# Package 'MaxMC'

October 12, 2022

**Type** Package

**Title** Maximized Monte Carlo

**Date** 2019-03-12

**Version** 0.1.1

**Maintainer** Julien Neves <jmn252@cornell.edu>

**Description** An implementation of the Monte Carlo techniques described in details by Dufour (2006) <doi:10.1016/j.jeconom.2005.06.007> and Dufour and Khalaf (2007) <doi:10.1002/9780470996249.ch24>. The two main features available are the Monte Carlo method with tie-breaker, mc(), for discrete statistics, and the Maximized Monte Carlo, mmc(), for statistics with nuisance parameters.

**License** GPL (>= 3)

**LazyData** TRUE

**RoxygenNote** 6.1.1

**Encoding** UTF-8

**URL** https://github.com/julienneves/MaxMC

**Suggests** fUnitRoots, microbenchmark, boot, MASS, knitr, rmarkdown

**Imports** GenSA, pso, GA, NMOF, scales, stats, graphics, utils

**NeedsCompilation** no

**Author** Julien Neves [aut, cre],
Jean-Marie Dufour [aut]

**Repository** CRAN

**Date/Publication** 2019-03-24 09:06:11 UTC

## R topics documented:

---

`MaxMC-package` *Maximized Monte Carlo*

---

## Description

Functions that implement the Maximized Monte Carlo technique based on Dufour, J.-M. (2006), Monte Carlo Tests with nuisance parameters: A general approach to finite sample inference and nonstandard asymptotics in econometrics. *Journal of Econometrics*, **133(2)**, 443-447.

The main functions of **MaxMC** are `mmc` and `mc`.

## Author(s)

Julien Neves, jmn252@cornell.edu (Maintainer)

Jean-Marie Dufour, jean-marie.dufour@mcgill.ca

## References

Dufour, J.-M. (2006), Monte Carlo Tests with nuisance parameters: A general approach to finite sample inference and nonstandard asymptotics in econometrics. *Journal of Econometrics*, **133(2)**, 443-447.

Dufour, J.-M. and Khalaf L. (2003), Monte Carlo Test Methods in Econometrics. in Badi H. Baltagi, ed., *A Companion to Theoretical Econometrics*, Blackwell Publishing Ltd, 494-519.

---

`mc` *Monte Carlo with Tie-Breaker*

---

## Description

Find the Monte Carlo (MC) p-value by generating N replications of a statistic.

## Usage

```
mc(y, statistic, ..., dgp = function(y) sample(y, replace = TRUE),
  N = 99, type = c("geq", "leq", "absolute", "two-tailed"))
```

## Arguments

| | |
|---|---|
| `y` | A vector or data frame. |
| `statistic` | A function or a character string that specifies how the statistic is computed. The function needs to input the `y` and output a scalar. |
| `...` | Other named arguments for statistic which are passed unchanged each time it is called |

| | |
|---|---|
| dgp | A function. The function inputs the first argument y and outputs a simulated y. It should represent the data generating process under the null. Default value is the function sample(y, replace = TRUE), i.e. the bootstrap resampling of y. |
| N | An atomic vector. Number of replications of the test statistic. |
| type | A character string. It specifies the type of test the p-value function produces. The possible values are geq, leq, absolute and two-tailed. Default is geq. |

## Details

The dgp function defined by the user is used to generate new observations in order to compute the simulated statistics.

Then pvalue is applied to the statistic and its simulated values. pvalue computes the p-value by ranking the statistic compared to its simulated values. Ties in the ranking are broken according to a uniform distribution.

We allow for four types of p-value: leq, geq, absolute and two-tailed. For one-tailed test, leq returns the proportion of simulated values smaller than the statistic while geq returns the proportion of simulated values greater than the statistic. For two-tailed test with a symmetric statistic, one can use the absolute value of the statistic and its simulated values to retrieve a two-tailed test (i.e. type = absolute). If the statistic is not symmetric, one can specify the p-value type as two-tailed which is equivalent to twice the minimum of leq and geq.

Ties in the ranking are broken according to a uniform distribution.

## Value

The returned value is an object of class mc containing the following components:

| | |
|---|---|
| S0 | Observed value of statistic. |
| pval | Monte Carlo p-value of statistic. |
| y | Data specified in call. |
| statistic | statistic function specified in call. |
| dgp | dgp function specified in call. |
| N | Number of replications specified in call. |
| type | type of p-value specified in call. |
| call | Original call to mmc. |
| seed | Value of .Random.seed at the start of mc call. |

## References

Dufour, J.-M. (2006), Monte Carlo Tests with nuisance parameters: A general approach to finite sample inference and nonstandard asymptotics in econometrics. *Journal of Econometrics*, **133(2)**, 443-447.

Dufour, J.-M. and Khalaf L. (2003), Monte Carlo Test Methods in Econometrics. in Badi H. Baltagi, ed., *A Companion to Theoretical Econometrics*, Blackwell Publishing Ltd, 494-519.

## See Also

mmc, pvalue

## Examples

```
## Example 1
## Kolmogorov-Smirnov Test using Monte Carlo

# Set seed
set.seed(999)

# Generate sample data
y <- rgamma(8, shape = 2, rate = 1)

# Set data generating process function
dgp <- function(y)  rgamma(length(y), shape = 2, rate = 1)

# Set the statistic function to the Kolomogorov-Smirnov test for gamma distribution
statistic <- function(y){
    out <- ks.test(y, "pgamma", shape = 2, rate = 1)
    return(out$statistic)
}

# Apply the Monte Carlo test with tie-breaker
mc(y, statistic = statistic, dgp = dgp, N = 999, type = "two-tailed")
```

---

# mmc *Find the Maximized Monte Carlo (MMC) p-value on a set of nuisance parameters.*

---

## Description

The dgp function defined by the user is used to generate new observations in order to compute the simulated statistics.

## Usage

```
mmc(y, statistic, ..., dgp = function(y, v) sample(y, replace = TRUE),
  est = NULL, lower, upper, N = 99, type = c("geq", "leq",
  "absolute", "two-tailed"), method = c("GenSA", "pso", "GA",
  "gridSearch"), control = list(), alpha = NULL, monitor = FALSE)
```

## Arguments

| | |
|---|---|
| y | A vector or data frame. |
| statistic | A function or a character string that specifies how the statistic is computed. The function needs to input the y and output a scalar. |

| | |
|---|---|
| ... | Other named arguments for statistic which are passed unchanged each time it is called |
| dgp | A function. The function inputs the first argument y and a vector of nuisance parameters v and outputs a simulated y. It should represent the data generating process under the null. Default value is the function sample(y, replace = TRUE), i.e. the bootstrap resampling of y. |
| est | A vector with length of v. It is the starting point of the algorithm. If est is a consistent estimate of v then mmc will return both the MMC and Local Monte Carlo (LMC). Default is NULL, in which case, default values will be generated automatically. |
| lower | A vector with length of v. Lower bounds for nuisance parameters under the null. See Details. |
| upper | A vector with length of v. Upper bounds for nuisance parameters under the null. See Details. |
| N | An atomic vector. Number of replications of the test statistic. |
| type | A character string. It specifies the type of test the p-value function produces. The possible values are geq, leq, absolute and two-tailed. Default is geq. |
| method | A character string. Type of algorithm to be used for global optimization. The four available methods are simulated annealing (GenSA), particle swarm (pso), genetic algorithm (GA), and grid search (gridSearch) Default is GenSA, |
| control | A list. Arguments to be used to control the behavior of the algorithm chosen in method. See controls section for more details. |
| alpha | An atomic vector. If mmc finds a p-value over alpha, then the algorithm will stop. This is particularly useful if we are only looking at testing a hypothesis at a particular level. Default is NULL. |
| monitor | A logical. If set to TRUE, then the p-values at every iteration and the cumulative maximum p-value are plotted on a graphical device. Default is FALSE. |

## Details

Then pvalue is applied to the statistic and its simulated values.pvalue computes the p-value by ranking the statistic compared to its simulated values. Ties in the ranking are broken according to a uniform distribution.

We allow for four types of p-value: leq, geq, absolute and two-tailed. For one-tailed test, leq returns the proportion of simulated values smaller than the statistic while geq returns the proportion of simulated values greater than the statistic. For two-tailed test with a symmetric statistic, one can use the absolute value of the statistic and its simulated values to retrieve a two-tailed test (i.e. type = absolute). If the statistic is not symmetric, one can specify the p-value type as two-tailed which is equivalent to twice the minimum of leq and geq.

Ties in the ranking are broken according to a uniform distribution.

Usually, to ensure that the MMC procedure is exact, lower and upper must be set such that any theoretically possible values for the nuisance parameters under the null are covered. This can be computationally expansive.

Alternatively, the consistent set estimate MMC method (CSEMMC) which is applicable when a consistent set estimator of the nuisance parameters is available can be used. If such set is available,

by setting `lower` and `upper` accordingly, `mmc` will yield an asymptotically justified version of the MMC procedure.

One version of this procedure is the Two-stage constrained maximized Monte Carlo test, where first a confidence set of level $1 - \alpha_1$ for the nuisance parameters is obtained and then the MMC with confidence level $\alpha_2$ is taken over this particular set. This procedure yields a conservative test with level $\alpha = \alpha_1 + \alpha_2$. Note that we generally advise against using asymptotic Wald-type confidence intervals based on their poor performance. Instead, it is simply best to build confidence set using problem-specific tools.

## Value

The returned value is an object of class `mmc` containing the following components:

| | |
|---|---|
| `S0` | Observed value of `statistic`. |
| `pval` | Maximized Monte Carlo p-value of `statistic` under null. |
| `y` | Data specified in call. |
| `statistic` | `statistic` function specified in call. |
| `dgp` | `dgp` function specified in call. |
| `est` | `est` vector if specified in call. |
| `lower` | `lower` vector if specified in call. |
| `upper` | `upper` vector if specified in call. |
| `N` | Number of replications specified in call. |
| `type` | `type` of p-value specified in call. |
| `method` | `method` specified in call. |
| `call` | Original call to `mmc`. |
| `seed` | Value of `.Random.seed` at the start of `mmc` call. |
| `lmc` | If `par` is specified, it returns an object of class `mc` corresponding to the Local Monte Carlo test. |
| `opt_result` | An object returning the optimization results. |
| `rejection` | If `alpha` is specified, it returns a vector specifying whether the hypothesis was rejected at level `alpha`. |

## Controls

**Controls - GenSA:**

**maxit** Integer. Maximum number of iterations of the algorithm. Defaults to 1000.

**nb.stop.improvement** Integer. The program will stop when there is no any improvement in nb.stop.improvement steps. Defaults to 25

**smooth** Logical.TRUE when the objective function is smooth, or differentiable almost everywhere in the region of par, FALSE otherwise. Default value is TRUE.

**max.call** Integer. Maximum number of call of the objective function. Default is set to 1e7.

**max.time** Numeric. Maximum running time in seconds.

**temperature** Numeric. Initial value for temperature.

**visiting.param** Numeric. Parameter for visiting distribution.

**acceptance.param** Numeric. Parameter for acceptance distribution.

**simple.function** Logical. FALSE means that the objective function has only a few local minima. Default is FALSE which means that the objective function is complicated with many local minima.

**Controls - psoptim:**

**maxit** The maximum number of iterations. Defaults to 1000.

**maxf** The maximum number of function evaluations (not considering any performed during numerical gradient computation). Defaults to Inf.

**reltol** The tolerance for restarting. Once the maximal distance between the best particle and all other particles is less than reltol*d the algorithm restarts. Defaults to 0 which disables the check for restarting.

**s** The swarm size. Defaults to floor(10+2*sqrt(length(par))) unless type is "SPSO2011" in which case the default is 40.

**k** The exponent for calculating number of informants. Defaults to 3.

**p** The average percentage of informants for each particle. A value of 1 implies that all particles are fully informed. Defaults to 1-(1-1/s)^k.

**w** The exploitation constant. A vector of length 1 or 2. If the length is two, the actual constant used is gradially changed from w[1] to w[2] as the number of iterations or function evaluations approach the limit provided. Defaults to 1/(2*log(2)).

**c.p** The local exploration constant. Defaults to .5+log(2).

**c.g** The global exploration constant. Defaults to .5+log(2).

**d** The diameter of the search space. Defaults to the euclidean distance between upper and lower.

**v.max** The maximal (euclidean) length of the velocity vector. Defaults to NA which disables clamping of the velocity. However, if specified the actual clamping of the length is v.max*d.

**rand.order** Logical; if TRUE the particles are processed in random order. If vectorize is TRUE then the value of rand.order does not matter. Defaults to TRUE.

**max.restart** The maximum number of restarts. Defaults to Inf.

**maxit.stagnate** The maximum number of iterations without improvement. Defaults to 25

**vectorize** Logical; if TRUE the particles are processed in a vectorized manner. This reduces the overhead associated with iterating over each particle and may be more time efficient for cheap function evaluations. Defaults to TRUE.

**type** Character vector which describes which reference implementation of SPSO is followed. Can take the value of "SPSO2007" or "SPSO2011". Defaults to "SPSO2007".

**Controls - GA:**

**popSize** the population size.

**pcrossover** the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.

**pmutation** the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.

**updatePop** a logical defaulting to FALSE. If set at TRUE the first attribute attached to the value returned by the user-defined fitness function is used to update the population. Be careful though, this is an experimental feature!

**postFitness** a user-defined function which, if provided, receives the current ga-class object as input, performs post fitness-evaluation steps, then returns an updated version of the object which is used to update the GA search. Be careful though, this is an experimental feature!

**maxiter** the maximum number of iterations to run before the GA search is halted.

**run** the number of consecutive generations without any improvement in the best fitness value before the GA is stopped.

**optim** a logical defaulting to FALSE determining whether or not a local search using general-purpose optimisation algorithms should be used. See argument optimArgs for further details and finer control.

**optimArgs** a list controlling the local search algorithm with the following components:

- **method** a string specifying the general-purpose optimisation method to be used, by default is set to "L-BFGS-B". Other possible methods are those reported in `optim`.
- **poptim** a value in the range [0,1] specifying the probability of performing a local search at each iteration of GA (default 0.1).
- **pressel** a value in the range [0,1] specifying the pressure selection (default 0.5). The local search is started from a random solution selected with probability proportional to fitness. High values of pressel tend to select the solutions with the largest fitness, whereas low values of pressel assign quasi-uniform probabilities to any solution.
- **control** a list of control parameters. See 'Details' section in `optim`.

**keepBest** a logical argument specifying if best solutions at each iteration should be saved in a slot called bestSol. See ga-class.

**parallel** a logical argument specifying if parallel computing should be used (TRUE) or not (FALSE, default) for evaluating the fitness function. This argument could also be used to specify the number of cores to employ; by default, this is taken from detectCores. Finally, the functionality of parallelization depends on system OS: on Windows only 'snow' type functionality is available, while on Unix/Linux/Mac OSX both 'snow' and 'multicore' (default) functionalities are available.

**Controls - `gridSearch`:**

**n** the number of levels. Default is 10.

**printDetail** print information on the number of objective function evaluations

**method** can be loop (the default), multicore or snow. See Details.

**mc.control** a list containing settings that will be passed to mclapply if method is multicore. Must be a list of named elements; see the documentation of mclapply in parallel.

**cl** default is NULL. If method snow is used, this must be a cluster object or an integer (the number of cores).

**keepNames** logical: should the names of levels be kept?

**asList** does fun expect a list? Default is FALSE

## References

Dufour, J.-M. (2006), Monte Carlo Tests with nuisance parameters: A general approach to finite sample inference and nonstandard asymptotics in econometrics. *Journal of Econometrics*, **133(2)**, 443-447.

Dufour, J.-M. and Khalaf L. (2003), Monte Carlo Test Methods in Econometrics. in Badi H. Baltagi, ed., *A Companion to Theoretical Econometrics*, Blackwell Publishing Ltd, 494-519.

Y. Xiang, S. Gubian. B. Suomela, J. Hoeng (2013). Generalized Simulated Annealing for Efficient Global Optimization: the GenSA Package for R. *The R Journal*, Volume **5/1**, June 2013. URL https://journal.r-project.org/.

Claus Bendtsen. (2012). pso: Particle Swarm Optimization. R package version 1.0.3. https://CRAN.R-project.org/package=pso

Luca Scrucca (2013). GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software*, **53(4)**, 1-37. URL http://www.jstatsoft.org/v53/i04/.

Luca Scrucca (2016). On some extensions to GA package: hybrid optimisation, parallelisation and islands evolution. Submitted to *R Journal*. Pre-print available at arXiv URL http://arxiv.org/abs/1605.01931.

Manfred Gilli (2011), Dietmar Maringer and Enrico Schumann. Numerical Methods and Optimization in Finance. *Academic Press*.

## See Also

mc, pvalue

## Examples

```
## Example 1
## Exact Unit Root Test
library(fUnitRoots)

# Set seed
set.seed(123)

# Generate an AR(2) process with phi = (-1.5,0.5), and n = 25
y <- filter(rnorm(25), c(-1.5, 0.5), method = "recursive")

# Set bounds for the nuisance parameter v
lower <- -1
upper <- 1

# Set the function to generate an AR(2) integrated process
dgp <- function(y, v) {
    ran.y <- filter(rnorm(length(y)), c(1-v,v), method = "recursive")
}

# Set the Augmented-Dicky Fuller statistic
statistic <- function(y){
    out <- suppressWarnings(adfTest(y, lags = 2, type = "nc"))
    return(out@test$statistic)
}

# Apply the mmc procedure
mmc(y, statistic = statistic , dgp = dgp, lower = lower,
    upper = upper, N = 99, type = "leq", method = "GenSA",
    control = list(max.time = 2))
```

```
## Example 2
## Behrens-Fisher Problem
library(MASS)

# Set seed
set.seed(123)

# Generate sample x1 ~ N(0,1) and x2 ~ N(0,4)
x1 <- rnorm(15, mean = 0, sd = 1)
x2 <- rnorm(25, mean = 0, sd = 2)
data <- list(x1 = x1, x2 = x2)

# Fit a normal distribution on x1 and x2 using maximum likelihood
fit1 <- fitdistr(x1, "normal")
fit2 <- fitdistr(x2, "normal")

# Extract the estimate for the nuisance parameters v = (sd_1, sd_2)
est <- c(fit1$estimate["sd"], fit2$estimate["sd"])

# Set the bounds of the nuisance parameters equal to the 99% CI
lower <- est - 2.577 * c(fit2$sd["sd"], fit1$sd["sd"])
upper <- est + 2.577 * c(fit2$sd["sd"], fit1$sd["sd"])

# Set the function for the DGP under the null (i.e. two population means are equal)
dgp <- function(data, v) {
    x1 <- rnorm(length(data$x1), mean = 0, sd = v[1])
    x2 <- rnorm(length(data$x2), mean = 0, sd = v[2])
    return(list(x1 = x1, x2 = x2))
}

# Set the statistic function to Welch's t-test
welch <- function(data) {
    test <- t.test(data$x2, data$x1)
    return(test$statistic)
}

# Apply Welch's t-test
t.test(data$x2, data$x1)

# Apply the mmc procedure
mmc(y = data, statistic = welch, dgp = dgp, est = est,
    lower = lower, upper = upper, N = 99,type = "absolute",
    method = "pso")
```

---

`plot.mmc` *Plot a* `mmc` *Object*

---

## Description

The `plot()` method for objects of the class `mmc` gives a plot of the best and current p-value found during the iterations of `mmc`.

## Usage

```
## S3 method for class 'mmc'
plot(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class `mmc`. |
| `...` | Arguments to be passed to methods, such as graphical parameters (see `par`). |

## Value

The `mmc` object is returned invisibly.

## Examples

```
## Example
library(fUnitRoots)
# Set seed
set.seed(123)

# Generate an AR(2) process with phi = (-1.5,0.5), and n = 25
y <- filter(rnorm(25), c(-1.5, 0.5), method = "recursive")

# Set bounds for the nuisance parameter v
lower <- -1
upper <- 1

# Set the function to generate an AR(2) integrated process
dgp <- function(y, v) {
    ran.y <- filter(rnorm(length(y)), c(1-v,v), method = "recursive")
}

# Set the Augmented-Dicky Fuller statistic
statistic <- function(y){
    out <- suppressWarnings(adfTest(y, lags = 2, type = "nc"))
    return(out@test$statistic)
}

# Apply the mmc procedure
est <- mmc(y, statistic = statistic , dgp = dgp, lower = lower,
           upper = upper, N = 99, type = "leq", method = "GenSA",
           control = list(max.time = 2))

# Plot result of object of class 'mmc'
plot(est)
```

print.mc *Print a Summary of a* mc *Object*

## Description

This is a method for the function print() for objects of the class mc.

## Usage

```
## S3 method for class 'mc'
print(x, digits = getOption("digits"), ...)
```

## Arguments

| | |
|---|---|
| x | an object used to select a method. |
| digits | minimal number of *significant* digits, see print.default. |
| ... | further arguments passed to or from other methods. |

## Value

The mc object is returned invisibly.

## Examples

```
## Example
# Set seed
set.seed(999)

# Generate sample data
y <- rgamma(8, shape = 2, rate = 1)

# Set data generating process function
dgp <- function(y)  rgamma(length(y), shape = 2, rate = 1)

# Set the statistic function to the Kolomogorov-Smirnov test for gamma distribution
statistic <- function(y){
    out <- ks.test(y, "pgamma", shape = 2, rate = 1)
    return(out$statistic)
}

# Apply the Monte Carlo test with tie-breaker
est <- mc(y, statistic = statistic, dgp = dgp, N = 999, type = "two-tailed")

# Print result of object of class 'mc'
print(est)
```

print.mmc *Print a Summary of a* mmc *Object*

## Description

This is a method for the function `print()` for objects of the class `mmc`.

## Usage

```
## S3 method for class 'mmc'
print(x, digits = getOption("digits"), ...)
```

## Arguments

| | |
|---|---|
| `x` | an object used to select a method. |
| `digits` | minimal number of *significant* digits, see `print.default`. |
| `...` | further arguments passed to or from other methods. |

## Value

The `mmc` object is returned invisibly.

## Examples

```
## Example
library(fUnitRoots)
# Set seed
set.seed(123)

# Generate an AR(2) process with phi = (-1.5,0.5), and n = 25
y <- filter(rnorm(25), c(-1.5, 0.5), method = "recursive")

# Set bounds for the nuisance parameter v
lower <- -1
upper <- 1

# Set the function to generate an AR(2) integrated process
dgp <- function(y, v) {
    ran.y <- filter(rnorm(length(y)), c(1-v,v), method = "recursive")
}

# Set the Augmented-Dicky Fuller statistic
statistic <- function(y){
    out <- suppressWarnings(adfTest(y, lags = 2, type = "nc"))
    return(out@test$statistic)
}

# Apply the mmc procedure
est <- mmc(y, statistic = statistic , dgp = dgp, lower = lower,
```

```
            upper = upper, N = 99, type = "leq", method = "GenSA",
            control = list(max.time = 2))

# Print result of object of class 'mmc'
print(est)
```

---

pvalue *p-value Function*

---

## Description

Computes the p-value of the statistic by computing its rank compared to its simulated values.

## Usage

```
pvalue(S0, S, type = c("geq", "leq", "absolute", "two-tailed"))
```

## Arguments

| | |
|---|---|
| S0 | An atomic vector. Value of the test statistic applied to the data. |
| S | A vector. It consists of replications of the test statistic. S must have length greater than one. |
| type | A character string. It specifies the type of test the p-value function produces. The possible values are geq, leq, absolute and two-tailed. Default is geq. |

## Details

We allow for four types of p-value: leq, geq, absolute and two-tailed. For one-tailed test, leq returns the proportion of simulated values smaller than the statistic while geq returns the proportion of simulated values greater than the statistic. For two-tailed test with a symmetric satistic, one can use the absolute value of the statistic and its simulated values to retrieve a two-tailed test (i.e. type = absolute). If the statistic is not symmetric, one can specify the p-value type as two-tailed which is equivalent to twice the minimum of leq and geq.

Ties in the ranking are broken according to a uniform distribution.

## Value

The p-value of the statistic S0 given a vector of replications S.

## References

Dufour, J.-M. (2006), Monte Carlo Tests with nuisance parameters: A general approach to finite sample inference and nonstandard asymptotics in econometrics. *Journal of Econometrics*, **133(2)**, 443-447.

Dufour, J.-M. and Khalaf L. (2003), Monte Carlo Test Methods in Econometrics. in Badi H. Baltagi, ed., *A Companion to Theoretical Econometrics*, Blackwell Publishing Ltd, 494-519.

## Examples

```
# Generate sample S0 and simulate statistics
S0 = 0
S = rnorm(99)

# Compute p-value
pvalue(S0, S, type = "geq")
```

# Index