

Package ‘MCHT’

October 5, 2018

Title Bootstrap and Monte Carlo Hypothesis Testing

Version 0.1.0

Date 2018-10-05

Description Facilitates bootstrap and Monte Carlo hypothesis testing.

Depends R (>= 3.2)

Imports testthat, doParallel, doRNG, memoise, foreach, GenSA, purrr,
stats, utils, Rdpack

RdMacros Rdpack

License MIT + file LICENSE

LazyData true

Encoding UTF-8

RoxygenNote 6.1.0

Suggests MASS, fitdistrplus

NeedsCompilation no

Author Curtis Miller [aut, cre]

Maintainer Curtis Miller <cmiller@math.utah.edu>

R topics documented:

.onAttach	2
check_params_in_functions	2
gen_memo_rng	3
get_MCHTest_settings	3
is.MCHTest	4
MCHTest	4
MCHT_startup_message	8
print.MCHTest	8
pval	9
%s%	10
%s0%	10
Index	11

`.onAttach`*Package Attach Hook Function*

Description

Hook triggered when package attached

Usage

```
.onAttach(lib, pkg)
```

Arguments

<code>lib</code>	a character string giving the library directory where the package defining the namespace was found
<code>pkg</code>	a character string giving the name of the package

Examples

```
MCHT:::onAttach(.libPaths()[1], "MCHT")
```

`check_params_in_functions`*Check That Parameters Are In Functions*

Description

Test that certain parameters are arguments for certain functions via **testthat** functions.

Usage

```
check_params_in_functions(params, func_list)
```

Arguments

<code>params</code>	Character vector with parameter names
<code>func_list</code>	List of functions to check

Examples

```
MCHT:::check_params_in_functions(c("x"), list(mean))
```

gen_memo_rng

*Memoised Random Variable Generation***Description**

Creates a function that generates random numbers with memoization

Usage

```
gen_memo_rng(r, seed = NULL)
```

Arguments

r	The random number generator
seed	The seed value, to be passed to set.seed

Details

This is a function generator, with the returned function being one that can handle a set seed and will remember if it needs to regenerate a new set of random numbers. This allows both for control over random number generation and for faster performance.

Value

A function that generates random numbers with a set seed and with memoization; accepts seed and all other arguments that could be passed to the original random number generator

Examples

```
memo_runif <- MCHT:::gen_memo_rng(runif)
memo_runif(10)
```

get_MCHTest_settings

*Get Attributes of MCHTest Object***Description**

Get the settings of an [MCHTest](#)-class object.

Usage

```
get_MCHTest_settings(x)
```

Arguments

x	The MCHTest-class object
---	--------------------------

Value

A list with all the variables relevant to x

Examples

```
f <- MCHTest(mean, mean, seed = 100)
get_MCHTest_settings(f)
```

is.MCHTest

Is an Object of Type MCHTest?

Description

Checks whether its argument is an `MCHTest`-class object.

Usage

```
is.MCHTest(x)
```

Arguments

x An R object

Value

TRUE if x is an `MCHTest`-class object, FALSE otherwise

Examples

```
f <- MCHTest(mean, mean, seed = 100)
is.MCHTest(1)
is.MCHTest(f)
```

MCHTest

Create an MCHTest Object

Description

This function creates an `MCHTest`-class object, an S3 object that defines a bootstrap or Monte Carlo test.

Usage

```
MCHTest(test_stat, stat_gen, rand_gen = stats::runif, N = 10000,
  seed = NULL, memoise_sample = TRUE, pval_func = MCHT::pval,
  method = "Monte Carlo Test", test_params = NULL,
  fixed_params = NULL, nuisance_params = NULL, optim_control = NULL,
  tiebreaking = FALSE, lock_alternative = TRUE, threshold_pval = 1,
  suppress_threshold_warning = FALSE)
```

Arguments

test_stat	A function that computes the test statistic from input data; <code>x</code> must be a parameter of this function representing test data
stat_gen	A function that generates values of the test statistic when given data; <code>x</code> (representing a sample) must be a parameter of this function, and this function is expected to return one numeric output, but if <code>n</code> is a parameter, this will be interpreted as sample size information (this could be useful for allowing a "burn-in" period in random data, as is often the case when working with time series data)
rand_gen	A function generating random data, accepting a parameter <code>n</code> (representing the size of the data) or <code>x</code> (which would be the actual data)
N	Integer representing the number of replications of <code>stat_gen</code> to generate
seed	The random seed used to generate simulated statistic values; if <code>NULL</code> , the seed will be randomly chosen each time the resulting function is called (unless <code>memoise_sample</code> is <code>TRUE</code>)
memoise_sample	If <code>TRUE</code> , simulated statistic values are saved and will be used repeatedly if the inputs to <code>stat_gen</code> don't change (such as the sample size, <code>n</code>); this could be in conflict with <code>seed</code> if <code>seed</code> is <code>NULL</code> , so set to <code>FALSE</code> to allow for regeneration of random samples for every call to the resulting function
pval_func	A function that computes <i>p</i> -values from the test statistic computed by <code>test_stat</code> using the simulated data generated via <code>stat_gen</code> ; see pval for an example of how this function should be specified
method	A string labelling the test
test_params	A character vector of the names of parameters with values specified under the null hypothesis; both <code>test_stat</code> and <code>stat_gen</code> need to be able to recognize the contents of this vector as parameters (for example, if this argument is <code>"mu"</code> , then <code>mu</code> needs to be an argument of both <code>test_stat</code> and <code>stat_gen</code>), and the resulting test will try to pass these parameters to <code>rand_gen</code> (but these <i>do not</i> need to be parameters of <code>rand_gen</code>)
fixed_params	A character vector of the names of parameters treated as fixed values; this isn't needed but if these parameters are being used then test output is more informative and errors will be raised if <code>test_stat</code> and <code>stat_gen</code> don't accept these parameters—which is safer—and the resulting test will try to pass these parameters to <code>rand_gen</code> (but these <i>do not</i> need to be parameters of <code>rand_gen</code>)
nuisance_params	A character vector of the names of parameters to be treated as nuisance parameters which must be chosen via optimization (see (Dufour 2006)); must be parameters of <code>test_stat</code> and <code>stat_gen</code> , but these <i>will not</i> be viewed as parameters of <code>rand_gen</code> , and cannot be non- <code>NULL</code> if <code>codeoptim_control</code> is <code>NULL</code>
optim_control	A list of arguments to be passed to GenSA , containing at least lower and upper elements as named vectors, with the names being identical to <code>nuisance_params</code> , but could also include other arguments to be passed to GenSA ; the <code>fn</code> parameter will be set, and parameters of that function will be the parameters mentioned in <code>nuisance_params</code> , and this argument will be ignored if <code>nuisance_params</code> is <code>NULL</code>
tiebreaking	Break ties using the method as described in Dufour (2006); won't work if <code>pval_func</code> doesn't support it via a <code>unif_gen</code> argument, and should only be used for test statistics not computed on continuously-distributed data

lock_alternative

If TRUE, then the resulting function will effectively ignore the alternative parameter, while if FALSE, the resulting function will be sensitive to values of alternative; this argument exists to prevent shooting yourself in the foot and accidentally computing p -values in inappropriate ways

threshold_pval A numeric value that represents a threshold p -value that, if surpassed by the optimization algorithm, will cause the algorithm to terminate; will override the `threshold.stop` argument in the control list that's used by [GenSA](#)

suppress_threshold_warning

If TRUE, user will not be warned if the threshold p -value was surpassed by the optimization algorithm

Value

A MCHTest-class object, a function with parameters `x`, `alternative`, and ..., with other parameters being passed to functions such as those passed to `test_stat` and `stat_gen`, controlling what's tested and how; depending on `lock_alternative`, the `alternative` argument may be ignored

Examples

```
dat <- c(0.16, 1.00, 0.67, 1.28, 0.31, 1.16, 1.25, 0.93, 0.66, 0.54)
# Monte Carlo t-test for exponentially distributed data
mc.t.test <- MCHTest(test_stat = function(x, mu = 1) {
  sqrt(length(x)) * (mean(x) - mu)/sd(x)
}, stat_gen = function(x, mu = 1) {
  x <- x * mu
  sqrt(length(x)) * (mean(x) - mu)/sd(x)
}, rand_gen = rexp, seed = 123,
method = "Monte Carlo t-Test", test_params = "mu",
lock_alternative = FALSE)

mc.t.test(dat)
mc.t.test(dat, mu = 0.1, alternative = "two.sided")

# Testing for the scale parameter of a Weibull distribution
# Two-sided test for location of scale parameter
library(MASS)
library(fitdistrplus)

ts <- function(x, scale = 1) {
  fit_null <- coef(fitdist(x, "weibull", fix.arg = list("scale" = scale)))
  kt <- fit_null[["shape"]]
  l0 <- scale
  fit_all <- coef(fitdist(x, "weibull"))
  kh <- fit_all[["shape"]]
  lh <- fit_all[["scale"]]
  n <- length(x)

  # Test statistic, based on the negative-log-likelihood ratio
  suppressWarnings(n * ((kt - 1) * log(l0) - (kh - 1) * log(lh) -
    log(kt/kh) - log(lh/l0)) - (kt - kh) * sum(log(x)) + l0^(-kt) *
    sum(x^kt) - lh^(-kh) * sum(x^kh))
}

sg <- function(x, scale = 1, shape = 1) {
  x <- qweibull(x, shape = shape, scale = scale)
  ts(x, scale = scale)
```

```

}

mc.wei.shape.test <- MCHTest(ts, sg, seed = 123, test_params = "scale",
                             nuisance_params = "shape",
                             optim_control = list(
                               lower = c("shape" = 0),
                               upper = c("shape" = 100),
                               control = list("max.time" = 10)
                             ), threshold_pval = .2, N = 1000)

mc.wei.shape.test(rweibull(100, scale = 4, shape = 2), scale = 2)

# Bootstrap hypothesis test
# Kolmogorov-Smirnov test for Weibull distribution via parametric bootstrap
# hypothesis test

ts <- function(x) {
  param <- coef(fitdist(x, "weibull"))
  shape <- param[['shape']]; scale <- param[['scale']]
  ks.test(x, pweibull, shape = shape, scale = scale,
          alternative = "two.sided")$statistic[[1]]
}

rg <- function(x) {
  n <- length(x)
  param <- coef(fitdist(x, "weibull"))
  shape <- param[['shape']]; scale <- param[['scale']]
  rweibull(n, shape = shape, scale = scale)
}

b.ks.test <- MCHTest(test_stat = ts, stat_gen = ts, rand_gen = rg,
                     seed = 123, N = 1000)
b.ks.test(rbeta(100, 2, 2))

# Permutation test

df <- data.frame(
  val = c(rnorm(5, mean = 2, sd = 3), rnorm(10, mean = 1, sd = 2)),
  group = rep(c("x", "y"), times = c(5, 10))
)

ts <- function(x) {
  means <- aggregate(val ~ group, data = x, mean)
  vars <- aggregate(val ~ group, data = x, var)
  counts <- aggregate(val ~ group, data = x, length)

  (means$val[1] - means$val[2])/sum(vars$val / sqrt(counts$val))
}

rg <- function(x) {
  x$group <- sample(x$group)
  x
}

permute.test <- MCHTest(ts, ts, rg, seed = 123, N = 1000,
                       lock_alternative = FALSE)

```

```
permute.test(df, alternative = "two.sided")
```

```
MCHT_startup_message
```

Create Package Startup Message

Description

Makes package startup message.

Usage

```
MCHT_startup_message()
```

Examples

```
MCHT:::MCHT_startup_message()
```

```
print.MCHTest
```

Print MCHTest-Class Object

Description

Print an link{MCHTest}-class object.

Usage

```
## S3 method for class 'MCHTest'
print(x, ...)
```

Arguments

x	The MCHTest-class object
...	Other arguments, such as prefix (a string wrapped around the first line; by default, "\t")

Examples

```
f <- MCHTest(mean, mean, seed = 100)
print(f)
```


pval

*Compute p-Value For a Test Statistic***Description**

Compute the p -value of a test statistic for Monte Carlo tests.

Usage

```
pval(S, sample_S, alternative = NULL, unif_gen = NULL)
```

Arguments

<code>S</code>	The value of the test statistic
<code>sample_S</code>	Simulated values of the
<code>alternative</code>	A string specifying the alternative hypothesis, or NULL
<code>unif_gen</code>	If not NULL, the function generating uniformly-distributed random variables for breaking ties; if NULL, no tie breaking is done

Details

Let S be a test statistic and S_i be simulated values of that test statistic under the null hypothesis, with $1 \leq i \leq N$. If `unif_gen` is not NULL, this function computes p -values via

$$p = \hat{p} = \frac{1}{N} \sum_{i=1}^N I_{\{(S, U_0) \leq (S_i, U_i)\}}$$

where $I_{\{S \in A\}} = 1$ if $S \in A$ and is 0 otherwise, U_i are uniformly distributed random variables, and the ordering over tuples is lexicographical ordering, as described by Dufour (2006).

If `unif_gen` is NULL, then the random variables are not generated and not used to break ties.

This function is designed to handle an alternative parameter similar to what appears in other **stats** functions like `t.test`. If `alternative` is "less", then $p = \hat{p}$; if `alternative` is "greater", then $p = 1 - \hat{p}$; and if `alternative` is "two.sided", then $p = 2 \min(\hat{p}, 1 - \hat{p})$. Any other value raises an error.

The parameter `S` is S , and the vector `sample_S` is the vector containing the values S_i .

Value

A number representing the p -value.

Examples

```
sample_S <- rnorm(10)
pval(1.01, sample_S)
pval(1.01, sample_S, alternative = "greater")
```

%s%	Concatenate (With Space)
-----	--------------------------

Description

Concatenate and form strings (with space separation)

Usage

x %s% y

Arguments

- x One object
- y Another object

Value

A string combining x and y with a space separating them

Examples

```
`%s%` <- MCHT:::`%s%`  
"Hello" %s% "world"
```

%s0%	Concatenate (Without Space)
------	-----------------------------

Description

Concatenate and form strings (no space separation)

Usage

x %s0% y

Arguments

- x One object
- y Another object

Value

A string combining x and y

Examples

```
`%s0%` <- MCHT:::`%s0%`  
"Hello" %s0% "world"
```

Index

`.onAttach`, [2](#)
`%s0%`, [10](#)
`%s%`, [10](#)

`check_params_in_functions`, [2](#)

`gen_memo_rng`, [3](#)
`GenSA`, [5](#), [6](#)
`get_MCHTest_settings`, [3](#)

`is.MCHTest`, [4](#)

`MCHT_startup_message`, [8](#)
`MCHTest`, [3](#), [4](#), [4](#)

`print.MCHTest`, [8](#)
`pval`, [5](#), [9](#)

`set.seed`, [3](#)

`t.test`, [9](#)