Package 'DiscreteTests'

February 16, 2026

Type Package

Title Vectorised Computation of P-Values and Their Supports for Several Discrete Statistical Tests

Version 0.3.0

Date 2026-02-13

Description Provides vectorised functions for computing p-values of various common discrete statistical tests, as described e.g. in Agresti (2002) <doi:10.1002/0471249688>, including their distributions. Exact and approximate computation methods are provided. For exact ones, several procedures of determining two-sided p-values are included, which are outlined in more detail in Hirji (2006) <doi:10.1201/9781420036190>.

License GPL-3

Encoding UTF-8

Language en-GB

Imports Rcpp, R6, checkmate, lifecycle, cli, tibble, withr

LinkingTo Rcpp

URL https://github.com/DISOhda/DiscreteTests

BugReports https://github.com/DISOhda/DiscreteTests/issues

RoxygenNote 7.3.3

NeedsCompilation yes

Author Florian Junge [cre, aut] (ORCID: <https://orcid.org/0009-0001-6856-6938>), Christina Kihn [aut], Sebastian Döhler [ctb] (ORCID: <https://orcid.org/0000-0002-0321-6355>), Guillermo Durand [ctb] (ORCID: <https://orcid.org/0000-0003-4056-5631>)

Maintainer Florian Junge <diso.fbmn@h-da.de>

Repository CRAN

Date/Publication 2026-02-16 08:00:09 UTC

Contents

`DiscreteTests-package` *Vectorised Computation of P-Values and Their Supports for Several Discrete Statistical Tests*

Description

This package provides vectorised functions for computing p-values of various discrete statistical tests. Exact and approximate computation methods are provided. For exact p-values, several procedures of determining two-sided p-values are included.

Additionally, these functions are capable of returning the discrete p-value supports, i.e. all observable p-values under a null hypothesis. These supports can be used for multiple testing procedures in the `DiscreteFDR` and `FDX` packages.

Author(s)

Maintainer: Florian Junge <diso.fbmn@h-da.de> (ORCID)

Authors:

- Christina Kihn

Other contributors:

- Sebastian Döhler (ORCID) [contributor]
- Guillermo Durand (ORCID) [contributor]

References

Fisher, R. A. (1935). The logic of inductive inference. *Journal of the Royal Statistical Society Series A*, **98**, pp. 39–54. doi:10.2307/2342435

Agresti, A. (2002). *Categorical data analysis* (2nd ed.). New York: John Wiley & Sons. doi:10.1002/0471249688

Blaker, H. (2000) Confidence curves and improved exact confidence intervals for discrete distributions. *Canadian Journal of Statistics*, **28**(4), pp. 783-798. doi:10.2307/3315916

Hirji, K. F. (2006). *Exact analysis of discrete data*. New York: Chapman and Hall/CRC. pp. 55-83. doi:10.1201/9781420036190

See Also

Useful links:

- https://github.com/DISOhda/DiscreteTests
- Report bugs at https://github.com/DISOhda/DiscreteTests/issues

---

`binom_test_pv` *Binomial Tests*

Description

`binom_test_pv()` performs an exact or approximate binomial test about the probability of success in a Bernoulli experiment. In contrast to `stats::binom.test()`, it is vectorised, only calculates *p*-values and offers a normal approximation of their computation. Furthermore, it is capable of returning the discrete *p*-value supports, i.e. all observable *p*-values under a null hypothesis. Multiple tests can be evaluated simultaneously. In two-sided tests, several procedures of obtaining the respective *p*-values are implemented.

Note: Please do not use the older `binom.test.pv()` anymore! It is now defunct and will be removed in a future version.
[Superseded]

Usage

```
binom_test_pv(
  x,
  n,
  p = 0.5,
  alternative = "two.sided",
  ts_method = "minlike",
  exact = TRUE,
  correct = TRUE,
  simple_output = FALSE
)

binom.test.pv(
  x,
  n,
  p = 0.5,
  alternative = "two.sided",
  ts.method = "minlike",
  exact = TRUE,
```

```
  correct = TRUE,
  simple.output = FALSE
)
```

Arguments

x	integer vector giving the number of successes.
n	integer vector giving the number of trials.
p	numerical vector of hypothesised probabilities of success.
alternative	character vector that indicates the alternative hypotheses; each value must be one of "two.sided" (the default), "less" or "greater".
ts_method, ts.method	single character string that indicates the two-sided p-value computation method (if any value in alternative equals "two.sided") and must be one of "minlike" (the default), "blaker", "absdist" or "central" (see details). Ignored, if exact = FALSE.
exact	logical value that indicates whether p-values are to be calculated by exact computation (TRUE; the default) or by a continuous approximation (FALSE).
correct	logical value that indicates if a continuity correction is to be applied (TRUE; the default) or not (FALSE). Ignored, if exact = TRUE.
simple_output, simple.output	logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

The parameters x, n, p and alternative are vectorised. They are replicated automatically to have the same lengths. This allows multiple hypotheses to be tested simultaneously.

If p = NULL, it is tested if the probability of success is 0.5 with the alternative being specified by alternative.

For exact computation, various procedures of determining two-sided p-values are implemented.

"minlike" The standard approach in stats::fisher.test() and stats::binom.test(). The probabilities of the likelihoods that are equal or less than the observed one are summed up. In Hirji (2006), it is referred to as the *Probability-based* approach.

"blaker" The minima of the observations' lower and upper tail probabilities are combined with the opposite tail not greater than these minima. More details can be found in Blaker (2000) or Hirji (2006), where it is referred to as the *Combined Tails* method.

"absdist" The probabilities of the absolute distances from the expected value that are greater than or equal to the observed one are summed up. In Hirji (2006), it is referred to as the *Distance from Center* approach.

"central" The smaller values of the observations' simply doubles the minimum of lower and upper tail probabilities. In Hirji (2006), it is referred to as the *Twice the Smaller Tail* method.

For non-exact (i.e. continuous approximation) approaches, ts_method is ignored, since all its methods would yield the same p-values. More specifically, they all converge to the doubling approach as in ts_mthod = "central".

Value

If `simple.output = TRUE`, a vector of computed p-values is returned. Otherwise, the output is a `DiscreteTestResults` R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. `$get_pvalues()`.

References

Agresti, A. (2002). *Categorical data analysis*. Second Edition. New York: John Wiley & Sons. pp. 14-15. doi:10.1002/0471249688

Blaker, H. (2000). Confidence curves and improved exact confidence intervals for discrete distributions. *Canadian Journal of Statistics*, **28**(4), pp. 783-798. doi:10.2307/3315916

Hirji, K. F. (2006). *Exact analysis of discrete data*. New York: Chapman and Hall/CRC. pp. 55-83. doi:10.1201/9781420036190

See Also

`stats::binom.test()`

Examples

```
# Constructing
k <- c(4, 2, 2, 14, 6, 9, 4, 0, 1)
n <- c(18, 12, 10)
p <- c(0.5, 0.2, 0.3)

# Exact two-sided p-values ("blaker") and their supports
results_ex  <- binom_test_pv(k, n, p, ts_method = "blaker")
print(results_ex)
results_ex$get_pvalues()
results_ex$get_pvalue_supports()

# Normal-approximated one-sided p-values ("less") and their supports
results_ap  <- binom_test_pv(k, n, p, "less", exact = FALSE)
print(results_ap)
results_ap$get_pvalues()
results_ap$get_pvalue_supports()
```

DiscreteTestResults *Discrete Test Results Class*

Description

This is the class used by the statistical test functions of this package for returning not only p-values, but also the supports of their distributions and the parameters of the respective tests. Objects of this class are obtained by setting the `simple.output` parameter of a test function to `FALSE` (the default). All data members of this class are private to avoid inconsistencies by deliberate or inadvertent changes by the user. However, the results can be read by public methods.

Methods

Public methods:

- DiscreteTestResults$new()
- DiscreteTestResults$get_pvalues()
- DiscreteTestResults$get_inputs()
- DiscreteTestResults$get_statistics()
- DiscreteTestResults$get_pvalue_supports()
- DiscreteTestResults$get_support_indices()
- DiscreteTestResults$print()
- DiscreteTestResults$clone()

Method `new()`**:** Creates a new `DiscreteTestResults` object.

Usage:

```
DiscreteTestResults$new(
  test_name,
  inputs,
  statistics,
  p_values,
  pvalue_supports,
  support_indices,
  data_name
)
```

Arguments:

`test_name` single character string with the name of the test(s).

`inputs` named list of **exactly four named** elements containing the observations, test parameters and hypothesised null values **as data frames or lists**; the names of these list fields must be `observations`, `parameters`, `nullvalues` and `computation`. See details for further information about the requirements for these fields.

`statistics` data frame containing the tests' statistics; `NULL` is allowed and recommended, e.g. if the observed values themselves are the statistics.

`p_values` numeric vector of the p-values calculated by each hypothesis test.

`pvalue_supports` list of **unique** numeric vectors containing all p-values that are observable under the respective hypothesis; each value of `p_values` must occur in its respective p-value support.

`support_indices` list of numeric vectors containing the test indices that indicates to which individual testing scenario each unique parameter set and each unique support belongs.

`data_name` single character string with the name of the variable that contains the observed data.

Details: The fields of the `inputs` have the following requirements:

`$observations` data frame or list of vectors that comprises of the observed data; if it is a matrix, it must be converted to a data frame; must not be `NULL`, only numerical and character values are allowed.

`$nullvalues` data frame that holds the hypothesised values of the tests, e.g. the rate parameters for Poisson tests; must not be `NULL`, only numerical values are allowed.

`$parameters` data frame that may contain additional parameters of each test (e.g. numbers of Bernoulli trials for binomial tests). Only numerical, character or logical values are permitted; `NULL` is allowed, too, e.g. if there are no additional parameters.

`$computation` data frame that consists of details about the p-value computation, e.g. if they were calculated exactly, the used distribution etc. It **must** include mandatory columns named `exact`, `alternative` and `distribution`. Any additional information may be added, like the marginals for Fisher's exact test etc., but only numerical, character or logical values are allowed.

All data frames must have the same number of rows. Their column names are used by the `print()` method for producing text output, therefore they should be informative, i.e. short and (if necessary) non-syntactic, like e.g. `` `number of success` ``.
The mandatory column `exact` of the data frame `computation` must be logical, while the values of `alternative` must be one of `"greater"`, `"less"`, `"two.sided"`, `"minlike"`, `"blaker"`, `"absdist"` or `"central"`. The `distribution` column must hold character strings that identify the distribution under the null hypothesis, e.g. `"normal"`. All the columns of this data frame are used by the `print()` method, so their names should also be informative and (if necessary) non-syntactic.

Method `get_pvalues()`**:** Returns the computed p-values.

Usage:

```
DiscreteTestResults$get_pvalues(named = TRUE)
```

Arguments:

`named` single logical value that indicates whether the vector is to be returned as a named vector (if names are present)

Returns: A numeric vector of the p-values of all null hypotheses.

Method `get_inputs()`**:** Return the list of the test inputs.

Usage:

```
DiscreteTestResults$get_inputs(unique = FALSE)
```

Arguments:

`unique` single logical value that indicates whether only unique combinations of parameter sets and null values are to be returned. If `unique = FALSE` (the default), the returned data frames may contain duplicate sets.

Returns: A list of four elements. The first one contains a data frame with the observations for each tested null hypothesis, while the second is another data frame with additional parameters (if any, e.g. `n` in case of a binomial test) that were passed to the respective test's function. The third list field holds the hypothesised null values (e.g. `p` for binomial tests). The last list element contains computational details, e.g. test `alternatives`, the used `distribution` etc. If `unique = TRUE`, only unique combinations of parameters, null values and computation specifics are returned, but observations remain unchanged (i.e. they are never unique).

Method `get_statistics()`**:** Returns the test statistics.

Usage:

```
DiscreteTestResults$get_statistics()
```

Returns: A numeric `data.frame` with one column containing the test statistics.

Method `get_pvalue_supports()`**:** Returns the p-value supports, i.e. all observable p-values under the respective null hypothesis of each test.

Usage:

```
DiscreteTestResults$get_pvalue_supports(unique = FALSE)
```

Arguments:

`unique` single logical value that indicates whether only unique p-value supports are to be returned. If `unique = FALSE` (the default), the returned supports may be duplicated.

Returns: A list of numeric vectors containing the supports of the p-value null distributions.

Method `get_support_indices()`**:** Returns the indices that indicate to which tested null hypothesis each unique support belongs.

Usage:

```
DiscreteTestResults$get_support_indices()
```

Returns: A list of numeric vectors. Each one contains the indices of the null hypotheses to which the respective support and/or unique parameter set belongs.

Method `print()`**:** Prints the computed p-values.

Usage:

```
DiscreteTestResults$print(
  inputs = TRUE,
  pvalue_details = TRUE,
  supports = FALSE,
  test_idx = NULL,
  limit = 10,
  ...
)
```

Arguments:

`inputs` single logical value that indicates if the input values (i.e. observations, statistics and parameters) are to be printed; defaults to `TRUE`.

`pvalue_details` single logical value that indicates if details about the p-value computation are to be printed; defaults to `TRUE`.

`supports` single logical value that indicates if the p-value supports are to be printed; defaults to `FALSE`.

`test_idx` integer vector giving the indices of the tests whose results are to be printed; if `NULL` (the default), results of every test up to the index specified by `limit` (see below) are printed.

`limit` single integer that indicates the maximum number of test results to be printed; if `limit = 0`, results of every test are printed; ignored if `test_idx` is not set to `NULL`

`...` further arguments passed to `print.default()`.

Returns: Prints a summary of the tested null hypotheses. The object itself is invisibly returned.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DiscreteTestResults$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## one-sided binomial test
#  parameters
x <- 2:4
n <- 5
p <- 0.4
m <- length(x)
#  support (same for all three tests) and p-values
support <- sapply(0:n, function(k) binom.test(k, n, p, "greater")$p.value)
pv <- support[x + 1]
#  DiscreteTestResults object
res <- DiscreteTestResults$new(
  # string with name of the test
  test_name = "Exact binomial test",
  # list of data frames
  inputs = list(
    observations = data.frame(
      `number of successes` = x,
      # no name check of column header to have a speaking name for 'print'
      check.names = FALSE
    ),
    parameters = data.frame(
      # parameter 'n', needs to be replicated to length of 'x'
      `number of trials` = rep(n, m),
      # no name check of column header to have a speaking name for 'print'
      check.names = FALSE
    ),
    nullvalues = data.frame(
      # here: only one null value, 'p'; needs to be replicated to length of 'x'
      `probability of success` = rep(p, m),
      # no name check of column header to have a speaking name for 'print'
      check.names = FALSE
    ),
    computation = data.frame(
      # mandatory parameter 'alternative', needs to be replicated to the length of 'x'
      alternative = rep("greater", m),
      # mandatory exactness information, replicated to the length of 'alternative'
      exact = rep(TRUE, m),
      # mandatory distribution information, replicated to the length of 'alternative'
      distribution = rep("binomial", m)
    )
  ),
  # test statistics (not needed, since observation itself is the statistic)
  statistics = NULL,
  # numerical vector of p-values
  p_values = pv,
  # list of supports (here: only one support); values must be sorted and unique
  pvalue_supports = list(unique(sort(support))),
  # list of indices that indicate which p-value/hypothesis each support belongs to
  support_indices = list(1:m),
  # name of input data variables
  data_name = "x, n and p"
```

```
)

#  print results without supports
print(res)
#  print results with supports
print(res, supports = TRUE)
```

`DiscreteTestResultsSummary`
Discrete Test Results Summary Class

Description

This is the class used by `DiscreteTests` for summarising `DiscreteTestResults` objects. It contains the summarised objects itself, as well as a summary `tibble` object as private members. Both can be extracted by public methods.

Methods

Public methods:

- `DiscreteTestResultsSummary$new()`
- `DiscreteTestResultsSummary$get_test_results()`
- `DiscreteTestResultsSummary$get_summary_table()`
- `DiscreteTestResultsSummary$print()`
- `DiscreteTestResultsSummary$clone()`

Method `new()`**:** Creates a new `summary.DiscreteTestResults` object.

Usage:

```
DiscreteTestResultsSummary$new(test_results)
```

Arguments:

`test_results` the `DiscreteTestResults` class object to be summarised.

Method `get_test_results()`**:** Returns the underlying DiscreteTestResults object.

Usage:

```
DiscreteTestResultsSummary$get_test_results()
```

Returns: A DiscreteTestResults R6 class object.

Method `get_summary_table()`**:** Returns the summary table of the underlying DiscreteTestResults object.

Usage:

```
DiscreteTestResultsSummary$get_summary_table()
```

Returns: A data frame.

Method `print()`**:** Prints the summary.

Usage:

```
DiscreteTestResultsSummary$print(...)
```

Arguments:

`...` further arguments passed to `print.data.frame`.

Returns: Prints a summary table of the tested null hypotheses. The object itself is invisibly returned.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
DiscreteTestResultsSummary$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# binomial tests
obj <- binom_test_pv(0:5, 5, 0.5)
# create DiscreteTestResultsSummary object
res <- DiscreteTestResultsSummary$new(obj)
# print summary
print(res)
# extract summary table
res$get_summary_table()
```

`fisher_test_pv` *Fisher's Exact Test for Count Data*

Description

`fisher_test_pv()` performs Fisher's exact test or a chi-square approximation to assess if rows and columns of a 2-by-2 contingency table with fixed marginals are independent. In contrast to `stats::fisher.test()`, it is vectorised, only calculates p-values and offers a normal approximation of their computation. Furthermore, it is capable of returning the discrete p-value supports, i.e. all observable p-values under a null hypothesis. Multiple tables can be analysed simultaneously. In two-sided tests, several procedures of obtaining the respective p-values are implemented.

Note: Please do not use the older `fisher.test.pv()` anymore! It is now defunct and will be removed in a future version.
[Superseded]

Usage

```
fisher_test_pv(
  x,
  alternative = "two.sided",
  ts_method = "minlike",
  exact = TRUE,
  correct = TRUE,
  simple_output = FALSE
)

fisher.test.pv(
  x,
  alternative = "two.sided",
  ts.method = "minlike",
  exact = TRUE,
  correct = TRUE,
  simple.output = FALSE
)
```

Arguments

x	integer vector with four elements, a 2-by-2 matrix or an integer matrix (or data frame) with four columns, where each line represents a 2-by-2 table to be tested.
alternative	character vector that indicates the alternative hypotheses; each value must be one of "two.sided" (the default), "less" or "greater".
ts_method, ts.method	single character string that indicates the two-sided p-value computation method (if any value in alternative equals "two.sided") and must be one of "minlike" (the default), "blaker", "absdist" or "central" (see details). Ignored, if exact = FALSE.
exact	logical value that indicates whether p-values are to be calculated by exact computation (TRUE; the default) or by a continuous approximation (FALSE).
correct	logical value that indicates if a continuity correction is to be applied (TRUE; the default) or not (FALSE). Ignored, if exact = TRUE.
simple_output, simple.output	logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

The parameters x and alternative are vectorised. They are replicated automatically, such that the number of x's rows is the same as the length of alternative. This allows multiple null hypotheses to be tested simultaneously. Since x is (if necessary) coerced to a matrix with four columns, it is replicated row-wise.

If exact = TRUE, Fisher's exact test is performed (the specific hypothesis depends on the value of alternative). Otherwise, if exact = FALSE, a chi-square approximation is used for two-sided

hypotheses or a normal approximation for one-sided tests, based on the square root of the chi-squared statistic. This is possible because the degrees of freedom of chi-squared tests on 2-by-2 tables are limited to 1.

For exact computation, various procedures of determining two-sided p-values are implemented.

`"minlike"` The standard approach in `stats::fisher.test()` and `stats::binom.test()`. The probabilities of the likelihoods that are equal or less than the observed one are summed up. In Hirji (2006), it is referred to as the *Probability-based* approach.

`"blaker"` The minima of the observations' lower and upper tail probabilities are combined with the opposite tail not greater than these minima. More details can be found in Blaker (2000) or Hirji (2006), where it is referred to as the *Combined Tails* method.

`"absdist"` The probabilities of the absolute distances from the expected value that are greater than or equal to the observed one are summed up. In Hirji (2006), it is referred to as the *Distance from Center* approach.

`"central"` The smaller values of the observations' simply doubles the minimum of lower and upper tail probabilities. In Hirji (2006), it is referred to as the *Twice the Smaller Tail* method.

For non-exact (i.e. continuous approximation) approaches, `ts_method` is ignored, since all its methods would yield the same p-values. More specifically, they all converge to the doubling approach as in `ts_mthod = "central"`.

Value

If `simple.output = TRUE`, a vector of computed p-values is returned. Otherwise, the output is a `DiscreteTestResults` R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. `$get_pvalues()`.

References

Fisher, R. A. (1935). The logic of inductive inference. *Journal of the Royal Statistical Society Series A*, **98**, pp. 39–54. doi:10.2307/2342435

Agresti, A. (2002). *Categorical data analysis*. Second Edition. New York: John Wiley & Sons. pp. 91–97. doi:10.1002/0471249688

Blaker, H. (2000). Confidence curves and improved exact confidence intervals for discrete distributions. *Canadian Journal of Statistics*, **28**(4), pp. 783-798. doi:10.2307/3315916

Hirji, K. F. (2006). *Exact analysis of discrete data*. New York: Chapman and Hall/CRC. pp. 55-83. doi:10.1201/9781420036190

See Also

`stats::fisher.test()`

Examples

```
# Constructing
S1 <- c(4, 2, 2, 14, 6, 9, 4, 0, 1)
S2 <- c(0, 0, 1, 3, 2, 1, 2, 2, 2)
N1 <- rep(148, 9)
```

```
N2 <- rep(132, 9)
F1 <- N1 - S1
F2 <- N2 - S2
df <- data.frame(S1, F1, S2, F2)

# Fisher's exact p-values and their supports
results_ex <- fisher_test_pv(df)
print(results_ex)
results_ex$get_pvalues()
results_ex$get_pvalue_supports()

# Chi-square-approximated p-values and their supports
results_ap <- fisher_test_pv(df, exact = FALSE)
print(results_ap)
results_ap$get_pvalues()
results_ap$get_pvalue_supports()
```

---

homogeneity_test_pv *Conditional Two-Sample Homogeneity Test for Binomial Experiments*

Description

Performs an exact or approximate conditional test about the homogeneity of two binomial samples, i.e. regarding the respective probabilities of success. It is vectorised, only calculates *p*-values and offers a normal approximation of their computation. Furthermore, it is capable of returning the discrete *p*-value supports, i.e. all observable *p*-values under a null hypothesis. Multiple tests can be evaluated simultaneously. In two-sided tests, several procedures of obtaining the respective *p*-values are implemented.

Usage

```
homogeneity_test_pv(
  x,
  n,
  alternative = "two.sided",
  ts_method = "minlike",
  exact = TRUE,
  correct = TRUE,
  simple_output = FALSE
)
```

Arguments

x	integer vector with two elements or a matrix with two columns or a data frame with two columns giving the number of successes for the two experiments.
n	integer vector with two elements or a matrix with two columns or a data frame with two columns giving the number of trials for the two experiments.

alternative	character vector that indicates the alternative hypotheses; each value must be one of "two.sided" (the default), "less" or "greater".
ts_method	single character string that indicates the two-sided p-value computation method (if any value in alternative equals "two.sided") and must be one of "minlike" (the default), "blaker", "absdist" or "central" (see details). Ignored, if exact = FALSE.
exact	logical value that indicates whether p-values are to be calculated by exact computation (TRUE; the default) or by a continuous approximation (FALSE).
correct	logical value that indicates if a continuity correction is to be applied (TRUE; the default) or not (FALSE). Ignored, if exact = TRUE.
simple_output	logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

The parameters x, n and alternative are vectorised. They are replicated automatically, such that the number of x's rows is the same as the length of alternative. This allows multiple null hypotheses to be tested simultaneously. Since x and n are coerced to matrices (if necessary) with two columns, they are replicated row-wise.

It can be shown that this test is a special case of Fisher's exact test, because it is conditional on the numbers of trials **and** the sums of successes and failures. Therefore, its computations are handled by fisher_test_pv().

For exact computation, various procedures of determining two-sided p-values are implemented.

"minlike" The standard approach in stats::fisher.test() and stats::binom.test(). The probabilities of the likelihoods that are equal or less than the observed one are summed up. In Hirji (2006), it is referred to as the *Probability-based* approach.

"blaker" The minima of the observations' lower and upper tail probabilities are combined with the opposite tail not greater than these minima. More details can be found in Blaker (2000) or Hirji (2006), where it is referred to as the *Combined Tails* method.

"absdist" The probabilities of the absolute distances from the expected value that are greater than or equal to the observed one are summed up. In Hirji (2006), it is referred to as the *Distance from Center* approach.

"central" The smaller values of the observations' simply doubles the minimum of lower and upper tail probabilities. In Hirji (2006), it is referred to as the *Twice the Smaller Tail* method.

For non-exact (i.e. continuous approximation) approaches, ts_method is ignored, since all its methods would yield the same p-values. More specifically, they all converge to the doubling approach as in ts_mthod = "central".

Value

If simple.output = TRUE, a vector of computed p-values is returned. Otherwise, the output is a DiscreteTestResults R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. $get_pvalues().

References

Fisher, R. A. (1935). The logic of inductive inference. *Journal of the Royal Statistical Society Series A*, **98**, pp. 39–54. doi:10.2307/2342435

Agresti, A. (2002). *Categorical data analysis*. Second Edition. New York: John Wiley & Sons. pp. 91–97. doi:10.1002/0471249688

Blaker, H. (2000) Confidence curves and improved exact confidence intervals for discrete distributions. *Canadian Journal of Statistics*, **28**(4), pp. 783-798. doi:10.2307/3315916

Hirji, K. F. (2006). *Exact analysis of discrete data*. New York: Chapman and Hall/CRC. pp. 55-83. doi:10.1201/9781420036190

See Also

`fisher_test_pv()`

Examples

```
# Constructing
set.seed(3)
p1 <- c(0.25, 0.5, 0.75)
p2 <- c(0.15, 0.5, 0.60)
n1 <- c(10, 20, 50)
n2 <- c(25, 75, 200)
x1 <- rbinom(3, n1, p1)
x2 <- rbinom(3, n2, p2)
x  <- cbind(x1 = x1, x2 = x2)
n  <- cbind(n1 = n1, n2 = n2)

# Exact two-sided p-values ("blaker") and their supports
results_ex <- homogeneity_test_pv(x, n, ts_method = "blaker")
print(results_ex)
results_ex$get_pvalues()
results_ex$get_pvalue_supports()

# Normal-approximated one-sided p-values ("less") and their supports
results_ap <- homogeneity_test_pv(x, n, "less", exact = FALSE)
print(results_ap)
results_ap$get_pvalues()
results_ap$get_pvalue_supports()
```

`mann_whitney_test_pv` *Wilcoxon-Mann-Whitney* U *test*

Description

`mann_whitney_test_pv()` performs an exact or approximate Wilcoxon-Mann-Whitney *U* test about the location shift between two independent groups when the data is not necessarily normally distributed. In contrast to `stats::wilcox.test()`, it is vectorised and only calculates *p*-values. Furthermore, it is capable of returning the discrete *p*-value supports, i.e. all observable *p*-values under a null hypothesis. Multiple tests can be evaluated simultaneously.

Usage

```
mann_whitney_test_pv(
  x,
  y,
  mu = 0,
  alternative = "two.sided",
  exact = NULL,
  correct = TRUE,
  digits_rank = Inf,
  simple_output = FALSE
)
```

Arguments

`x`, `y`	numerical vectors forming the samples to be tested or lists of numerical vectors for multiple tests.
`mu`	numerical vector of hypothesised location shift(s).
`alternative`	character vector that indicates the alternative hypotheses; each value must be one of `"two.sided"` (the default), `"less"` or `"greater"`.
`exact`	logical value that indicates whether p-values are to be calculated by exact computation (`TRUE`; the default) or by a continuous approximation (`FALSE`).
`correct`	logical value that indicates if a continuity correction is to be applied (`TRUE`; the default) or not (`FALSE`). Ignored, if `exact = TRUE`.
`digits_rank`	single number giving the significant digits used to compute ranks for the test statistics.
`simple_output`	logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

We use a test statistic called the Wilcoxon Rank Sum Statistic, defined by

$$U = \sum_{i=1}^{n_X} rank(X_i) - \frac{n_X(n_X+1)}{2},$$

where $rank(X_i)$ is the rank of X_i in the concatenated sample of X and Y, and n_X and n_Y are the respective sizes of the samples X and Y. Note that U can range from 0 to $n_X \cdot n_Y$. This is the same statistic used by `stats::wilcox.test()` and whose distribution is accessible with `pwilcox`.

This is also the statistic defined by the two given references. Note, however, that it is not what is called the Mann-Whitney U Statistic in the (English-language) Wikipedia article (as of February 12, 2026). The latter is defined as, using our notation, $\min(U, n_X \cdot n_Y - U)$. Using the Wikipedia notation, the Wilcoxon Rank Sum Statistic is U_2.

The parameters `x`, `y`, `mu` and `alternative` are vectorised. If `x` and `y` are lists, they are replicated automatically to have the same lengths. In case `x` or `y` are not lists, they are added to new ones, which are then replicated to the appropriate lengths. This allows multiple hypotheses to be tested simultaneously.

In the presence of ties, computation of exact *p*-values is not possible. Therefore, `exact` is ignored in these cases and *p*-values of the respective test settings are calculated by a normal approximation.

By setting `exact = NULL`, exact computation is performed if both samples in a test setting do not have any ties and if both sample sizes are lower than or equal to 200.

If `digits_rank = Inf` (the default), `rank()` is used to compute ranks for the tests statistics instead of `rank(signif(., digits_rank))`

Value

If `simple.output = TRUE`, a vector of computed p-values is returned. Otherwise, the output is a `DiscreteTestResults` R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. `$get_pvalues()`.

References

Mann, H. D. & Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Statist.*, *18*(1), pp. 50-60. doi:10.1214/aoms/1177730491

Hollander, M. & Wolfe, D. (1973). *Nonparametric Statistical Methods*. Third Edition. New York: Wiley. pp. 115-135. doi:10.1002/9781119196037

See Also

`stats::wilcox.test()`, `pwilcox`, `wilcox_test_pv()`

Examples

```
# Constructing
set.seed(1)
r1 <- rnorm(100)
r2 <- rnorm(100, 1)

# Exact two-sided p-values and their supports
results_ex  <- mann_whitney_test_pv(r1, r2)
print(results_ex)
results_ex$get_pvalues()
results_ex$get_pvalue_supports()

# Normal-approximated one-sided p-values ("less") and their supports
results_ap  <- mann_whitney_test_pv(r1, r2, alternative = "less", exact = FALSE)
print(results_ap)
```

```
results_ap$get_pvalues()
results_ap$get_pvalue_supports()
```

`mcnemar_test_pv` *McNemar's Test for Count Data*

Description

Performs McNemar's chi-square test or an exact variant to assess the symmetry of rows and columns in a 2-by-2 contingency table. In contrast to `stats::mcnemar.test()`, it is vectorised, only calculates *p*-values and offers their exact computation. Furthermore, it is capable of returning the discrete *p*-value supports, i.e. all observable *p*-values under a null hypothesis. Multiple tables can be analysed simultaneously. In two-sided tests, several procedures of obtaining the respective *p*-values are implemented. It is a special case of the binomial test.

Note: Please do not use the older `mcnemar.test.pv()` anymore! It is now defunct and will be removed in a future version.
[Superseded]

Usage

```
mcnemar_test_pv(
  x,
  alternative = "two.sided",
  exact = TRUE,
  correct = TRUE,
  simple_output = FALSE
)

mcnemar.test.pv(
  x,
  alternative = "two.sided",
  exact = TRUE,
  correct = TRUE,
  simple.output = FALSE
)
```

Arguments

`x`	integer vector with four elements, a 2-by-2 matrix or an integer matrix (or data frame) with four columns where each line represents a 2-by-2 table to be tested.
`alternative`	character vector that indicates the alternative hypotheses; each value must be one of `"two.sided"` (the default), `"less"` or `"greater"`.
`exact`	logical value that indicates whether p-values are to be calculated by exact computation (`TRUE`; the default) or by a continuous approximation (`FALSE`).

`correct` logical value that indicates if a continuity correction is to be applied (`TRUE`; the default) or not (`FALSE`). Ignored, if `exact = TRUE`.

`simple_output, simple.output` logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

The parameters `x` and `alternative` are vectorised. They are replicated automatically, such that the number of `x`'s rows is the same as the length of `alternative`. This allows multiple null hypotheses to be tested simultaneously. Since `x` is coerced to a matrix (if necessary) with four columns, it is replicated row-wise.

It can be shown that McNemar's test is a special case of the binomial test. Therefore, its computations are handled by `binom_test_pv()`. In contrast to that function, `mcnemar_test_pv()` does not allow specifying exact two-sided p-value calculation procedures. The reason is that McNemar's exact test always tests for a probability of 0.5, in which case all these exact two-sided p-value computation methods yield exactly the same results.

Value

If `simple.output = TRUE`, a vector of computed p-values is returned. Otherwise, the output is a `DiscreteTestResults` R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. `$get_pvalues()`.

References

Agresti, A. (2002). *Categorical data analysis*. Second Edition. New York: John Wiley & Sons. pp. 411–413. doi:10.1002/0471249688

See Also

`stats::mcnemar.test()`, `binom_test_pv()`

Examples

```
# Constructing
S1 <- c(4, 2, 2, 14, 6, 9, 4, 0, 1)
S2 <- c(0, 0, 1, 3, 2, 1, 2, 2, 2)
N1 <- rep(148, 9)
N2 <- rep(132, 9)
F1 <- N1 - S1
F2 <- N2 - S2
df <- data.frame(S1, F1, S2, F2)

# Exact p-values and their supports
results_ex  <- mcnemar_test_pv(df)
print(results_ex)
results_ex$get_pvalues()
results_ex$get_pvalue_supports()
```

```
# Chi-square-approximated p-values and their supports
results_ap  <- mcnemar_test_pv(df, exact = FALSE)
print(results_ap)
results_ap$get_pvalues()
results_ap$get_pvalue_supports()
```

---

poisson_test_pv *Poisson Test*

Description

poisson_test_pv() performs an exact or approximate Poisson test about the rate parameter of a Poisson distribution. In contrast to stats::poisson.test(), it is vectorised, only calculates *p*-values and offers a normal approximation of their computation. Furthermore, it is capable of returning the discrete *p*-value supports, i.e. all observable *p*-values under a null hypothesis. Multiple tests can be evaluated simultaneously. In two-sided tests, several procedures of obtaining the respective *p*-values are implemented.

Note: Please do not use the older poisson.test.pv() anymore! It is now defunct and will be removed in a future version.
[Superseded]

Usage

```
poisson_test_pv(
  x,
  lambda = 1,
  alternative = "two.sided",
  ts_method = "minlike",
  exact = TRUE,
  correct = TRUE,
  simple_output = FALSE
)

poisson.test.pv(
  x,
  lambda = 1,
  alternative = "two.sided",
  ts.method = "minlike",
  exact = TRUE,
  correct = TRUE,
  simple.output = FALSE
)
```

Arguments

x	integer vector giving the number of events.
lambda	non-negative numerical vector of hypothesised rate(s).
alternative	character vector that indicates the alternative hypotheses; each value must be one of "two.sided" (the default), "less" or "greater".
ts_method, ts.method	single character string that indicates the two-sided p-value computation method (if any value in alternative equals "two.sided") and must be one of "minlike" (the default), "blaker", "absdist" or "central" (see details). Ignored, if exact = FALSE.
exact	logical value that indicates whether p-values are to be calculated by exact computation (TRUE; the default) or by a continuous approximation (FALSE).
correct	logical value that indicates if a continuity correction is to be applied (TRUE; the default) or not (FALSE). Ignored, if exact = TRUE.
simple_output, simple.output	logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

The parameters x, lambda and alternative are vectorised. They are replicated automatically to have the same lengths. This allows multiple null hypotheses to be tested simultaneously.

Since the Poisson distribution itself has an infinite support, so do the *p*-values of exact Poisson tests. Therefore, supports only contain *p*-values that are not rounded off to 0.

For exact computation, various procedures of determining two-sided p-values are implemented.

"minlike" The standard approach in stats::fisher.test() and stats::binom.test(). The probabilities of the likelihoods that are equal or less than the observed one are summed up. In Hirji (2006), it is referred to as the *Probability-based* approach.

"blaker" The minima of the observations' lower and upper tail probabilities are combined with the opposite tail not greater than these minima. More details can be found in Blaker (2000) or Hirji (2006), where it is referred to as the *Combined Tails* method.

"absdist" The probabilities of the absolute distances from the expected value that are greater than or equal to the observed one are summed up. In Hirji (2006), it is referred to as the *Distance from Center* approach.

"central" The smaller values of the observations' simply doubles the minimum of lower and upper tail probabilities. In Hirji (2006), it is referred to as the *Twice the Smaller Tail* method.

For non-exact (i.e. continuous approximation) approaches, ts_method is ignored, since all its methods would yield the same p-values. More specifically, they all converge to the doubling approach as in ts_mthod = "central".

Value

If simple.output = TRUE, a vector of computed p-values is returned. Otherwise, the output is a DiscreteTestResults R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. $get_pvalues().

References

Blaker, H. (2000). Confidence curves and improved exact confidence intervals for discrete distributions. *Canadian Journal of Statistics*, **28**(4), pp. 783-798. doi:10.2307/3315916

Hirji, K. F. (2006). *Exact analysis of discrete data*. New York: Chapman and Hall/CRC. pp. 55-83. doi:10.1201/9781420036190

See Also

`stats::poisson.test()`, `binom_test_pv()`

Examples

```
# Constructing
k <- c(4, 2, 2, 14, 6, 9, 4, 0, 1)
lambda <- c(3, 2, 1)

# Exact two-sided p-values ("blaker") and their supports
results_ex  <- poisson_test_pv(k, lambda, ts_method = "blaker")
print(results_ex)
results_ex$get_pvalues()
results_ex$get_pvalue_supports()

# Normal-approximated one-sided p-values ("less") and their supports
results_ap  <- poisson_test_pv(k, lambda, "less", exact = FALSE)
print(results_ap)
results_ap$get_pvalues()
results_ap$get_pvalue_supports()
```

`summary.DiscreteTestResults`

Summarizing Discrete Test Results

Description

`summary` method for class `DiscreteTestResults`.

Usage

```
## S3 method for class 'DiscreteTestResults'
summary(object, ...)
```

Arguments

`object`	object of class `DiscreteTestResults` to be summarised; usually created by using one of the packages test functions, e.g. `binom_test_pv()`, with `simple_output = FALSE`.
`...`	further arguments passed to or from other methods.

Value

A summary.DiscreteTestResults R6 class object.

Examples

```
# binomial tests
obj <- binom_test_pv(0:5, 5, 0.4)
# print summary
summary(obj)
# extract summary table
smry <- summary(obj)
smry$get_summary_table()
```

---

wilcox_test_pv *Wilcoxon's signed-rank test*

Description

wilcox_test_pv() performs an exact or approximate Wilcoxon signed-rank test about the location of a population on a single sample or the differences between two paired groups when the data is not necessarily normally distributed. In contrast to stats::wilcox.test(), it is vectorised and only calculates *p*-values. Furthermore, it is capable of returning the discrete *p*-value supports, i.e. all observable *p*-values under a null hypothesis. Multiple tests can be evaluated simultaneously.

Usage

```
wilcox_test_pv(
  x,
  y = NULL,
  mu = 0,
  alternative = "two.sided",
  exact = NULL,
  correct = TRUE,
  digits_rank = Inf,
  simple_output = FALSE
)
```

Arguments

x	numerical vector forming the sample to be tested or a list of numerical vectors for multiple tests.
y	numerical vector forming the second sample to be tested or a list of numerical vectors for multiple tests; if y = NULL (the default), the one-sample version is performed; for two-sample tests, all sample pairs must have the same length.
mu	numerical vector of hypothesised location(s) for one-sample tests or location shift(s) for two-sample tests.

`alternative`	character vector that indicates the alternative hypotheses; each value must be one of `"two.sided"` (the default), `"less"` or `"greater"`.
`exact`	logical value that indicates whether p-values are to be calculated by exact computation (`TRUE`; the default) or by a continuous approximation (`FALSE`).
`correct`	logical value that indicates if a continuity correction is to be applied (`TRUE`; the default) or not (`FALSE`). Ignored, if `exact = TRUE`.
`digits_rank`	single number giving the significant digits used to compute ranks for the test statistics.
`simple_output`	logical value that indicates whether an R6 class object, including the tests' parameters and support sets, i.e. all observable p-values under each null hypothesis, is to be returned (see below).

Details

The parameters `x`, `mu` and `alternative` are vectorised. If `x` is a list, they are replicated automatically to have the same lengths. In case `x` is not a list, it is added to one, which is then replicated to the appropriate length. This allows multiple hypotheses to be tested simultaneously.

In the presence of ties or observations that are equal to `mu`, computation of exact *p*-values is not possible. Therefore, `exact` is ignored in these cases and *p*-values of the respective test settings are calculated by a normal approximation.

By setting `exact = NULL`, exact computation is performed if the sample in a test setting does not have any ties or zeros and if the sample size is lower than or equal to 200.

The used test statistics `W` is also known as $T+$ and is defined as the sum of ranks of all strictly positive values of the sample `x`.

If `digits_rank = Inf` (the default), `rank()` is used to compute ranks for the tests statistics instead of `rank(signif(., digits_rank))`

Value

If `simple.output = TRUE`, a vector of computed p-values is returned. Otherwise, the output is a `DiscreteTestResults` R6 class object, which also includes the p-value supports and testing parameters. These have to be accessed by public methods, e.g. `$get_pvalues()`.

References

Hollander, M. & Wolfe, D. (1973). *Nonparametric Statistical Methods*. Third Edition. New York: Wiley. pp. 40-55. doi:10.1002/9781119196037

See Also

`stats::wilcox.test()`, `mann_whitney_test_pv()`

Examples

```
# Constructing
set.seed(1)
r1 <- rnorm(200)
```

```
r2 <- rnorm(200, 1)
r3 <- rnorm(200, 2)

## One-sample tests
#  Exact two-sided p-values and their supports
results_ex_1s <- wilcox_test_pv(r1)
print(results_ex_1s)
results_ex_1s$get_pvalues()
results_ex_1s$get_pvalue_supports()

#  Multiple normal-approximated one-sided tests ("greater")
results_ap_1s <- wilcox_test_pv(list(r1, r2), alternative = "greater", exact = FALSE)
print(results_ap_1s)
results_ap_1s$get_pvalues()
results_ap_1s$get_pvalue_supports()

## Two-sample-tests
#  Normal-approximated one-sided p-values ("less") and their supports
results_ap_2s <- wilcox_test_pv(r1, r2, alternative = "less", exact = FALSE)
print(results_ap_2s)
results_ap_2s$get_pvalues()
results_ap_2s$get_pvalue_supports()

#  Multiple exact two-sided tests ("greater")
results_ex_2s <- wilcox_test_pv(list(r1, r3), r2)
print(results_ex_2s)
results_ex_2s$get_pvalues()
results_ex_2s$get_pvalue_supports()
```

Index