Package 'fgof'

November 24, 2026

Type Package

Title Fast Goodness-of-Fit Test

Version 0.3-0

Date 2026-11-18

Imports mvtnorm, numDeriv

Suggests testthat (>= 3.0.0), roxygen2

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Description Goodness-of-fit test with multiplier or parametric bootstrap for continuous distributions with estimated parameters. Implements the procedures of Kojadinovic and Yan (2012) <doi:10.1002/cjs.11135>.

License GPL (>= 3)

Encoding UTF-8

URL https://github.com/jun-yan/fgof

BugReports https://github.com/jun-yan/fgof/issues

NeedsCompilation no

Author Ivan Kojadinovic [aut], Jun Yan [aut, cre]

Maintainer Jun Yan <jun.yan@uconn.edu>

Repository CRAN

Date/Publication 2026-11-24 17:10:02 UTC

Contents

gof.test 2

Index **6**

`gof.test` *One Sample Goodness-of-fit Test*

Description

Goodness-of-fit testing with multiplier or parametric bootstrap.

Usage

```
gof.test(
  x,
  distr,
  nsim,
  start,
  simulation = c("mult", "pb"),
  ng = 1000,
  qgrid = c("empirical", "fitted"),
  gridStat = NULL,
  numDeriv.args = gradControl(),
  derCdfWrtPar = NULL,
  derLogPdfWrtPar = NULL,
  distr.args = list(),
  optim.args = list(),
  refit.failure = c("stop", "report")
)
```

Arguments

`x`	A nonempty numeric vector or matrix of finite observations.
`distr`	Name of the hypothesized distribution.
`nsim`	Positive integer giving the number of bootstrap replicates.
`start`	Nonempty named list of starting values for parameters to estimate.
`simulation`	Bootstrap method: multiplier (‘mult‘) or parametric (‘pb‘).
`ng`	Target number of grid points.
`qgrid`	Grid construction method: empirical or fitted quantiles.
`gridStat`	Whether to compute grid-based statistics in addition to observation-based statistics.
`numDeriv.args`	Numerical Jacobian controls passed to `numDeriv`. Omission selects automatic step checks; explicit values are forwarded unchanged.
`derCdfWrtPar`	Optional CDF derivative callback.
`derLogPdfWrtPar`	Optional log-density derivative callback.
`distr.args`	Named list of fixed arguments passed to the distribution functions.

`optim.args` Named list of arguments passed to `stats::optim`, such as `method`, `lower`, `upper`, and `control`.

`refit.failure` Action after a parametric bootstrap refit fails. Use `"stop"` (default) or `"report"` to attempt all replicates and return failure details. Ignored for the multiplier bootstrap.

Details

Fixed distribution parameters belong in `distr.args`; optimizer settings belong in `optim.args`. Both are named lists. For example, use `distr.args = list(sd = 2)` to fix a normal standard deviation and `optim.args = list(method = "BFGS")` to choose an optimizer method. Parameters listed in `start` are estimated and must not also appear in `distr.args`. Observations are assumed independent and identically distributed from a continuous distribution. Parameters in `start` are fitted by maximum likelihood. The optimizer defaults to Nelder-Mead, or L-BFGS-B when bounds are supplied. Choose appropriate starting values and bounds for the model; numerical derivatives may fail near parameter boundaries, and the multiplier procedure requires an invertible score covariance matrix.

Automatic numerical differentiation applies to standard and user-defined distributions alike. Parameter scales are inferred from two-sided changes in the fitted log density, or taken from the absolute optimizer `control$parscale` values supplied in `optim.args`. These are numerical scales, not parameter standard errors. A separate feasible radius records finite log-density probes; optimizer bounds do not by themselves define this radius. Candidate steps cannot exceed either that radius or 10 percent of the scale. Representable power-of-two steps are compared independently for log-density and CDF Jacobians. Three successive estimates must agree componentwise in scaled units, with absolute tolerance `1e-7` and relative tolerance `1e-5`. Richardson extrapolation retains six levels. A previous default derivative is kept when it meets these checks.

Selection stops with an actionable error if no reliable two-sided stencil is found or repeated function values reveal numerical randomness. It does not silently fall back to unchecked derivatives. Functions used for automatic selection must be deterministic and free of side effects. Approximate CDF integration, weak parameter sensitivity, constraints, and nonsmooth models can require analytical callbacks or explicitly selected numerical controls. Agreement between finite differences does not establish model regularity. No automatic selection is performed for parametric bootstrap. Analytical callbacks replace their corresponding derivatives; automatic checks continue for any remaining numerical Jacobian.

The parametric bootstrap simulates from the fitted model and refits every replicate. If L-BFGS-B reports an abnormal line-search termination, the refit is retried once on the same sample, starting from the optimizer's returned estimate when the estimate and objective are finite. The retry sets `ndeps = 1e-5` per parameter on the optimizer's `par / parscale` scale. Supplying `optim.args$control$ndeps` or an optimizer gradient via `optim.args$gr` skips the retry. A failed refit stops the test by default. With `refit.failure = "report"`, failed columns contain `NA`. If any refits fail, p-values are `NA` and the bounds treat every missing statistic first as a nonexceedance and then as an exceedance, using the full requested replicate count. The bounds describe this bootstrap run and are separate from Monte Carlo uncertainty. The multiplier bootstrap uses centered standard normal multipliers and an estimated parameter influence function. Grid-based statistics reuse the grid constructed from the original observations in both procedures.

The observation-based statistics `ksa` and `cvma` are respectively the maximum absolute empirical-minus-fitted CDF difference at observations and n times its mean square. The grid-based counter-

parts ks and cvm evaluate these expressions at empirical or fitted quantile points. The KS quantities use the unscaled differences, consistently in observed and simulated statistics. The empirical CDF in multiple dimensions counts componentwise inequalities.

Univariate models are the primary supported interface. Multivariate models require compatible distribution wrappers with a single vector parameter. The internal normal and t wrappers use integration algorithms for dimensions two and three; they do not provide general higher-dimensional support. Analytical t derivatives assume unrestricted correlations and fixed degrees of freedom. There is no interface for a test with all parameters known.

Set a random seed before calling the function for reproducible simulations within a fixed R and dependency environment. No seed is set internally.

Value

a list with the following components:

`statistic`	Named vector of test statistics: `ks`, `cvm`, `ksa`, `cvma` when `gridStat = TRUE`; otherwise `ksa`, `cvma`.
`p.value`	Named vector of p-values in the same order as `statistic`. Computed as `(count + 0.5) / (nsim + 1)`, where `count` is the number of bootstrap statistics at least as large as the observed statistic. Values lie strictly between zero and one, including when no or all bootstrap statistics exceed the observed statistic.
`estimate`	vector of estimated parameter values
`stat.sim`	Matrix of bootstrap draws with one named row per statistic and `nsim` columns.
`refit.failures`	In parametric report mode, a list containing the failure count, fraction, replicate indices, and error messages.
`p.value.bounds`	In parametric report mode, a matrix of lower and upper bounds with one row per statistic. Both bounds equal the reported p-value when all refits succeed.

Author(s)

Ivan Kojadinovic and Jun Yan.

References

Kojadinovic I. and Yan J. (2012): Goodness-of-fit testing based on a weighted bootstrap: A fast large-sample alternative to the parametric bootstrap. Canadian Journal of Statistics, 40(3), 480–500. doi:10.1002/cjs.11135.

Examples

```
set.seed(123)
# Small simulation counts illustrate usage; use more for final inference.

n <- 200
x <- rlnorm(200, 1, 1)

system.time(m1 <- gof.test(x, "lnorm", 99, list(meanlog=1, sdlog=1), simulation="pb"))
system.time(m2 <- gof.test(x, "lnorm", 99, list(meanlog=1, sdlog=1), simulation="mult"))
```

```
apply(m1$stat.sim, 1, summary)
apply(m2$stat.sim, 1, summary)

system.time(m1 <- gof.test(x, "gamma", 99, list(shape=1, rate=1), simulation="pb"))
system.time(m2 <- gof.test(x, "gamma", 99, list(shape=1, rate=1), simulation="mult"))
apply(m1$stat.sim, 1, summary)
apply(m2$stat.sim, 1, summary)


system.time(m1 <- gof.test(x, "exp", 99, list(rate=1), simulation="pb"))
system.time(m2 <- gof.test(x, "exp", 99, list(rate=1), simulation="mult"))
apply(m1$stat.sim, 1, summary)
apply(m2$stat.sim, 1, summary)
```

Index

∗ **htest**
gof.test, 2

gof.test, 2