-components with unequal variances
mod$parameters # and approximately same parameters as specified above
sd^2            # Note reports var (sigma-squared) instead of sd used above

## End(Not run)

# Density, distribution, and quantile functions
plot(seq(0, 10, .1), dmixnorm(seq(0, 10, .1), mean=mean, sd=sd, pro=pro),
     type="l", main="Normal mixture density")
plot(seq(0, 10, .1), pmixnorm(seq(0, 10, .1), mean=mean, sd=sd, pro=pro),
     type="l", main="Normal mixture cumulative")
plot(stats::ppoints(100), qmixnorm(stats::ppoints(100), mean=mean, sd=sd, pro=pro),
     type="l", main="Normal mixture quantile")

# Any number of mixture components are allowed
plot(seq(0, 50, .01), pmixnorm(seq(0, 50, .01), mean=1:50, sd=.05, pro=rep(1, 50)),
     type="l", main="50-component normal mixture cumulative")

# 'expand' can be specified to prevent non-calculable quantiles:
q1 <- qmixnorm(stats::ppoints(30), mean=c(1, 20), sd=c(1, 1), pro=c(1, 1))
q1 # Calls a warning because of NaNs
# Reduce 'expand'. (Values < 0.8 allow correct approximation)
q2 <- qmixnorm(stats::ppoints(30), mean=c(1, 20), sd=c(1, 1), pro=c(1, 1), expand=.5)
plot(stats::ppoints(30), q2, type="l", main="Quantile with reduced range")
```

```
## Not run:
# Requires functions from the 'mclust' package
# Confirmation that qmixnorm approximates correct solution
#   (single component 'mixture' should mimic qnorm):
x <- rmixnorm(n=5000, mean=0, pro=1, sd=1)
mpar <- mclust::Mclust(x)$param
approx <- qmixnorm(p=ppoints(100), mean=mpar$mean, pro=mpar$pro,
     sd=sqrt(mpar$variance$sigmasq))
known <- qnorm(p=ppoints(100), mean=mpar$mean, sd=sqrt(mpar$variance$sigmasq))
cor(approx, known)  # Approximately the same
plot(approx, main="Quantiles for (unimodal) normal")
lines(known)
legend("topleft", legend=c("known", "approximation"), pch=c(NA,1),
     lty=c(1, NA), bty="n")

## End(Not run)
```

`ks_test_stat` *Internal KScorrect Function.*

Description

Internal function not intended to be called directly by users.

Usage

```
ks_test_stat(x, y, ...)
```

Arguments

x	a numeric vector of data values.
y	a character string naming a cumulative distribution function or an actual cumulative distribution function such as pnorm. Only continuous CDFs are valid. See /codeLcKS for accepted functions.
...	parameters of the distribution specified (as a character string) by y.

Details

Simplified and faster `ks.test` function that calculates just the two-sided test statistic D.

Note

Calculating the Kolmogorov-Smirnov test statistic D by itself is faster than calculating the other ouput that that function produces.

See Also

`ks.test`

LcKS *Lilliefors-corrected Kolmogorov-Smirnov Goodness-Of-Fit Test*

Description

Implements the Lilliefors-corrected Kolmogorov-Smirnov test for use in goodness-of-fit tests, suitable when population parameters are unknown and must be estimated by sample statistics. It uses Monte Carlo simulation to estimate *p*-values. Using a modification of `ks.test`, it can be used with a variety of continuous distributions, including normal, lognormal, univariate mixtures of normals, uniform, loguniform, exponential, gamma, and Weibull distributions. The Monte Carlo algorithm can run 'in parallel.'

Usage

```
LcKS(x, cdf, nreps = 4999, G = 1:9, varModel = c("E", "V"),
  parallel = FALSE, cores = NULL)
```

Arguments

`x`	A numeric vector of data values (observed sample).
`cdf`	Character string naming a cumulative distribution function. Case insensitive. Only continuous CDFs are valid. Allowed CDFs include: • `"pnorm"` for normal, • `"pmixnorm"` for (univariate) normal mixture, • `"plnorm"` for lognormal (log-normal, log normal), • `"punif"` for uniform, • `"plunif"` for loguniform (log-uniform, log uniform), • `"pexp"` for exponential, • `"pgamma"` for gamma, • `"pweibull"` for Weibull.
`nreps`	Number of replicates to use in simulation algorithm. `Default = 4999` replicates. See `details` below. Should be a positive integer.
`G`	Numeric vector of mixture components to consider, for mixture models only. `Default = 1:9` fits up to 9 components. Must contain positive integers. See `details` below.
`varModel`	For mixture models, character string determining whether to allow equal-variance mixture components (`E`), variable-variance mixture components (`V`) or both (the `default`).
`parallel`	Logical value that switches between running Monte Carlo algorithm in parallel (if `TRUE`) or not (if `FALSE`, the `default`).
`cores`	Numeric value to control how many cores to build when running in parallel. `Default = detectCores - 1`.

Details

The function builds a simulation distribution D.sim of length nreps by drawing random samples from the specified continuous distribution function cdf with parameters calculated from the provided sample x. Observed statistic D and simulated test statistics are calculated using a simplified version of ks.test.

The default nreps = 4999 provides accurate *p*-values. nreps = 1999 is sufficient for most cases, and computationally faster when dealing with more complicated distributions (such as univariate normal mixtures, gamma, and Weibull). See below for potentially faster parallel implementations.

The *p*-value is calculated as the number of Monte Carlo samples with test statistics *D* as extreme as or more extreme than that in the observed sample D.obs, divided by the nreps number of Monte Carlo samples. A value of 1 is added to both the numerator and denominator to allow the observed sample to be represented within the null distribution (Manly 2004); this has the benefit of avoiding nonsensical p.value = 0.000 and accounts for the fact that the *p*-value is an estimate.

Parameter estimates are calculated for the specified continuous distribution, using maximum-likelihood estimates. When testing against the gamma and Weibull distributions, MASS::fitdistr is used to calculate parameter estimates using maximum likelihood optimization, with sensible starting values. Because this incorporates an optimization routine, the simulation algorithm can be slow if using large nreps or problematic samples. Warnings often occur during these optimizations, caused by difficulties estimating sample statistic standard errors. Because such SEs are not used in the Lilliefors-corrected simulation algorithm, warnings are suppressed during these optimizations.

Sample statistics for the (univariate) normal mixture distribution pmixnorm are calculated using package mclust, which uses BIC to identify the optimal mixture model for the sample, and the EM algorithm to calculate parameter estimates for this model. The number of mixture components G (with default allowing up to 9 components), variance model (whether equal E or variable V variance), and component statistics (means, sds, and mixing proportions pro) are estimated from the sample when calculating D.obs and passed internally when creating random Monte Carlo samples. It is possible that some of these samples may differ in their optimal G (for example a two-component input sample might yield a three-component random sample within the simulation distribution). This can be constrained by specifying that simulation BIC-optimizations only consider G mixture components.

Be aware that constraining G changes the null hypothesis. The default (G = 1:9) null hypothesis is that a sample was drawn from *any* G = 1:9-*component mixture distribution*. Specifying a particular value, such as G = 2, restricts the null hypothesis to particular mixture distributions with just G components, even if simulated samples might better be represented as different mixture models.

The LcKS(cdf = "pmixnorm") test implements two control loops to avoid errors caused by this constraint and when working with problematic samples. The first loop occurs during model-selection for the observed sample x, and allows for estimation of parameters for the second-best model when those for the optimal model are not able to be calculated by the EM algorithm. A second loop occurs during the simulation algorithm, rejecting samples that cannot be fit by the mixture model specified by the observed sample x. Such problematic cases are most common when the observed or simulated samples have a component(s) with very small variance (i.e., duplicate observations) or when a Monte Carlo sample cannot be fit by the specified G.

Parellel computing can be implemented using parallel = TRUE, using the operating-system versatile doParallel-package and foreach infrastructure, using a default detectCores - 1 number of cores. Parallel computing is generally advisable for the more complicated cumulative density

functions (i.e., univariate normal mixture, gamma, Weibull), where maximum likelihood estimation is time-intensive, but is generally not advisable for density functions with quickly calculated sample statistics (i.e., other distribution functions). Warnings within the function provide sensible recommendations, but users are encouraged to experiment to discover their fastest implementation for their individual cases.

Value

A list containing the following components:

`D.obs`	The value of the test statistic *D* for the observed sample.
`D.sim`	Simulation distribution of test statistics, with `length = nreps`. This can be used to calculate critical values; see examples.
`p.value`	*p*-value of the test, calculated as $(\sum(D.sim > D.obs) + 1)/(nreps + 1)$.

Note

The Kolmogorov-Smirnov (such as `ks.test`) is only valid as a goodness-of-fit test when the population parameters are known. This is typically not the case in practice. This invalidation occurs because estimating the parameters changes the null distribution of the test statistic; i.e., using the sample to estimate population parameters brings the Kolmogorov-Smirnov test statistic *D* closer to the null distribution than it would be under the hypothesis where the population parameters are known. In other words, it is biased and results in increased Type II error rates. Lilliefors (1967, 1969) provided a solution, using Monte Carlo simulation to approximate the shape of the null distribution when the sample statistics are used to estimate population parameters, and to use this null distribution as the basis for critical values. The function `LcKS` generalizes this solution for a range of continuous distributions.

Author(s)

Phil Novack-Gottshall <pnovack-gottshall@ben.edu>, based on code from Charles Geyer (University of Minnesota).

References

Lilliefors, H. W. 1967. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association* 62(318):399-402.

Lilliefors, H. W. 1969. On the Kolmogorov-Smirnov test for the exponential distribution with mean unknown. *Journal of the American Statistical Association* 64(325):387-389.

Manly, B. F. J. 2004. *Randomization, Bootstrap and Monte Carlo Methods in Biology*. Chapman & Hall, Cornwall, Great Britain.

Parsons, F. G., and P. H. Wirsching. 1982. A Kolmogorov-Smirnov goodness-of-fit test for the two-parameter Weibull distribution when the parameters are estimated from the data. *Microelectronics Reliability* 22(2):163-167.

See Also

Distributions for standard cumulative distribution functions, plunif for the loguniform cumulative distribution function, and pmixnorm for the univariate normal mixture cumulative distribution function.

Examples

```
x <- runif(200)
Lc <- LcKS(x, cdf = "pnorm", nreps = 999)
hist(Lc$D.sim)
abline(v = Lc$D.obs, lty = 2)
print(Lc, max = 50)  # Print first 50 simulated statistics
# Approximate p-value (usually) << 0.05

# Confirmation uncorrected version has increased Type II error rate when
#   using sample statistics to estimate parameters:
ks.test(x, "pnorm", mean(x), sd(x))   # p-value always larger, (usually) > 0.05

# Confirm critical values for normal distribution are correct
nreps <- 9999
x <- rnorm(25)
Lc <- LcKS(x, "pnorm", nreps = nreps)
sim.Ds <- sort(Lc$D.sim)
crit <- round(c(.8, .85, .9, .95, .99) * nreps, 0)
# Lilliefors' (1967) critical values, using improved values from
#   Parsons & Wirsching (1982) (for n = 25):
# 0.141 0.148 0.157 0.172 0.201
round(sim.Ds[crit], 3) # Approximately the same critical values

# Confirm critical values for exponential are the same as reported by Lilliefors (1969)
nreps <- 9999
x <- rexp(25)
Lc <- LcKS(x, "pexp", nreps = nreps)
sim.Ds <- sort(Lc$D.sim)
crit <- round(c(.8, .85, .9, .95, .99) * nreps, 0)
# Lilliefors' (1969) critical values (for n = 25):
# 0.170 0.180 0.191 0.210 0.247
round(sim.Ds[crit], 3) # Approximately the same critical values

## Not run:
# Gamma and Weibull tests require functions from the 'MASS' package
# Takes time for maximum likelihood optimization of statistics
require(MASS)
x <- runif(100, min = 1, max = 100)
Lc <- LcKS(x, cdf = "pgamma", nreps = 499)
Lc$p.value

# Confirm critical values for Weibull the same as reported by Parsons & Wirsching (1982)
nreps <- 9999
x <- rweibull(25, shape = 1, scale = 1)
Lc <- LcKS(x, "pweibull", nreps = nreps)
sim.Ds <- sort(Lc$D.sim)
```

```
crit <- round(c(.8, .85, .9, .95, .99) * nreps, 0)
# Parsons & Wirsching (1982) critical values (for n = 25):
# 0.141 0.148 0.157 0.172 0.201
round(sim.Ds[crit], 3) # Approximately the same critical values

# Mixture test requires functions from the 'mclust' package
# Takes time to identify model parameters
require(mclust)
x <- rmixnorm(200, mean = c(10, 20), sd = 2, pro = c(1,3))
Lc <- LcKS(x, cdf = "pmixnorm", nreps = 499, G = 1:9)   # Default G (1:9) takes long time
Lc$p.value
G <- Mclust(x)$parameters$variance$G             # Optimal model has only two components
Lc <- LcKS(x, cdf = "pmixnorm", nreps = 499, G = G)    # Restricting to likely G saves time
# But note changes null hypothesis: now testing against just two-component mixture
Lc$p.value

# Running 'in parallel'
require(doParallel)
set.seed(3124)
x <- rmixnorm(300, mean = c(110, 190, 200), sd = c(3, 15, .1), pro = c(1, 3, 1))
system.time(LcKS(x, "pgamma"))
system.time(LcKS(x, "pgamma", parallel = TRUE)) # Should be faster

## End(Not run)
```

Index