Package: TauStar (via r-universe)

October 29, 2026

Type Package

Title Efficient Computation and Testing of the Bergsma-Dassios Sign Covariance

Version 1.1.9

Description Computes the t* statistic corresponding to the tau* population coefficient introduced by Bergsma and Dassios (2014) <DOI:10.3150/13-BEJ514> and does so in O(n^2) time following the algorithm of Heller and Heller (2016) <DOI:10.48550/arXiv.1605.08732> building off of the work of Weihs, Drton, and Leung (2016) <DOI:10.1007/s00180-015-0639-x>. Also allows for independence testing using the asymptotic distribution of t* as described by Nandy, Weihs, and Drton (2016) <DOI:10.1214/16-EJS1166>.

License GPL (>= 3)

Imports Rcpp (>= 1.0.1)

LinkingTo Rcpp, RcppArmadillo

Suggests testthat

RoxygenNote 7.3.2

Encoding UTF-8

URL https://github.com/karchjd/TauStar

BugReports https://github.com/karchjd/TauStar/issues

NeedsCompilation yes

Author Luca Weihs [aut], Emin Martinian [ctb] (Created the red-black tree library included in package.), Julian D. Karch [cre] (ORCID: <https://orcid.org/0000-0002-1625-2822>)

Maintainer Julian D. Karch <j.d.karch@fsw.leidenuniv.nl>

Repository https://cran.r-universe.dev

Date/Publication 2026-03-05 15:10:03 UTC

RemoteUrl https://github.com/cran/TauStar

RemoteRef HEAD

RemoteSha 91b6068d2463a723e018b52363df79992ef386de

Contents

TauStar-package . . . 2
binaryQuantileSearch . . . 4
eigenForDiscreteProbs . . . 4
isDiscrete . . . 5
isProb . . . 5
isProbVector . . . 6
isValidDataVector . . . 6
pDisHoeffInd . . . 7
pHoeffInd . . . 8
pMixHoeffInd . . . 8
print.tstest . . . 9
tauStarTest . . . 10
tStar . . . 11

Index **14**

`TauStar-package` *Efficient Computation and Testing of the t* Statistic of Bergsma and Dassios*

Description

Computes the t* statistic corresponding to the tau star population coefficient introduced by Bergsma and Dassios (2014) <DOI:10.3150/13-BEJ514> and does so in O(n^2*log(n)) time following the algorithm of Weihs, Drton, and Leung (2016) <DOI:10.1007/s00180-015-0639-x>. Also allows for independence testing using the asymptotic distribution of t* as described by Nandy, Weihs, and Drton (2016) <http://arxiv.org/abs/1602.04387>. To directly compute the t* statistic see the function tStar. If otherwise interested in performing tests of independence then see the function tauStarTest.

Author(s)

Maintainer: Julian D. Karch <j.d.karch@fsw.leidenuniv.nl> (ORCID)

Authors:

- Luca Weihs <lucaw@uw.edu>

Other contributors:

- Emin Martinian (Created the red-black tree library included in package.) [contributor]

References

Bergsma, Wicher; Dassios, Angelos. A consistent test of independence based on a sign covariance related to Kendall's tau. *Bernoulli* 20 (2014), no. 2, 1006–1028.

Luca Weihs, Mathias Drton, and Dennis Leung. Efficient Computation of the Bergsma-Dassios Sign Covariance. *Computational Statistics*, x:x-x, 2016. to appear.

Preetam Nandy, Luca Weihs, and Mathias Drton. Large-Sample Theory for the Bergsma-Dassios Sign Covariance. arXiv preprint arXiv:1602.04387. 2016.

See Also

Useful links:

- https://github.com/karchjd/TauStar
- Report bugs at https://github.com/karchjd/TauStar/issues

Examples

```
library(TauStar)

# Compute t* for a concordant quadruple
tStar(c(1, 2, 3, 4), c(1, 2, 3, 4)) # == 2/3

# Compute t* for a discordant quadruple
tStar(c(1, 2, 3, 4), c(1, -1, 1, -1)) # == -1/3

# Compute t* on random normal iid normal data
set.seed(23421)
tStar(rnorm(4000), rnorm(4000)) # near 0

# Compute t* as a v-statistic
set.seed(923)
tStar(rnorm(100), rnorm(100), vStatistic = TRUE)

# Compute an approximation of tau* via resampling
set.seed(9492)
tStar(rnorm(10000), rnorm(10000),
    resample = TRUE, sampleSize = 30, numResamples = 5000
)

# Perform a test of independence using continuous data
set.seed(123)
x <- rnorm(100)
y <- rnorm(100)
testResults <- tauStarTest(x, y)
print(testResults$pVal) # big p-value

# Now make x and y correlated so we expect a small p-value
y <- y + x
testResults <- tauStarTest(x, y)
```

```
print(testResults$pVal) # small p-value
```

`binaryQuantileSearch` *Quantiles of a distribution.*

Description

Computes the pth quantile of a cumulative distribution function using a simple binary search algorithm. This can be extremely slow but has the benefit of being trivial to implement.

Usage

```
binaryQuantileSearch(pDistFunc, p, lastLeft, lastRight, error = 10^-4)
```

Arguments

`pDistFunc`	a cumulative distribution function on the real numbers, it should take a single argument x and return the cummulative distribution function evaluated at x.
`p`	the quantile $p \in [0, 1]$
`lastLeft`	binary search works by continuously decreasing the search space from the left and right. lastLeft should be a lower bound for the quantile p.
`lastRight`	similar to lastRight but should be an upper bound.
`error`	the error tolerated from the binary search

Value

the quantile (within error).

`eigenForDiscreteProbs` *Eigenvalues for discrete asymptotic distribution*

Description

Computes the eigenvalues needed to determine the asymptotic distributions in the mixed/discrete cases. See Nandy, Weihs, and Drton (2016) <http://arxiv.org/abs/1602.04387> for more details.

Usage

```
eigenForDiscreteProbs(p)
```

Arguments

`p`	a vector of probabilities that sum to 1.

Value

the eigenvalues associated to the matrix generated by p

isDiscrete *Heuristic check for discrete numeric data*

Description

Assumes x is numeric, length(x) >= 4, and all values are finite and non-missing.

Usage

```
isDiscrete(
  x,
  tol = sqrt(.Machine$double.eps),
  unique_ratio = 0.2,
  max_levels = 10
)
```

Arguments

x	Numeric vector.
tol	Relative tolerance for "integer-like" checks.
unique_ratio	Max ratio of unique values to n to still consider discrete (must be < 1).
max_levels	Max absolute number of unique values to still consider discrete.

Details

Uses integer-within-tolerance and low unique-to-n with repetition (via unique ratio).

Value

TRUE/FALSE best-guess.

isProb *Check if a Valid Probability*

Description

Checks if the input vector has a single entry that is between 0 and 1

Usage

```
isProb(prob)
```

Arguments

prob	the probability to check

Value

TRUE if conditions are met, FALSE if otherwise

---

isProbVector *Check if Vector of Probabilities*

Description

Checks if the input vector has entries that sum to 1 and are non-negative

Usage

```
isProbVector(probs)
```

Arguments

probs	the probability vector to check

Value

TRUE if conditions are met, FALSE if otherwise

---

isValidDataVector *Is Vector Valid Data?*

Description

Determines if input vector is a valid vector of real valued observations

Usage

```
isValidDataVector(x)
```

Arguments

x	the vector to be tested

Value

TRUE or FALSE

pDisHoeffInd *Null asymptotic distribution of t* in the discrete case*

Description

Density, distribution function, quantile function and random generation for the asymptotic null distribution of t* in the discrete case. That is, in the case that t* is generated from a sample of jointly discrete independent random variables X and Y.

Usage

```
pDisHoeffInd(x, probs1, probs2, lower.tail = TRUE, error = 10^-5)

dDisHoeffInd(x, probs1, probs2, error = 10^-3)

rDisHoeffInd(n, probs1, probs2)

qDisHoeffInd(p, probs1, probs2, error = 10^-4)
```

Arguments

x	the value (or vector of values) at which to evaluate the function.
probs1	a vector of probabilities corresponding to the (ordered) support of X. That is if your first random variable has support $u_1, ..., u_n$ then the ith entry of probs should be $P(X = u_i)$.
probs2	just as probs1 but for the second random variable Y.
lower.tail	a logical value, if TRUE (default), probabilities are $P(X \leq x)$ otherwise $P(X > x)$.
error	a tolerated error in the result. This should be considered as a guide rather than an exact upper bound to the amount of error.
n	the number of observations to return.
p	the probability (or vector of probabilities) for which to get the quantile.

Value

dDisHoeffInd gives the density, pDisHoeffInd gives the distribution function, qDisHoeffInd gives the quantile function, and rDisHoeffInd generates random samples.

pHoeffInd *Null asymptotic distribution of t* in the continuous case*

Description

Density, distribution function, quantile function and random generation for the asymptotic null distribution of t* in the continuous case. That is, in the case that t* is generated from a sample of jointly continuous independent random variables.

Usage

```
pHoeffInd(x, lower.tail = TRUE, error = 10^-5)

rHoeffInd(n)

dHoeffInd(x, error = 1/2 * 10^-3)

qHoeffInd(p, error = 10^-4)
```

Arguments

x	the value (or vector of values) at which to evaluate the function.
lower.tail	a logical value, if TRUE (default), probabilities are $P(X \leq x)$ otherwise $P(X > x)$.
error	a tolerated error in the result. This should be considered as a guide rather than an exact upper bound to the amount of error.
n	the number of observations to return.
p	the probability (or vector of probabilities) for which to get the quantile.

Value

dHoeffInd gives the density, pHoeffInd gives the distribution function, qHoeffInd gives the quantile function, and rHoeffInd generates random samples.

pMixHoeffInd *Null asymptotic distribution of t* in the mixed case*

Description

Density, distribution function, quantile function and random generation for the asymptotic null distribution of t* in the mixed case. That is, in the case that t* is generated a sample from an independent bivariate distribution where one coordinate is marginally discrete and the other marginally continuous.

Usage

```
pMixHoeffInd(x, probs, lower.tail = TRUE, error = 10^-6)

dMixHoeffInd(x, probs, error = 10^-3)

rMixHoeffInd(n, probs, error = 10^-8)

qMixHoeffInd(p, probs, error = 10^-4)
```

Arguments

x	the value (or vector of values) at which to evaluate the function.
probs	a vector of probabilities corresponding to the (ordered) support the marginally discrete random variable. That is, if the marginally discrete distribution has support $u_1, ..., u_n$ then the ith entry of probs should be the probability of seeing u_i.
lower.tail	a logical value, if TRUE (default), probabilities are $P(X \leq x)$ otherwise $P(X > x)$.
error	a tolerated error in the result. This should be considered as a guide rather than an exact upper bound to the amount of error.
n	the number of observations to return.
p	the probability (or vector of probabilities) for which to get the quantile.

Value

dMixHoeffInd gives the density, pMixHoeffInd gives the distribution function, qMixHoeffInd gives the quantile function, and rMixHoeffInd generates random samples.

---

print.tstest *Print Tau* Test Results*

Description

A simple print function for tstest (Tau* test) objects.

Usage

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

Arguments

x	the tstest object to be printed
...	ignored.

Value

No return value, prints to console.

`tauStarTest` *Test of Independence Using the Tau* Measure*

Description

Performs a (consistent) test of independence between two input vectors using the asymptotic (or permutation based) distribution of the test statistic t*. The asymptotic results hold in the case that x is generated from either a discrete or continuous distribution and similarly for y (in particular it is allowed for one to be continuous while the other is discrete). The asymptotic distributions were computed in Nandy, Weihs, and Drton (2016) <http://arxiv.org/abs/1602.04387>.

Usage

```
tauStarTest(x, y, mode = "auto", resamples = 1000)
```

Arguments

`x`	a vector of sampled values.
`y`	a vector of sampled values corresponding to x, y must be the same length as x.
`mode`	should be one of five possible values: "auto", "continuous", "discrete", "mixed", or "permutation". If "auto" is selected then the function will attempt to automatically determine whether x,y are discrete or continuous and then perform the appropriate asymptotic test. In cases "continuous", "discrete", and "mixed" we perform the associated asymptotic test making the given assumption. Finally if "permutation" is selected then the function runs a Monte-Carlo permutation test for some given number of resamplings.
`resamples`	the number of resamplings to do if mode = "permutation". Otherwise this value is ignored.

Value

a list with class "tstest" recording the outcome of the test.

References

Preetam Nandy, Luca Weihs, and Mathias Drton. Large-Sample Theory for the Bergsma-Dassios Sign Covariance. arXiv preprint arXiv:1602.04387. 2016.

Examples

```
set.seed(123)
x <- rnorm(100)
y <- rnorm(100)
testResults <- tauStarTest(x, y)
print(testResults$pVal) # big p-value

y <- y + x # make x and y correlated
testResults <- tauStarTest(x, y)
print(testResults$pVal) # small p-value
```

`tStar` *Computing t**

Description

Computes the t* U-statistic for input data pairs (x_1,y_1), (x_2,y_2), ..., (x_n,y_n) using the algorithm developed by Heller and Heller (2016) <arXiv:1605.08732> building off of the work of Weihs, Drton, and Leung (2015) <DOI:10.1007/s00180-015-0639-x>.

Usage

```
tStar(
  x,
  y,
  vStatistic = FALSE,
  resample = FALSE,
  numResamples = 500,
  sampleSize = min(length(x), 1000),
  method = "fastest",
  slow = FALSE
)
```

Arguments

`x`	A numeric vector of x values (length >= 4).
`y`	A numeric vector of y values, should be of the same length as x.
`vStatistic`	If TRUE then will compute the V-statistic version of t*, otherwise will compute the U-Statistic version of t*. Default is to compute the U-statistic.
`resample`	If TRUE then will compute an approximation of t* using a subsetting approach: samples of size sampleSize are taken from the data numResample times, t* is computed on each subsample, and all subsample t* values are then averaged. Note that this only works when vStatistic == FALSE, in general you probably don't want to compute the V-statistic via resampling as the size of the bias depends on the sampleSize irrespective numResamples. Default is resample == FALSE so that t* is computed on all of the data, this may be slow for very large sample sizes. Resampling can only be used when the method argument is using its default.

numResamples	See resample variable description for details, this value is ignored if resample == FALSE (ignored by default).
sampleSize	See resample variable description for details, this value is ignored if resample == FALSE (ignored by default).
method	which method to use to compute the statistic. Default is "fastest" which uses the fastest available method (currently "heller"). The options are "heller" described in Heller and Heller (2016), "weihs", using the algorithm from Weihs et al. (2015), and "naive" using a naive algorithm.
slow	a deprecated option kept for backwards compatibility If TRUE then will override the method parameter and compute the t* statistic using a naive O(n^4) algorithm.

Value

The numeric value of the t* statistic.

References

Bergsma, Wicher; Dassios, Angelos. A consistent test of independence based on a sign covariance related to Kendall's tau. *Bernoulli* 20 (2014), no. 2, 1006–1028.

Heller, Yair and Heller, Ruth. "Computing the Bergsma Dassios sign-covariance." arXiv preprint arXiv:1605.08732 (2016).

Weihs, Luca, Mathias Drton, and Dennis Leung. "Efficient Computation of the Bergsma-Dassios Sign Covariance." arXiv preprint arXiv:1504.00964 (2015).

Examples

```
library(TauStar)

# Compute t* for a concordant quadruple
tStar(c(1, 2, 3, 4), c(1, 2, 3, 4)) # == 2/3

# Compute t* for a discordant quadruple
tStar(c(1, 2, 3, 4), c(1, -1, 1, -1)) # == -1/3

# Compute t* on random normal iid normal data
set.seed(23421)
tStar(rnorm(4000), rnorm(4000)) # near 0

# Compute t* as a v-statistic
set.seed(923)
tStar(rnorm(100), rnorm(100), vStatistic = TRUE)

# Compute an approximation of tau* via resampling
set.seed(9492)
tStar(rnorm(10000), rnorm(10000),
  resample = TRUE, sampleSize = 30,
  numResamples = 5000
```

)

Index

binaryQuantileSearch, 4

dDisHoeffInd *(*pDisHoeffInd*)*, 7
dHoeffInd *(*pHoeffInd*)*, 8
dMixHoeffInd *(*pMixHoeffInd*)*, 8

eigenForDiscreteProbs, 4

isDiscrete, 5
isProb, 5
isProbVector, 6
isValidDataVector, 6

pDisHoeffInd, 7
pHoeffInd, 8
pMixHoeffInd, 8
print.tstest, 9

qDisHoeffInd *(*pDisHoeffInd*)*, 7
qHoeffInd *(*pHoeffInd*)*, 8
qMixHoeffInd *(*pMixHoeffInd*)*, 8

rDisHoeffInd *(*pDisHoeffInd*)*, 7
rHoeffInd *(*pHoeffInd*)*, 8
rMixHoeffInd *(*pMixHoeffInd*)*, 8

TauStar *(*TauStar-package*)*, 2
TauStar-package, 2
tauStarTest, 10
tStar, 11