# Package 'MCHT'

## October 5, 2018

Title Bootstrap and Monte Carlo Hypothesis Testing
Version 0.1.0
<b>Date</b> 2018-10-05
<b>Description</b> Facilitates bootstrap and Monte Carlo hypothesis testing.
<b>Depends</b> R (>= $3.2$ )
<b>Imports</b> 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></cmiller@math.utah.edu>
R topics documented:
.onAttach
check_params_in_functions
get_MCHTest_settings
is.MCHTest
MCHTest
MCHT_startup_message
print.MCHTest
pval
%s0%
7,550,70
Index 1

.onAttach

Package Attach Hook Function

#### **Description**

Hook triggered when package attached

## Usage

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

#### **Arguments**

1ib a character string giving the library directory where the package defining the

namespace was found

pkg 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

params Character vector with parameter names

func\_list List of functions to check

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

gen\_memo\_rng 3

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)</pre>
```

```
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)</pre>
```

is.MCHTest

Is an Object of Type MCHTest?

#### **Description**

Checks whether its argument is an MCHTest-class object.

## Usage

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

#### **Arguments**

Х

An R object

## Value

TRUE if x is an MCHTest-class objet, FALSE otherwise

## **Examples**

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

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; x must be a parameter

of this function representing test data

stat\_gen A function that generates values of the test statistic when given data; x (rep-

resenting a sample) must be a parameter of this function, and this function is expected to return one numeric output, but if n 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 n (representing the

size of the data) or x (which would be the actual data)

N Integer representing the number of replications of stat\_gen to generate

seed The random seed used to generate simulated statistic values; if NULL, the seed

 $will be \ randomly \ chosen \ each \ time \ the \ resulting \ function \ is \ called \ (unless \ memoise\_sample$ 

is TRUE)

memoise\_sample If TRUE, simulated statistic values are saved and will be used repeatedly if the

inputs to  $stat\_gen\ don't\ change\ (such\ as\ the\ sample\ size,\ n);$  this could be in conflict with seed if seed is NULL, so set to FALSE to allow for regeneration of

random samples for every call to the resulting function

pval\_func A function that computes p-values from the test statistic computed by test\_stat

using the simulated data generated via stat\_gen; 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 test\_stat and stat\_gen need to be able to recognize the contents of this vector as parameters (for example, if this argument is "mu", then mu needs to be an argument of both test\_stat and stat\_gen), and the resulting test will try to pass these parameters to rand\_gen (but these *do not* need to be

parameters of rand\_gen)

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 test\_stat and stat\_gen don't accept these parameters—which is safer—and the resulting test will try to pass these param-

eters to rand\_gen (but these *do not* need to be parameters of rand\_gen)

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 test\_stat and stat\_gen, but these *will not* be viewed as param-

eters of rand\_gen, and cannot be non-NULL if codeoptim\_control is NULL

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 nuisance\_params, but could also include other arguments to be passed to GenSA; the fn parameter will be set, and parameters of that function will be the parameters mentioned in nuisance\_params, and this argument will be ignored if nuisance\_params is

NULL

tiebreaking Break ties using the method as described in Dufour (2006); won't work if pval\_func

doesn't support it via a unif\_gen 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

#### **Details**

MCHTest-class objects are effectively functions that accept data and maybe some parameters and return an htest-class object containing the results of a Monte Carlo or bootstrap statistical test. These object will accept datasets and perhaps some parameters and will return the results of a test.

Bootstrap tests can be implemented when the dataset is passed as an argument to  $rand\_gen$  (which occurs when x is one of  $rand\_gen$ 's parameters). The only difference between a Monte Carlo test and a bootstrap test in the context of this function is that bootstrap tests use information from the original dataset when generating simulated test statistics, while a Monte Carlo test does not. When the default function for computing p-values is used, this function will perform a test similar to that described by .

For Monte Carlo tests, when the default function for computing p-values is used (see pval), this is effectively the test described in Dufour (2006). This includes using simulated annealing to find values of nuisance parameters that maximize the p-value if the null hypothesis is true. Simulated annealing is implemented using GenSA from the GenSA package, and the optim\_control parameter is used for controlling GenSA's behavior. We highly recommend reading GenSA's documentation.

The threshold\_pval argument can be used for stopping the optimization procedure when a specified p-value is reached or surpassed. Dufour (2006) showed that p-values found using the procedure implemented here are conservative (in the sense that they are larger than they necessarily need to be). If the algorithm terminates early due to surpassing a prespecified p-value, then the estimated p-value is known to at least be the value returned, but because the p-value is a conservative estimate of the "true" p-value, this latter number could be smaller. Thus we cannot say much about the location of the true p-value if the algorithm terminates early. For this reason, a MCHTest-class function will, by default, issue a warning if the algorithm terminated early. However, by setting suppress\_threshold\_warning to TRUE, this behavior can be disabled. This recognizes the fact that even though an early termination leads to us not being able to say much about the location of the true p-value, we know that whatever the more accurate estimate is, we would not reject the null hypothesis based on that result.

This function uses foreach, %dorng%, and %dopar% to perform simulations. If the R session is not set up at the start for parallelization, there will be an initial complaint (after which there are no more complaints), then these functions will default to using a single core. The example shows how to set up R to use all available cores.

#### 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

```
dat \leftarrow 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) {</pre>
                        sqrt(length(x)) * (mean(x) - mu)/sd(x)
                      }, stat_gen = function(x, mu = 1) {
                        x \leftarrow 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) {</pre>
  fit_null <- coef(fitdist(x, "weibull", fix.arg = list("scale" = scale)))</pre>
  kt <- fit_null[["shape"]]</pre>
  10 <- scale
  fit_all <- coef(fitdist(x, "weibull"))</pre>
  kh <- fit_all[["shape"]]</pre>
  lh <- fit_all[["scale"]]</pre>
  n \leftarrow 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) - 1h^(-kh) * sum(x^kh)
}
sg <- function(x, scale = 1, shape = 1) {</pre>
 x \leftarrow qweibull(x, shape = shape, scale = scale)
  ts(x, scale = scale)
}
mc.wei.shape.test <- MCHTest(ts, sg, seed = 123, test_params = "scale",</pre>
                               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 botstrap
# hypothesis test
ts <- function(x) {</pre>
  param <- coef(fitdist(x, "weibull"))</pre>
  shape <- param[['shape']]; scale <- param[['scale']]</pre>
```

```
ks.test(x, pweibull, shape = shape, scale = scale,
           alternative = "two.sided")$statistic[[1]]
rg <- function(x) {</pre>
  n \leftarrow length(x)
  param <- coef(fitdist(x, "weibull"))</pre>
  shape <- param[['shape']]; scale <- param[['scale']]</pre>
  rweibull(n, shape = shape, scale = scale)
}
b.ks.test <- MCHTest(test_stat = ts, stat_gen = ts, rand_gen = rg,
                       seed = 123, N = 1000)
b.ks.test(rbeta(100, 2, 2))
# Permutation test
df <- data.frame(</pre>
  val = c(rnorm(5, mean = 2, sd = 3), rnorm(10, mean = 1, sd = 2)),
  group = rep(c("x", "y"), times = c(5, 10))
ts <- function(x) {</pre>
  means <- aggregate(val ~ group, data = x, mean)</pre>
  vars <- aggregate(val ~ group, data = x, var)</pre>
  counts <- aggregate(val ~ group, data = x, length)</pre>
  (means$val[1] - means$val[2])/sum(vars$val / sqrt(counts$val))
}
rg <- function(x) {</pre>
 x$group <- sample(x$group)</pre>
permute.test <- MCHTest(ts, ts, rg, seed = 123, N = 1000,</pre>
                          lock_alternative = FALSE)
permute.test(df, alternative = "two.sided")
```

## **Description**

Makes package startup message.

## Usage

```
MCHT_startup_message()
```

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

print.MCHTest 9

## 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)</pre>
```

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

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

10 %s%

#### **Details**

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

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

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")</pre>
```

%s%

Concatenate (With Space)

#### **Description**

Concatenate and form strings (with space separation)

## Usage

```
x %s% y
```

#### **Arguments**

x One objecty Another object

#### Value

A string combining x and y with a space separating them

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

%s0%

%s0%

Concatenate (Without Space)

## Description

Concatenate and form strings (no space separation)

## Usage

```
x %s0% y
```

## **Arguments**

```
x One objecty Another object
```

## Value

A string combining x and y

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

# **Index**

```
.onAttach, 2
%dopar\%, 6
%dorng%, 6
%s0%, 11
%s%, 10
\verb|check_params_in_functions|, 2|
foreach, 6
gen_memo_rng, 3
GenSA, 5, 6
{\tt get\_MCHTest\_settings}, 3
\verb"is.MCHTest", 4
{\tt MCHT\_startup\_message}, 8
MCHTest, 3, 4, 4
print.MCHTest,9
pval, 5, 6, 9
set.seed, 3
t.test, 10
```