Package 'multivariance'

June 18, 2019

Title Measuring Multivariate Dependence Using Distance Multivariance

Version 2.2.0

Date 2019-06-18

Description Distance multivariance is a measure of dependence which can be used to detect and quantify dependence. The necessary functions are implemented in this packages, and examples are given. For the theoretic background we refer to the papers:
B. Böttcher, Dependence and Dependence Structures: Estimation and Visualization Using Distance Multivariance. <arXiv:1712.06532>.
B. Böttcher, M. Keller-Ressel, R.L. Schilling, Detecting independence of random vectors: generalized distance covariance and Gaussian covariance. VMSTA, 2018, Vol. 5, No. 3, 353-383. <arXiv:1711.07778>.
B. Böttcher, M. Keller-Ressel, R.L. Schilling, Distance multivariance: New dependence measures for random vectors. <arXiv:1711.07775>.
G. Berschneider, B. Böttcher, On complex Gaussian random fields, Gaussian quadratic forms and sample distance multivariance. <arXiv:1808.07280>.

Depends R (>= 3.3.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports igraph, graphics, stats, Rcpp, microbenchmark

RoxygenNote 6.1.1

Suggests testthat

LinkingTo Rcpp

NeedsCompilation yes

Author Björn Böttcher [aut, cre],
Martin Keller-Ressel [ctb]

Maintainer Björn Böttcher <bjoern.boettcher@tu-dresden.de>

Repository CRAN

Date/Publication 2019-06-18 12:20:03 UTC

R topics documented:

`multivariance-package` *multivariance: Measuring Multivariate Dependence Using Distance Multivariance*

Description

The multivariance package provides basic functions to calculate distance multivariance and related quantities. To test independence use `multivariance.test`, it provides an interface (via its arguments) to all the tests based on distance (m-/total-)multivariance. The package offers also several other functions related to distance multivariance, e.g. a detection and visualization of dependence structures `dependence.structure`. See below for details on the full content of the package.

Details

Distance multivariance is a multivariate dependence measure, which can be used to detect dependencies between an arbitrary number of random vectors each of which can have a distinct dimension. The necessary functions are implemented in this package, and examples are given. For the theoretic background we refer to the papers [1,2,3,4]. Paper [3] includes a summary of the first two. It is the recommended starting point for users with an applied interest. Paper [4] is concerned with new (faster) p-value estimates for the independence tests.

The (current) code is tested and speed improved in comparison to the former releases. Certainly there is still room for improvement and development. Questions, comments and remarks are welcome: <bjoern.boettcher@tu-dresden.de>

For infos on the latest changes and/or updates to the package use `news(package="multivariance")`.

To cite this package use the standard citation for R packages, i.e., the output of `citation("multivariance")`.

Multivariance

`multivariance` computes the distance multivariance

`total.multivariance` computes the total distance multivariance

`m.multivariance` computes the m-multivariance (introduced in [3])

It might be convenient to compute these simultaneously using `multivariances.all`.

Functions to use and interpret multivariance

`rejection.level` computes a (conservative) rejection level for a given significance level. This can be used for a conservative interpretation of distance multivariance. The counterpart is `multivariance.pvalue`, which computes a conservative p-value for a given distance multivariance. Both methods are distribution-free.

`resample.rejection.level` and `resample.pvalue` are the distribution dependent versions of the above. They are approximately sharp, but computational more expensive. Any resampling is done by `resample.multivariance`.

Using the methods developed in [4] approximate p-value estimates are provided by `pearson.pvalue`. This method is much faster than the resampling method.

`multivariance.test` and `independence.test` provide the corresponding tests of independence. The former provides output as common for tests in R, the latter is a simple significance test returning for a given significance level just `TRUE` or `FALSE`

`cdm` and `cdms` compute the (doubly) centered distance matrix and matrices, respectively. These can be used to speed up repeated computations of distance multivariance.

In [4] various methods to estimate the moments of the test statistic under H0 were developed, these are (implicitly) implemented in this package only for the moments used in `pearson.pvalue`. Further and explicit functions can be added upon request. Please feel free to contact the author.

For planing of large projects or studies it might be convenient to estimate the computation time of multivariance via `multivariance.timing`.

Dependence structures

dependence.structure performs the dependence structure detection algorithm as described in [3].

find.cluster is the basic building block of dependence.structure. It is recommended to use dependence.structure.

Examples

coins and tetrahedron generate samples of pairwise independent random variables, with dependence of higher order.

dep_struct_iterated_13_100, dep_struct_ring_15_100, dep_struct_several_26_100 and dep_struct_star_9_100 are example data sets for the dependence structure detection. These might also serve as benchmark examples.

References

[1] B. Böttcher, M. Keller-Ressel, R.L. Schilling, Detecting independence of random vectors: generalized distance covariance and Gaussian covariance. Modern Stochastics: Theory and Applications 2018, Vol. 5, No. 3, 353-383. https://www.vmsta.org/journal/VMSTA/article/127/info

[2] B. Böttcher, M. Keller-Ressel, R.L. Schilling, Distance multivariance: New dependence measures for random vectors. Accepted for publication in Annals of Statistics. https://arxiv.org/abs/1711.07775

[3] B. Böttcher, Dependence and Dependence Structures: Estimation and Visualization Using Distance Multivariance. Preprint. https://arxiv.org/abs/1712.06532

[4] G. Berschneider, B. Böttcher, On complex Gaussian random fields, Gaussian quadratic forms and sample distance multivariance. Preprint. https://arxiv.org/abs/1808.07280

---

cdm *centered distance matrix*

Description

computes the centered distance matrix

Usage

```
cdm(x, normalize = TRUE, psi = NULL, p = NULL, isotropic = FALSE,
  external.dm.fun = NULL)
```

Arguments

x	matrix, each row of the matrix is treated as one sample
normalize	logical, indicates if the matrix should be normalized

psi — if it is NULL, the euclidean distance will be used. In the case of isotropic = TRUE: a real valued negative definite function of one variable (accepting vectors as arguments; returning a vector of the same length). In the case of isotropic = FALSE: a real valued function of two variables (or vectors) to compute the distance of two samples based on a continuous negative definite function.

p — numeric, if it is a value between 1 and 2 then the Minkowski distance with parameter p is used.

isotropic — logical, indicates if psi of the Euclidean distance matrix should be computed, i.e., if an isotropic distance should be used.

external.dm.fun — here one can supply an external function, which computes the distance matrix given x.

Details

The centered distance matrices are required for the computation of (total / m-) multivariance.

If normalize = TRUE then the value of multivariance is comparable and meaningful. It can be compared to the rejection.level or its p-value multivariance.pvalue can be computed.

More details: If normalize = TRUE the matrix is scaled such that the multivariance based on it, times the sample size, has in the limit - in the case of independence - the distribution of an L^2 norm of a Gaussian process with known expectation.

As default the Euclidean distance is used. The parameters psi, p, isotropic and external.dm.fun can be used to select a different distance. In particular, external.dm.fun can be used to provide any function which calculates a distance matrix for the rows of a given matrix.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

Examples

```
x = coins(100)
cdm(x) # fast euclidean distances
cdm(x,psi = function(x,y) sqrt(sum((x-y)^2))) # this is identical to the previous (but slower)

# the function cdm does the following three lines in a faster way
N = nrow(x)
C = diag(N) - matrix(1/N,nrow = N,ncol = N)
A = - C %*% as.matrix(stats::dist(x,method="euclidean")) %*% C #'
all(abs(A- cdm(x,normalize = FALSE)) < 10^(-12))
```

cdms *computes the centered distance matrices*

Description

computes the centered distance matrices

Usage

```
cdms(x, vec = 1:ncol(x), membership = NULL, ...)
```

Arguments

x	matrix, each row is a sample
vec	vector which indicates which columns are treated as one sample
membership	depreciated. Now use vec.
...	these are passed to cdm

Value

It returns a list of distance matrices.

clean.graph *cleanup dependence structure graph*

Description

Given a dependence structure graph: vertices representing the multivariances of only two vertices can be turned into an edge labeled with the label of the vertex. Moreover, only subsets of the graph can be selected.

Usage

```
clean.graph(g, only.level = NULL, simplify.pairs = TRUE,
  drop.label.pairs = FALSE)
```

Arguments

g	graph, created by dependence.structure
only.level	integer vector, if provided all edges and dependency nodes corresponding to dependence orders not given in 'only.level' are removed
simplify.pairs	boolean, if true dependency nodes which are only connected to two variables are turned into edges
drop.label.pairs	boolean, if true the labels for edges indicating pairwise dependence are removed

Details

Note: The option 'only.level' works only properly for a full dependence structure graph, in the case of a clustered dependence structure graph dependency nodes representing a cluster might be removed.

Value

graph

Examples

```
N = 200
y = coins(N,2)
x = cbind(y,y,y)

ds = dependence.structure(x,structure.type = "clustered")
plot(clean.graph(ds$graph))
plot(clean.graph(ds$graph,only.level = 2))
plot(clean.graph(ds$graph,only.level = 3)) # of limited use for a clustered graph,
# i.e., here the three-dependence node without edges indicates that
# all edges were connected to clusters

ds = dependence.structure(x,structure.type = "full")
plot(clean.graph(ds$graph))
plot(clean.graph(ds$graph,drop.label.pairs = TRUE))
plot(clean.graph(ds$graph,only.level = 2))
plot(clean.graph(ds$graph,only.level = 2,drop.label.pairs = TRUE))
plot(clean.graph(ds$graph,only.level = 3))
```

---

coins *dependence example: k-independent coin sampling*

Description

This function creates samples which are dependent but k-independent.

Usage

```
coins(N = 1000, k = 2)
```

Arguments

N	number of samples
k	each k-tuple will be independent

Value

It returns the samples as rows of an N by k+1 matrix. The columns are dependent but k-independent.

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
coins(200,4)
```

`dependence.structure` *determines the dependence structure*

Description

Determines the dependence structure as described in [3].

Usage

```
dependence.structure(x, vec = 1:ncol(x), verbose = TRUE,
  detection.aim = NULL, type = "conservative",
  structure.type = "clustered", c.factor = 2, list.cdm = NULL,
  alpha = 0.05, p.adjust.method = "holm", stop.too.many = NULL, ...)
```

Arguments

`x`	matrix, each row of the matrix is treated as one sample
`vec`	vector, it indicates which columns are initially treated together as one sample
`verbose`	boolean, if `TRUE` details are printed during the detection and whenever a cluster is newly detected the (so far) detected dependence structure is plotted.
`detection.aim`	`=NULL` or a list of vectors which indicate the expected detection, see below for more details
`type`	the method used for the detection, one of '`conservative`','`resample`','`pearson_approx`' or '`consistent`'
`structure.type`	either the '`clustered`' or the '`full`' structure is detected
`c.factor`	numeric, larger than 0, a constant factor used in the case of '`type = "consistent"`'
`list.cdm`	not required, the list of centered distance matrices corresponding to `x` speeds up the computation if given
`alpha`	numeric between 0 and 1, the significance level used for the tests
`p.adjust.method`	a string indicating the p-value adjustment for multiple testing, see `p.adjust.methods`
`stop.too.many`	numeric, upper limit for the number of tested tuples. A warning is issued if it is used. Use `stop.too.many = NULL` for no limit.
`...`	these are passed to `find.cluster`

Details

Performs the detection of the dependence structure as described in [3] (the options 'structure.type = "full"' and 'type = "consistent"' are part of a revised version of the paper). In the clustered structure variables are clustered and treated as one variable as soon as a dependence is detected, the full structure treats always each variable separately. The detection is either based on tests with significance level alpha or a consistent estimator is used. The latter yields (in the limit for increasing sample size) under very mild conditions always the correct dependence structure (but the convergence might be very slow).

If fixed.rejection.level is not provided, the significance level alpha is used to determine which multivariances are significant using the distribution-free rejection level. As default the Holm method is used for p-value correction corresponding to multiple testing.

The resulting graph can be simplified (pairwise dependence can be represented by edges instead of vertices) using clean.graph.

Advanced: The argument detection.aim is currently only implemented for structure.type = clustered. It can be used to check, if an expected dependence structure was detected. This might be useful for simulation studies to determine the empirical power of the detection algorithm. Hereto detection.aim is set to a list of vectors which indicate the expected detected dependence structures (one for each run of find.cluster). The vector has as first element the k for which k-tuples are detected (for this aim the detection stops without success if no k-tuple is found), and the other elements, indicate to which clusters all present vertices belong after the detection, e.g. c(3,2,2,1,2,1,1,2,1) expects that 3-tuples are detected and in the graph are 8 vertices (including those representing the detected 3 dependencies), the order of the 2's and 1's indicate which vertices belong to which cluster. If detection.aim is provided, the vector representing the actual detection is printed, thus one can use the output with copy-paste to fix successively the expected detection aims.

Note that a failed detection might invoke the warning:

```
run$mem == detection.aim[[k]][-1] :
longer object length is not a multiple of shorter object length
```

Value

returns a list with elements:

multivariances calculated multivariances,

cdms calculated centered distance matrices,

graph graph representing the dependence structure,

detected boolean, this is only included if a detection.aim is given,

number.of.dep.tuples vector, with the number of dependent tuples for each tested order. For the full dependence structure a value of -1 indicates that all tuples of this order are already lower order dependent, a value of -2 indicates that there were more than stop.too.many tuples,

structure.type either clustered or full,

type the type of p-value estimation or consistent estimation used,

total.number.of.tests numeric vector, with the number of tests for each group of tests,

typeI.error.prob estimated probability of a type I error,

`alpha` significance level used if a p-value estimation procedure is used,

`c.factor` factor used if a consistent estimation procedure is used,

`parameter.range` significance levels (or 'c.factor' values) which yield the same detection result.

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
# structures for the datasets included in the package
dependence.structure(dep_struct_several_26_100)
dependence.structure(dep_struct_star_9_100)
dependence.structure(dep_struct_iterated_13_100)
dependence.structure(dep_struct_ring_15_100)

# basic examples:

dependence.structure(coins(100)) # 3-dependent
dependence.structure(coins(100),vec = c(1,1,2))
# 3-dependent rv of which the first two rv are used together as one rv, thus 2-dependence.

dependence.structure(cbind(coins(200),coins(200,k=5)),verbose = TRUE)
#1,2,3 are 3-dependent, 4,..,9 are 6-dependent

# similar to the the previous example, but
# the pair 1,3 is treated as one sample,
# anagously the pair 2,4. In the resulting structure one does not
# see anymore that the dependence of 1,2,3,4 with the rest is due
# to 4.
dependence.structure(cbind(coins(200),coins(200,k=5)),
                           vec = c(1,2,1,2,3,4,5,6,7),verbose = TRUE)


### Advanced:

# How to check the empirical power of the detection algorithm?
# Use a dataset for which the structure is detected, e.g. dep_struct_several_26_100.
# run:
dependence.structure(dep_struct_several_26_100,
                     detection.aim = list(c(ncol(dep_struct_several_26_100))))
# The output provides the first detection aim. Now we run the same line with the added
# detection aim
dependence.structure(dep_struct_several_26_100,detection.aim = list(c(3,1, 1, 1, 2, 2, 2, 3, 4,
  5, 6, 7, 8, 8, 8, 9, 9, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1, 2, 8, 9),
  c(ncol(dep_struct_several_26_100))))
# and get the next detection aim ... thus we finally obtain all detection aims.
# now we can run the code with new sample data ....
N = 100
dependence.structure(cbind(coins(N,2),tetrahedron(N),coins(N,4),tetrahedron(N),
```

```
                          tetrahedron(N),coins(N,3),coins(N,3),rnorm(N)),
                      detection.aim = list(c(3,1, 1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 8, 8,
  9, 9, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1, 2, 8, 9),
  c(4,1, 1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 10, 11, 11, 11,
    11, 12, 1, 2, 8, 9, 10, 11),
  c(5, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 1,
    2, 4, 5, 6, 7, 3),
  c(5, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 1,
    2, 4, 5, 6, 7, 3)))$detected
# ... and one could start to store the results and compute the rate of successes.

# ... or one could try to check how many samples are necessary for the detection:
re = numeric(100)
for (i in 2:100) {
  re[i] =
    dependence.structure(dep_struct_several_26_100[1:i,],verbose = FALSE,
                         detection.aim = list(c(3,1, 1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8,
      8, 8, 9, 9, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1, 2, 8, 9),
      c(4,1, 1, 1, 2, 2, 2, 3, 4, 5, 6, 7, 8, 8, 8, 9, 9, 9, 10, 10, 10, 10, 11, 11,
        11, 11, 12, 1, 2, 8, 9, 10, 11),
      c(5, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7,
        8, 1, 2, 4, 5, 6, 7, 3),
      c(5, 1, 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7,
        8, 1, 2, 4, 5, 6, 7, 3)))$detected
  print(paste("First", i,"samples. Detected?", re[i]==1))
}
cat(paste("Given the 1 to k'th row the structure is not detected for k =",which(re == FALSE),"\n"))
```

---

dep_struct_iterated_13_100

example dataset for dependence.structure

Description

It was generated by

```
set.seed(532333356)
N = 100
x = matrix(sample.int(2,10*N,replace = TRUE)-1,ncol = 10)
for (i in c(2,5,9)) x = cbind(x,(rowSums(as.matrix(x[,1:(i-1)])))
dep_struct_iterated_13_100 = x
save(dep_struct_iterated_13_100,file ="dep_struct_iterated_13_100.rda")
```

Usage

```
dep_struct_iterated_13_100
```

Format

matrix 13 variables (columns), 100 independent samples (rows)

Details

To avoid irritation, note that the seed is just a simple integer hash value of the variable name.

---

dep_struct_ring_15_100

example dataset for dependence.structure

Description

It was generated by

```
set.seed(436646700)
N = 100
n= 15
x=matrix(sample.int(2,N*n,replace = TRUE)-1,nrow =N)
x[,4] = rowSums(x[,1:3])
x[,7] = rowSums(x[,4:6])
x[,10] = rowSums(x[,7:9])
x[,13] = rowSums(x[,10:12])
x[,15] = rowSums(x[,c(13,14,1)])
dep_struct_ring_15_100 = x
save(dep_struct_ring_15_100,file ="dep_struct_ring_15_100.rda")
```

Usage

```
dep_struct_ring_15_100
```

Format

matrix 15 variables (columns), 100 independent samples (rows)

Details

To avoid irritation, note that the seed is just a simple integer hash value of the variable name.

---

dep_struct_several_26_100 *example dataset for* dependence.structure

---

Description

It was generated by

```
set.seed(1348879148)
N = 100
dep_struct_several_26_100 = cbind(coins(N,2),tetrahedron(N),coins(N,4),
    tetrahedron(N),tetrahedron(N),coins(N,3),coins(N,3),rnorm(N))
save(dep_struct_several_26_100,file ="dep_struct_several_26_100.rda")
```

Usage

```
dep_struct_several_26_100
```

Format

matrix 26 variables (columns), 100 independent samples (rows)

Details

To avoid irritation, note that the seed is just a simple integer hash value of the variable name.

---

dep_struct_star_9_100 *example dataset for* dependence.structure

---

Description

It was generated by

```
set.seed(222454572)
N = 100
y = coins(N,2)
dep_struct_star_9_100 = cbind(y,y,y)
save(dep_struct_star_9_100,file ="dep_struct_star_9_100.rda")
```

Usage

```
dep_struct_star_9_100
```

Format

matrix 9 variables (columns), 100 independent samples (rows)

Details

To avoid irritation, note that the seed is just a simple integer hash value of the variable name.

`fastdist` *fast Euclidean distance matrix*

Description

fast Euclidean distance matrix

Usage

```
fastdist(x)
```

Arguments

`x`	matrix with sample rows for which the distance matrix is computed (to use with vectors, use `as.matrix(x)`)

Examples

```
#require(microbenchmark)
#x = rnorm(100)
#microbenchmark(fastdist(as.matrix(x)),as.matrix(dist(x)))
```

`fastEuclideanCdm` *fast centered Euclidean distance matrix*

Description

fast centered Euclidean distance matrix

Usage

```
fastEuclideanCdm(x, normalize)
```

Arguments

`x`	matrix with sample rows for which the distance matrix is computed (to use with vectors, use `as.matrix(x)`)
`normalize`	boolean. If `TRUE` the matrix will be normalized to mean 1.

`find.cluster` *cluster detection*

Description

Performs the detection of dependence structures algorithm until a cluster is found. This function is the basic building block `dependence.structure`. Advanced users, might use it directly.

Usage

```
find.cluster(x, vec = 1:ncol(x), list.cdm = cdms(x, vec = vec),
  mem = as.numeric(1:max(vec)), cluster.to.vertex = 1:max(mem),
  vertex.to.cdm = 1:max(mem), previous.n.o.cdms = rep(0, max(mem)),
  all.multivariances = numeric(0),
  g = igraph::add.vertices(igraph::graph.empty(, directed = FALSE),
  max(mem), label = sapply(1:max(mem), function(r) paste(colnames(x,
  do.NULL = FALSE, prefix = "")[vec == r], collapse = ",")), shape =
  "circle"), fixed.rejection.level = NA, alpha = 0.05,
  p.adjust.method = "holm", verbose = TRUE, kvec = 2:max(mem),
  parameter.range = NULL, type = "conservative",
  stop.too.many = NULL, ...)
```

Arguments

`x`	matrix with the samples
`vec`	vector, it indicates which columns are initially treated together as one sample
`list.cdm`	list of centered distance matrices
`mem`	numeric vector, its length is the number of vertices, its content is the number of the corresponding cluster for the current iteration, i.e., vertex `i` belongs to cluster `mem[i]`
`cluster.to.vertex`	vector, contains the cluster to vertex relations, i.e., `cluster.to.vertex[i]` is the index of the vertex which represents cluster `i`
`vertex.to.cdm`	vector, contains the vertex to centered distance matrix relations, i.e., `vertex.to.cdm[i]` is the index centered distance matrix in `list.cdm` which corresponds to vertex `i`
`previous.n.o.cdms`	vector, number of centered distance matrices in the previous iteration (it is used to ensure that previously check tuples are not checked again)
`all.multivariances`	vector, which contains all distance multivariances which have been calculated so far. Only used to finally return all distance multivariances which have been calculated.
`g`	dependence structure graph

`fixed.rejection.level`
: vector, if not `NA` the `fixed.rejection.level[k]` is used for the k-tuples, instead of a level derived from the significance level `alpha`

`alpha`
: numeric, significance level used for the (distribution-free) tests

`p.adjust.method`
: name of the method used to adjust the p-values for multiple testing, see `p.adjust` for all possible options.

`verbose`
: boolean, if `TRUE` details during the detection are printed and whenever a cluster is newly detected the (so far) detected dependence structure is plotted.

`kvec`
: vector, k-tuples are only checked for each k in `kvec`, i.e., for `kvec = 2:4` only 2,3 and 4-tuples would be check and then the algorithm stops.

`parameter.range`
: numeric matrix, which hosts the range of significance levels or '`c.factor`' which yield the same detected structure

`type`
: the method for the detection, one of '`conservative`','`resample`','`pearson_approx`' or '`consistent`'.

`stop.too.many`
: numeric, upper limit for the number of tested tuples. A warning is issued if it is used. Use `stop.too.many = NULL` for no limit.

`...`
: are passed to `resample.multivariance` in the case of '`type = resample`'

Details

For further details see `dependence.structure`.

`independence.test` *test for independence*

Description

This computes a test of independence for the columns of a sample matrix (required for the resampling test) or for given centered distance matrices (only possible for the distribution-free test).

Usage

```
independence.test(x, vec = 1:ncol(x), alpha = 0.05,
  type = "distribution_free", verbose = TRUE, ...)
```

Arguments

`x`
: either a data matrix or a list of centered distance matrices

`vec`
: if x is a matrix, then this indicates which columns are treated together as one sample; if x is a list, these are the indexes for which the multivariance is calculated. The default is all columns and all indexes, respectively.

`alpha`
: significance level

`type`
: one of `"pearson_approx"`,`"distribution_free"`,`"resample"`

`verbose`
: logical, if TRUE meaningful text output is generated.

`...`
: these are passed to `cdms` (which is only invoked if x is a matrix)

Details

For a test with p-value output (as standard for tests in R) see `multivariance.test`.

The `"pearson_approx"` and `"resample"` are approximately sharp. The latter is based on a resampling approach and thus much slower. The `"distribution_free"` test might be very conservative. The centered distance matrices can be prepared by `cdms`. But note that for the test based on Pearson's approximation and for the resampling test, the data matrix has to be given.

Value

Returns `TRUE` if the hypothesis of independence is NOT rejected, otherwise `FALSE`.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

Examples

```
independence.test(coins(100)) #dependent sample which is 2-independent
independence.test(coins(100),type = "resample") #dependent sample which is 2-independent

independence.test(coins(100)[,2:3]) # independent sample
independence.test(coins(100)[,2:3],type = "resample") # independent sample

independence.test(coins(10),type = "resample") #dependent sample which is 2-independent
independence.test(coins(10)[,2:3],type = "resample") #dependent sample which is 2-independent
```

---

layout_on_circles *A special igraph layout for the dependence structure visualization*

Description

It places the variable nodes on an outer circle and the dependency nodes on an inner circle

Usage

```
layout_on_circles(g, n = sum(is.na(igraph::V(g)$level)))
```

Arguments

g	graph
n	number of vertices on outer circle

Details

This is the standard layout for the full dependence structure, since in this case there often too many nodes which make the other (usual) layout incomprehensible.

Examples

```
N = 200
y = coins(N,2)
x = cbind(y,y,y)

g = dependence.structure(x,structure.type = "clustered",verbose = FALSE)$graph
plot(g)
plot(g,layout = layout_on_circles(g))
```

`m.multivariance` *m distance multivariance*

Description

Computes m distance multivariance.

Usage

```
m.multivariance(x, vec = NA, m = 2, Nscale = TRUE, Escale = TRUE,
  squared = TRUE, ...)
```

Arguments

`x`	either a data matrix or a list of centered distance matrices
`vec`	if x is a matrix, then this indicates which columns are treated together as one sample; if x is a list, these are the indexes for which the multivariance is calculated. The default is all columns and all indexes, respectively.
`m`	=2 or 3 the m-multivariance will be computed.
`Nscale`	if `TRUE` the multivariance is scaled up by the sample size (and thus it is exactly as required for the test of independence)
`Escale`	if `TRUE` then it is scaled by the number of multivariances which are theoretically summed up (in the case of independence this yields for normalized distance matrices an estimator with expectation 1)
`squared`	if `FALSE` it returns the actual multivariance, otherwise the squared multivariance (less computation)
`...`	these are passed to `cdms` (which is only invoked if `x` is a matrix)

Details

m-distance multivariance is per definition the scaled sum of certain distance multivariances, and it characterize m-dependence.

As a rough guide to interpret the value of total distance multivariance note:

- Large values indicate dependence.

- If the random variables are (m-1)-independent and `Nscale = TRUE`, values close to 1 and smaller indicate m-independence, larger values indicate dependence. In fact, in the case of independence the test statistic is a Gaussian quadratic form with expectation 1 and samples of it can be generated by `resample.multivariance`.
- If the random variables are (m-1)-independent and `Nscale = FALSE`, small values (close to 0) indicate m-independence, larger values indicate dependence.

Since random variables are always 1-independent, the case `m=2` characterizes pairwise independence.

Finally note, that due to numerical (in)precision the value of m-multivariance might become negative. In these cases it is set to 0. A warning is issued, if the value is negative and further than the usual (used by `all.equal`) tolerance away from 0.

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
x = matrix(rnorm(3*30),ncol = 3)

# the following values are identical
m.multivariance(x,m =2)
1/choose(3,2)*(multivariance(x[,c(1,2)]) +
               multivariance(x[,c(1,3)]) +
               multivariance(x[,c(2,3)]))

# the following values are identical
m.multivariance(x,m=3)
multivariance(x)

# the following values are identical
1/4*(3*(m.multivariance(x,m=2)) + m.multivariance(x,m=3))
total.multivariance(x, Nscale = TRUE)
1/4*(multivariance(x[,c(1,2)], Nscale = TRUE) +
     multivariance(x[,c(1,3)], Nscale = TRUE) +
     multivariance(x[,c(2,3)], Nscale = TRUE) + multivariance(x, Nscale = TRUE))
```

`multicorrelation` *distance multicorrelation*

Description

Computes various types of sample distance multicorrelation as defined in [3].

Usage

```
multicorrelation(x, vec = 1:ncol(x), type = "m.multi.2",
  multicorrelation.type = "unnormalized", squared = TRUE, ...)
```

Arguments

x	either a data matrix or a list of centered distance matrices
vec	if x is a matrix, then this indicates which columns are treated together as one sample; if x is a list, these are the indexes for which the multivariance is calculated. The default is all columns and all indexes, respectively.
type	one of "m.multi.2","multi","m.multi.3","(lower bound) total"
multicorrelation.type	one of "normalized","unnormalized"
squared	if FALSE it returns the actual multivariance, otherwise the squared multivariance (less computation)
...	these are passed to cdms (which is only invoked if x is a matrix)

Details

The unnormalized and normalized versions coincide if an even number of variables is considered (in particular always for 2-multivariance). They usually differ if an odd number of variables is considered (always for 3-multivariance). If all variables are related by similarity transforms the unnormalized "unnormalized" multicorrelations are 1.

For "m.multi.2" the empirical 2-multicorrelation is computed. Which is a dependence measure for pairwise dependence, i.e. the 3-multicorrelation is 0 if and only if all variables are pairwise independent.

For total multicorrelation there is currently only a feasible empirical estimator for a lower bound ("(lower bound) total"). But it still characterizes dependence in the sense that the population version of this bound is 0 if and only if the variables are independent.

A value of 0 of multicorrelation "multi" or 3-multicorrelation "m.multi.3" does not characterize independence.

Value

Value of the multicorrelation.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

Examples

```
y = rnorm(100)
x = cbind(y,y*2,(y-2)/3,y+1,y*5) # all variables are related by similarity transforms

# compute all types of correlations for x:
```

```
for (ty in c("(lower bound) total","m.multi.2","m.multi.3","multi"))
 for (mty in c("normalized","unnormalized"))
  print(paste(format(multicorrelation(x,type=ty,multicorrelation.type = mty)
                     ,digits=3,nsmall = 3,width = 7),mty,ty,"correlation"))
```

`multivariance` *distance multivariance*

Description

Computes the distance multivariance, either for given data or a given list of centered distance matrices.

Usage

```
multivariance(x, vec = NA, Nscale = TRUE, correlation = FALSE,
  squared = TRUE, ...)
```

Arguments

`x`	either a data matrix or a list of centered distance matrices
`vec`	if x is a matrix, then this indicates which columns are treated together as one sample; if x is a list, these are the indexes for which the multivariance is calculated. The default is all columns and all indexes, respectively.
`Nscale`	if `TRUE` the multivariance is scaled up by the sample size (and thus it is exactly as required for the test of independence)
`correlation`	depreciated, please use the function `multicorrelation` instead.
`squared`	if `FALSE` it returns the actual multivariance, otherwise the squared multivariance (less computation)
`...`	these are passed to `cdms` (which is only invoked if `x` is a matrix)

Details

If `x` is an matrix and `vec` is not given, then each column is treated as a separate sample. Otherwise `vec` has to have as many elements as `x` has columns and values starting from 1 up to the number of 'variables', e.g. if `x` is an `N` by 5 matrix and `vec = c(1,2,1,3,1)` then the multivariance of the 1-dimensional variables represented by column 2 and 4 and the 3-dimensional variable represented by the columns 1,3,5 is computed.

As default it computes the normalized Nscaled squared multivariance, for a multivariance without normalization the argument `normalize = FALSE` has to be passed to `cdms`.

`correlation = TRUE` yields values between 0 and 1. These can be interpreted similarly to classical correlations, see also `multicorrelation`.

As a rough guide to interpret the value of distance multivariance note:

- If the random variables are not (n-1)-independent, large values indicate dependence, but small values are meaningless. Thus in this case use `total.multivariance`.

- If the random variables are (n-1)-independent and `Nscale = TRUE`, values close to 1 and smaller indicate independence, larger values indicate dependence. In fact, in the case of independence the test statistic is a Gaussian quadratic form with expectation 1 and samples of it can be generated by `resample.multivariance`.
- If the random variables are (n-1)-independent and `Nscale = FALSE`, small values (close to 0) indicate independence, larger values indicate dependence.

Finally note, that due to numerical (in)precision the value of multivariance might become negative. In these cases it is set to 0. A warning is issued, if the value is negative and further than the usual (used by `all.equal`) tolerance away from 0.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

Examples

```
multivariance(matrix(rnorm(100*3),ncol = 3)) #independent sample
multivariance(coins(100)) #dependent sample which is 2-independent

x = matrix(rnorm(100*2),ncol = 2)
x = cbind(x,x[,2])
multivariance(x) #dependent sample which is not 2-independent (thus small values are meaningless!)
multivariance(x[,1:2]) #these are independent
multivariance(x[,2:3]) #these are dependent

multivariance(x[,2:3],correlation = TRUE)
```

`multivariance.pvalue` *transform multivariance to p-value*

Description

Computes a conservative p-value for the hypothesis of independence for a given multivariance / m-multivariance / total multivariance.

Usage

```
multivariance.pvalue(x)
```

Arguments

`x`	value of a normalized `multivariance` scaled by the sample size (i.e., computed with `Nscale = TRUE`)

Details

This is based on a distribution-free approach. The p-value is conservative, i.e. it might be much smaller. This is the counterpart to `rejection.level`. For a less conservative approach see `resample.pvalue` or `pearson.pvalue`.

p-values larger than 0.215 might be incorrect, since the distribution-free estimate on which the computation is based only holds up to 0.215.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

`multivariance.test` *independence tests based on (total-/2-/3-) multivariance*

Description

This performs the (specified by `type` and `p.value.type`) independence test for the columns of a sample matrix.

Usage

```
multivariance.test(x, vec = 1:ncol(x), type = "total",
  p.value.type = "distribution_free", verbose = TRUE, ...)
```

Arguments

`x`	matrix, each row is a sample
`vec`	vector which indicates which columns are treated as one sample
`type`	one of `"independence"`, `"pairwise independence"`, `"multi"`, `"total"`, `"m.multi.2"`, `"m.multi.3"`
`p.value.type`	one of `"pearson_approx"`, `"distribution_free"`, `"resample"`
`verbose`	logical, if TRUE meaningful text output is generated.
`...`	these are passed to `cdm`

Details

For the use of `vec` see the examples below and the more detailed explanation of this argument for `multivariance`.

The types `"independence"` and `"total"` are identical: an independence test is performed.

Also the types `"pairwise independence"` and `"m.multi.2"` are identical: a test of pairwise independence is performed.

The type `"m.multi.3"`, performs a test for 3-independence, assuming pairwise independence. The type `"multi"` performs a test for n-independence, assuming (n-1)-independence.

There are several ways (determined by `p.value.type`) to estimate the p-value: The `"pearson_approx"` and `"resample"` are approximately sharp. The latter is based on a resampling approach and thus much slower. The `"distribution_free"` test might be very conservative, its p-value estimates are only valid for p-values lower than 0.215 - values above should be interpreted as "values larger than 0.215".

All tests are performed using the standard euclidean distance. Other distances can be supplied via the `...`, see `cdm` for the accepted arguments.

Value

A list with class `"htest"` containing the following components:

`statistic` the value of the test statistic,

`p.value` the p-value of the test statistic,

`method` a character string indicating the type of test performed,

`data.name` a character string giving the name(s) of the data.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

Examples

```
# an independence test
multivariance.test(dep_struct_several_26_100) # conservative
multivariance.test(dep_struct_several_26_100,p.value.type = "resample") #sharp but slow
multivariance.test(dep_struct_several_26_100,p.value.type = "pearson_approx") #

# as an example, all tests for one data set:
coins100 = coins(100)
for (ty in c("total","m.multi.2","m.multi.3","multi"))
 for (pvt in c("distribution_free","resample","pearson_approx"))
  print(multivariance.test(coins100,type=ty,p.value.type = pvt))

# using the vec argument:
x = matrix(rnorm(50*6),ncol = 10) # a 50x6 data matrix
vec = c(1,2,3,4,5,6) # each column is treated as one variable
multivariance.test(x,vec) # is the same as the default

vec = c(1,2,2,1,3,1)
# column 1,4,6 are treated as one variable
# column 2,3 are treated as one variable
# column 5 is treated as one variable
multivariance.test(x,vec)
```

`multivariance.timing` *estimate of the computation time*

Description

Estimates the computation time. This is relative rough. First run with `determine.parameters = TRUE` (which takes a while). Then use the computed parameters to determine the computation time/or sample size.

Usage

```
multivariance.timing(N = NULL, n, sectime = NULL, coef.cdm = 15.2,
  coef.prod = 2.1, coef.sum = 1.05, determine.parameters = FALSE)
```

Arguments

`N`	number of samples. If `NULL` and `sectime` is given, then `N` is computed.
`n`	number of variables
`sectime`	desired computation time in seconds. If `NULL` then the required computation time is computed.
`coef.cdm`	computation time parameter for the centered distance matrices
`coef.prod`	computation time parameter for matrix products
`coef.sum`	computation time parameter for matrix sums
`determine.parameters`	if `TRUE` then the parameters for the current computer are determined. This might take a while (3 loops to N=1000).

Details

When detecting the parameters, the median of the computation times is used.

Examples

```
Ns = (1:100)*10
ns = 1:100
fulltime = outer(Ns,ns,FUN = function(N,n) multivariance.timing(N,n))
contour(Ns,ns,fulltime,xlab = "N",ylab = "n",
 main = "computation time of multivariance in secs",
 sub = "using default parameters -
 use 'determine.parameters = TRUE' to compute machine specific values")

# Run to determine the parameters of your system:
# multivariance.timing(determine.parameters = TRUE)
```

`multivariances.all` *simultaneous computation of multivariance and total/ 2-/ 3-multivariance*

Description

Computes simultaneously multivariance, total multivariance, 2-multivariance and 3-multivariance.

Usage

```
multivariances.all(x, vec = NA, Nscale = TRUE, squared = TRUE, ...)
```

Arguments

`x`	either a data matrix or a list of centered distance matrices
`vec`	if x is a matrix, then this indicates which columns are treated together as one sample; if x is a list, these are the indexes for which the multivariance is calculated. The default is all columns and all indexes, respectively.
`Nscale`	if `TRUE` the multivariance is scaled up by the sample size (and thus it is exactly as required for the test of independence)
`squared`	if `FALSE` it returns the actual multivariance, otherwise the squared multivariance (less computation)
`...`	these are passed to `cdms` (which is only invoked if x is a matrix)

Details

The computation is faster than the separate computations.

Value

Returns a vector with multivariance, total.multivariance, 2-multivariance and 3-multivariance

See Also

`multivariance`, `total.multivariance`, `m.multivariance`

Examples

```
x = coins(100,k = 3)
multivariances.all(x)
# yields the same as:
multivariance(x)
total.multivariance(x)
m.multivariance(x,m=2)
m.multivariance(x,m=3)
```

pearson.pvalue *fast p-value approximation*

Description

Computes the p-value of a sample using Pearson's approximation of Gaussian quadratic forms with the estimators developed by Berschneider and Böttcher in [4].

Usage

```
pearson.pvalue(x, vec = NA, type = "multi", ...)
```

Arguments

x	matrix, the rows should be iid samples
vec	vector, which indicates which columns of x are treated together as one sample. The default case treats each column as a separate sample.
type	one of "multi","total","m.multi.2","m.multi.3","all"
...	these are passed to cdms

Details

This is the method recommended in [4], i.e., using Pearson's quadratic form estimate with the unbiased finite sample estimators for the mean and variance of normalized multivariance together with the unbiased estimator for the limit skewness.

References

For the theoretic background see the reference [4] given on the main help page of this package: multivariance-package.

pearson.qf *approximate distribution of Gaussian quadratic form*

Description

Approximation of the of the value of the distribution function of a Gaussian quadratic form based on its first three moments.

Usage

```
pearson.qf(x, moment, lower.tail = TRUE, verbose = FALSE)
```

Arguments

x	value at which the distribution function is to be evaluated
moment	vector with the mean, variance and skewness of the quadratic form
lower.tail	logical, indicating of the lower or upper tail of the distribution function should be calculated
verbose	logical, if TRUE a warning is issued if negative moments are sanitized to 0.

Details

This is Pearson's approximation for Gaussian quadratic forms as stated in [4] (equation (4.65) in arXiv:1808.07280v2)

References

For the theoretic background see the reference [4] given on the main help page of this package: multivariance-package.

---

rejection.level *rejection level for the test statistic*

Description

Under independence the probability for the normalized and Nscaled (squared) multivariance to be above this level is less than alpha. The same holds for the normalized, Nscaled and Escaled (squared) total multivariance and m-multivariance.

Usage

```
rejection.level(alpha)
```

Arguments

alpha	level of significance

Details

This is based on a distribution-free approach. The value might be very conservative. This is the counterpart to multivariance.pvalue. For a less conservative approach see resample.rejection.level.

The estimate is only valid for alpha smaller than 0.215.

Examples

```
rejection.level(0.05) #the rejection level, for comparison with the following values
total.multivariance(matrix(rnorm(100*3),ncol = 3)) #independent sample
total.multivariance(coins(100)) #dependent sample which is 2-independent

# and the p values are (to compare with alpha)
multivariance.pvalue(total.multivariance(matrix(rnorm(100*3),ncol = 3))) #independent sample
multivariance.pvalue(total.multivariance(coins(100))) #dependent sample which is 2-independent

## Not run:
# visualization of the rejection level
curve(rejection.level(x),xlim = c(0.001,0.215),xlab = "alpha")

## End(Not run)
```

`resample.multivariance`

resampling (total /m-) multivariance

Description

The distribution of the test statistic under the hypothesis of independence is required for the independence tests. This function generates approximate samples of this distribution either by sampling without replacement (permutations) or by sampling with replacement (bootstrap).

Usage

```
resample.multivariance(x, vec = 1:ncol(x), times = 300,
  type = "multi", resample.type = "permutation", ...)
```

Arguments

`x`	matrix, the rows should be iid samples
`vec`	vector, which indicates which columns of `x` are treated together as one sample
`times`	integer, number of samples to generate
`type`	one of `"multi"`,`"total"`,`"m.multi.2"`,`"m.multi.3"`,`"all"`
`resample.type`	one of `"permutation"`, `"bootstrap"`. The samples are generated without replacement (permutations) or with replacement (bootstrap).
`...`	is passed to `cdms`, `multivariance`, `total.multivariance`, `m.multivariance`, respectively.

Details

The resampling is done by sampling from the original data either without replacement (`"permutation"`) or with replacement (`"bootstrap"`). Using resampling without replacement is (much) faster (due to special identities which only hold in this case).

For convenience also the actual (total /m-) multivariance is computed and its p-value.

Value

A list with elements

`resampled` the (total/m-)multivariances of the resampled data,

`original` the (total/m-)multivariance of the original data,

`p.value` the p-value of the original data, computed using the resampled data

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
re.m = resample.multivariance(matrix(rnorm(30*2),nrow = 30),
                        type= "multi",times = 300)$resampled
curve(ecdf(re.m)(x), xlim = c(0,4),main = "empirical distribution of the test statistic under H_0")
```

---

resample.pvalue *p-value via resampling*

Description

Use a resampling method to generate samples of the test statistic under the hypothesis of independence. Based on these the p.value of a given value of a test statistic is computed.

Usage

```
resample.pvalue(value, ...)
```

Arguments

`value`	numeric, the value of (total-/m-)multivariance for which the p-value shall be computed
`...`	passed to `resample.multivariance`. Required is the data matrix `x`.

Details

This function is useful if a p-value of a test statistic shall be computed based on the resampling values of the test statistic of a different sample. For the p-value based on the same sample `resample.multivariance(...)$p.va` is sufficient.

Value

It returns 1 minus the value of the empirical distribution function of the resampling samples evaluated at the given value.

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
x = coins(100)
resample.pvalue(multivariance(x),x=x,times = 300)
resample.pvalue(multivariances.all(x),x=x,times = 300,type = "all")
```

`resample.rejection.level`

rejection level via resampling

Description

Uses the resample method to sample from the test statistic under the hypothesis of independence. The alpha quantile of these samples is returned.

Usage

```
resample.rejection.level(alpha = 0.05, ...)
```

Arguments

`alpha`	numeric, the significance value
`...`	passed to `resample.multivariance`. Required is the data matrix `x`.

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
resample.rejection.level(0.05,matrix(rnorm(30*2),nrow = 30))
resample.rejection.level(0.05,matrix(rnorm(30*3),nrow = 30),vec = c(1,1,2))
```

sample.cdms *resamples centered distance matrices*

Description

resamples centered distance matrices

Usage

```
sample.cdms(list.cdm, replace = FALSE, incl.first = FALSE)
```

Arguments

list.cdm	a list of centered distance matrices
replace	boolean, sampling with or without replacement
incl.first	boolean, if TRUE also the first component is resampled

Value

Returns a list of centered distance matrices, each matrix corresponds to the resampled columns of the corresponding sample, using resampling with replacement (bootstrap) or without replacement (permutations).

sample.cols *resample the columns of a matrix*

Description

resample the columns of a matrix

Usage

```
sample.cols(x, vec = 1:ncol(x), replace = TRUE, incl.first = TRUE)
```

Arguments

x	matrix
vec	vector, indicates which columns belong together
replace	boolean, sampling with or without replacement
incl.first	boolean, if TRUE also the first component is resampled

Value

Returns a matrix with the same dimensions as `x`. The columns are resampled from the original columns. The resampling is done with replacement (`replace = TRUE`) or without (`replace = FALSE`). Columns which belong together (indicated by vec) are resampled identically, i.e., all values in rows of these are kept together.

Examples

```
sample.cols(matrix(1:15,nrow = 5),vec = c(1,1,2))
```

---

tetrahedron *dependence example: tetrahedron sampling*

Description

This function creates samples of a tetrahedron-dice colored r, g, b and rgb. Each sample indicates if for the thrown dice the colors r, g and b are contained on the bottom side of the dice.

Usage

```
tetrahedron(N = 1000)
```

Arguments

`N`	number of samples

Value

It returns the samples of the events r, g and b as rows of a `N` by 3 matrix (the first column corresponds to r, the second to g,...). TRUE indicates that this color is on the bottom side of the dice. The columns are dependent but 2-independent.

References

For the theoretic background see the reference [3] given on the main help page of this package: multivariance-package.

Examples

```
tetrahedron(10)
```

`total.multivariance` *total distance multivariance*

Description

computes the total distance multivariance

Usage

```
total.multivariance(x, vec = NA, lambda = 1, Nscale = TRUE,
  Escale = TRUE, squared = TRUE, ...)
```

Arguments

`x`	either a data matrix or a list of centered distance matrices
`vec`	if x is a matrix, then this indicates which columns are treated together as one sample; if x is a list, these are the indexes for which the multivariance is calculated. The default is all columns and all indexes, respectively.
`lambda`	a scaling parameter >0. Each k-tuple multivariance gets weight `lambda^(n-k)`.
`Nscale`	if `TRUE` the multivariance is scaled up by the sample size (and thus it is exactly as required for the test of independence)
`Escale`	if `TRUE` then it is scaled by the number of multivariances which are theoretically summed up (in the case of independence this yields for normalized distance matrices an estimator with expectation 1)
`squared`	if `FALSE` it returns the actual multivariance, otherwise the squared multivariance (less computation)
`...`	these are passed to `cdms` (which is only invoked if x is a matrix)

Details

Total distance multivariance is per definition the scaled sum of certain distance multivariances, and it characterize dependence.

As a rough guide to interpret the value of total distance multivariance note:

- Large values indicate dependence.
- For `Nscale = TRUE` values close to 1 and smaller indicate independence, larger values indicate dependence. In fact, in the case of independence the test statistic is a Gaussian quadratic form with expectation 1 and samples of it can be generated by `resample.multivariance`.
- For `Nscale = FALSE` small values (close to 0) indicate independence, larger values indicate dependence.

Finally note, that due to numerical (in)precision the value of total multivariance might become negative. In these cases it is set to 0. A warning is issued, if the value is negative and further than the usual (used by `all.equal`) tolerance away from 0.

References

For the theoretic background see the references given on the main help page of this package: multivariance-package.

Examples

```
x = matrix(rnorm(100*3),ncol = 3)
total.multivariance(x) #for an independent sample
# the value coincides with
(multivariance(x[,c(1,2)],Nscale = TRUE) + multivariance(x[,c(1,3)],Nscale = TRUE)+
 multivariance(x[,c(2,3)],Nscale = TRUE) + multivariance(x,Nscale = TRUE))/4

total.multivariance(coins(100)) #value for a dependent sample which is 2-independent
```

Index