

# Package ‘multivariate’

October 6, 2021

**Title** Measuring Multivariate Dependence Using Distance Multivariate

**Version** 2.4.1

**Date** 2021-10-06

**Description** Distance multivariate is a measure of dependence which can be used to detect and quantify dependence of arbitrarily many random vectors. The necessary functions are implemented in this packages and examples are given. It includes: distance multivariate, distance multicorrelation, dependence structure detection, tests of independence and copula versions of distance multivariate based on the Monte Carlo empirical transform. Detailed references are given in the package description, as starting point for the theoretic background we refer to:  
B. Böttcher, Dependence and Dependence Structures: Estimation and Visualization Using the Unifying Concept of Distance Multivariate. Open Statistics, Vol. 1, No. 1 (2020), <[doi:10.1515/stat-2020-0001](https://doi.org/10.1515/stat-2020-0001)>.

**Depends** R (>= 3.3.0)

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**Imports** igraph, graphics, stats, Rcpp, microbenchmark

**RoxygenNote** 7.1.2

**Suggests** testthat

**LinkingTo** Rcpp

**NeedsCompilation** yes

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

**Maintainer** Björn Böttcher <[bjorn.boettcher@tu-dresden.de](mailto:bjorn.boettcher@tu-dresden.de)>

**Repository** CRAN

**Date/Publication** 2021-10-06 15:50:05 UTC

**R topics documented:**

multivariate-package . . . . .	3
anscombe.extended . . . . .	5
cdm . . . . .	6
cdms . . . . .	7
clean.graph . . . . .	8
coins . . . . .	9
copula.multicorrelation . . . . .	10
copula.multicorrelation.test . . . . .	11
copula.multivariate . . . . .	11
dependence.structure . . . . .	12
dep_struct_iterated_13_100 . . . . .	15
dep_struct_ring_15_100 . . . . .	16
dep_struct_several_26_100 . . . . .	17
dep_struct_star_9_100 . . . . .	17
emp.transf . . . . .	18
fastdist . . . . .	18
fastEuclideanCdm . . . . .	19
find.cluster . . . . .	19
independence.test . . . . .	21
layout_on_circles . . . . .	22
m.multivariate . . . . .	23
multicorrelation . . . . .	25
multivariate . . . . .	27
multivariate.pvalue . . . . .	29
multivariate.test . . . . .	29
multivariate.timing . . . . .	31
multivariates.all . . . . .	32
pearson.pvalue . . . . .	33
pearson.qf . . . . .	34
rejection.level . . . . .	35
resample.multivariate . . . . .	36
resample.pvalue . . . . .	37
resample.rejection.level . . . . .	38
sample.cdms . . . . .	38
sample.cols . . . . .	39
tetrahedron . . . . .	40
total.multivariate . . . . .	40

---

multivariate-package *multivariate: Measuring Multivariate Dependence Using Distance Multivariate*

---

## Description

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

## Details

Distance multivariate 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 [1,2,3,4,5,6]. 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, [5] introduces the copula versions of distance multivariate, [6] discusses the quantification of dependence using distance multicorrelations.

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

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

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

## Multivariate

`multivariate` computes the distance multivariate

`total.multivariate` computes the total distance multivariate

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

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

`copula.multivariate` computes the copula versions of the above (introduced in [5])

`multicorrelation` computes the multicorrelations (discussed specifically in [6])

## Functions to use and interpret multivariate

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

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

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

`multivariate.test` provides the corresponding tests of independence. The former provides output as common for tests in R.

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

In [4] various methods to estimate the moments of the test statistic under  $H_0$  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.

`emp.transf` computes the Monte Carlo empirical transform of the data. This data yields the copula version of distance multivariate. Hereto note, that values become randomized due to the "Monte Carlo empirical transform", i.e., the copula versions yield in a finite sample setting not identical values for repeated runs.

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

`anscombe.extended` provides an extension of Anscombe's Quartett. It illustrates that a large value of Pearson's correlation can occur for very different dependencies and that this is not a small-sample problem. These dependencies are at least partly differentiated by values of distance multicorrelation.

## 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*, Vol. 5, No. 3(2018) 353-383. <https://www.vmsta.org/journal/VMSTA/article/127/info>
- [2] B. Böttcher, M. Keller-Ressel, R.L. Schilling, Distance multivariate: New dependence measures for random vectors. *The Annals of Statistics*, Vol. 47, No. 5 (2019) 2757-2789. doi: [10.1214/18AOS1764](https://doi.org/10.1214/18AOS1764)

- [3] B. Böttcher, Dependence and Dependence Structures: Estimation and Visualization using the Unifying Concept of Distance Multivariate. Open Statistics, Vol. 1, No. 1 (2020) 1-46. doi: [10.1515/stat20200001](https://doi.org/10.1515/stat20200001)
- [4] G. Berschneider, B. Böttcher, On complex Gaussian random fields, Gaussian quadratic forms and sample distance multivariate. Preprint. <https://arxiv.org/abs/1808.07280>
- [5] B. Böttcher, Copula versions of distance multivariate and dHSIC via the distributional transform – a general approach to construct invariant dependence measures. Statistics, (2020) 1-18. doi: [10.1080/02331888.2020.1748029](https://doi.org/10.1080/02331888.2020.1748029)
- [6] B. Böttcher, Notes on the interpretation of dependence measures – Pearson’s correlation, distance correlation, distance multicorrelations and their copula versions. Preprint. <https://arxiv.org/abs/2004.07649>

---

anscombe.extended

*Extended Anscombe’s Quartett*

---

## Description

The dataset extends ‘anscombe’ provided in the standard R-package ‘datasets’. All examples feature the same correlation of 0.82, but different types of dependencies. The main aim was to extend the classical examples, which have sample size 11, to larger sample sizes. This illustrates that the implied problems of Pearson’s correlation are not small sample problems! Distance multicorrelation (which coincides in this case with distance correlation) yields different values for the datasets.

## Usage

```
anscombe.extended
```

## Format

list with elements:

- `anscombe.extended$N11` matrix with 11 samples for 5 examples the first 4 are the classical Anscombe Quartett, the fifth is a monoton relation which also features the same correlation.
- `anscombe.extended$N100` same as above but 100 samples
- `anscombe.extended$N1000` same as above but 1000 samples

## Details

Note: Anscombe’s quartett features further identical parameters besides Pearson’s correlation. The extended set is only concerned with correlation.

## References

This example was introduced in the reference [6] given on the main help page of this package: [multivariate-package](#).

## Examples

```
# Code which generates plots of all included data:
op = par(mfrow = c(3,5),mar = c(0.5,0.5,3,0.5))
for (name in c("N11","N100","N1000")) {
  for (i in 1:5) {
    x = anscombe.extended[[name]][,2*i-1]
    y = anscombe.extended[[name]][,2*i]
    plot(x,y,main = paste0("cor = ",round(cor(x,y),2),
"\n Mcor = ",round(multicorrelation(cbind(x,y),type = "pairwise",squared = FALSE),2),
"\n CMcor = ",round(copula.multicorrelation(cbind(x,y),type = "pairwise",squared = FALSE),2)),
      axes = FALSE,xlab = "",ylab = "", cex.main=1)
    # for two variables 'pairwise' coincides with
    # both values of 'total.upper.lower'.
    box()
  }
}
par(op)
```

---

cdm

*computes a doubly centered distance matrix*

---

## Description

computes the doubly centered distance matrix

## Usage

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

## Arguments

<code>x</code>	matrix, each row of the matrix is treated as one sample
<code>normalize</code>	logical, indicates if the matrix should be normalized
<code>psi</code>	if it is NULL, the euclidean distance will be used. In the case of <code>isotropic = TRUE</code> : a real valued negative definite function of one variable (accepting vectors as arguments; returning a vector of the same length). In the case of <code>isotropic = FALSE</code> : a real valued function of two variables (or vectors) to compute the distance of two samples based on a continuous negative definite function.

<code>p</code>	numeric, if it is a value between 1 and 2 then the Minkowski distance with parameter <code>p</code> is used.
<code>isotropic</code>	logical, indicates if <code>psi</code> of the Euclidean distance matrix should be computed, i.e., if an isotropic distance should be used.
<code>external.dm.fun</code>	here one can supply an external function, which computes the distance matrix given <code>x</code> .

## Details

The doubly centered distance matrices are required for the computation of (total / m-) multivariate. If `normalize = TRUE` then the value of multivariate is comparable and meaningful. It can be compared to the [rejection.level](#) or its p-value [multivariate.pvalue](#) can be computed.

More details: If `normalize = TRUE` the matrix is scaled such that the multivariate 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: [multivariate-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 doubly centered distance matrices*

---

## Description

computes the doubly 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	deprecated. Now use vec.
...	these are passed to <a href="#">cdm</a>

**Value**

It returns a list of distance matrices.

---

clean.graph	<i>cleanup dependence structure graph</i>
-------------	---

---

**Description**

Given a dependence structure graph: vertices representing the multivariates 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 <a href="#">dependence.structure</a>
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, type = "even")
```

**Arguments**

N	number of samples
k	each k-tuple will be independent
type	one of "even" or "odd"

**Details**

Throw k independent fair coins. Now consider the k+1 events: The first shows head, the second shows head,... the k-th shows head, there is an even (or odd as selected via type) number of heads. Each row contains the state of these k+1 events.

**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: [multivariate-package](#).

**Examples**

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

---

```
copula.multicorrelation
```

*coupla versions of distance multicorrelation*

---

**Description**

Formally it is nothing but distance multicorrelation applied to the Monte Carlo empirical transform of the data. Hence its values vary for repeated runs.

**Usage**

```
copula.multicorrelation(x, vec = 1:ncol(x), ...)
```

```
CMcor(x, vec = 1:ncol(x), ...)
```

**Arguments**

x	either a data matrix or a list of doubly 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 multivariate is calculated. The default is all columns and all indexes, respectively.
...	are passed to <a href="#">multicorrelation</a>

**References**

For the theoretic background see the reference [5] given on the main help page of this package: [multivariate-package](#).

**See Also**

[multicorrelation](#)

---

copula.multicorrelation.test  
*independence tests using the copula versions of distance multivariate*

---

### Description

Formally it is nothing but tests for distance multivariate applied to the Monte Carlo empirical transform of the data. Hence its values vary for repeated runs.

### Usage

```
copula.multicorrelation.test(x, vec = 1:ncol(x), ...)
```

### Arguments

x	matrix, each row is a sample
vec	vector which indicates which columns are treated as one sample
...	these are passed to <a href="#">cdm</a>

### References

For the theoretic background see the reference [5] given on the main help page of this package: [multivariate-package](#).

---

copula.multivariate *copula version of distance multivariate*

---

### Description

Formally it is nothing but distance multivariate applied to the Monte Carlo empirical transform of the data. Hence its values vary for repeated runs.

### Usage

```
copula.multivariate(x, vec = 1:ncol(x), type = "total", ...)
```

### Arguments

x	either a data matrix or a list of doubly 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 multivariate is calculated. The default is all columns and all indexes, respectively.
type	default: "total.lower.upper", for details and other options see below
...	these are passed to <a href="#">cdms</a> (which is only invoked if x is a matrix)

## References

For the theoretic background see the reference [5] given on the main help page of this package: [multivariate-package](#).

---

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 doubly 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]. 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 multivariates 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:

`multivariates` calculated multivariates,

`cdms` calculated doubly 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: [multivariate-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:

x = coins(100) # 3-dependent
dependence.structure(x)

colnames(x) = c("A", "B", "C")
dependence.structure(x) # names of variables are used as labels

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(x, vec = c(1,1,2)) # names of variables are used as labels

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.

---

emp.transf	<i>Monte Carlo empirical transform</i>
------------	--

---

**Description**

Transforms a matrix (rows: samples, columns: variables) into a matrix of uniform samples with the same dependence structure via the Monte Carlo empirical transform.

**Usage**

```
emp.transf(x, continuous = FALSE)
```

**Arguments**

x	data matrix (rows: samples, columns: variables)
continuous	boolean, if TRUE it provides the classical (non-Monte-Carlo) transformation by the empirical distribution function, which is a reasonable choice for data of continuous distributions.

**References**

For the theoretic background see the reference [5] given on the main help page of this package: [multivariate-package](#).

---

fastdist	<i>fast Euclidean distance matrix</i>
----------	---------------------------------------

---

**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 <code>matrix(x)</code> )
---	---

**Examples**

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

---

fastEuclideanCdm	<i>fast centered Euclidean distance matrix</i>
------------------	--

---

**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 <code>as.matrix(x)</code> )
normalize	boolean. If TRUE the matrix will be normalized to mean 1.

---

find.cluster	<i>cluster detection</i>
--------------	--------------------------

---

**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.multivariiances = 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

<code>x</code>	matrix with the samples
<code>vec</code>	vector, it indicates which columns are initially treated together as one sample
<code>list.cdm</code>	list of doubly centered distance matrices
<code>mem</code>	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 <code>i</code> belongs to cluster <code>mem[i]</code>
<code>cluster.to.vertex</code>	vector, contains the cluster to vertex relations, i.e., <code>cluster.to.vertex[i]</code> is the index of the vertex which represents cluster <code>i</code>
<code>vertex.to.cdm</code>	vector, contains the vertex to doubly centered distance matrix relations, i.e., <code>vertex.to.cdm[i]</code> is the index of the doubly centered distance matrix in <code>list.cdm</code> which corresponds to vertex <code>i</code>
<code>previous.n.o.cdms</code>	vector, number of the doubly centered distance matrices in the previous iteration (it is used to ensure that previously check tuples are not checked again)
<code>all.multivariiances</code>	vector, which contains all distance multivariiances which have been calculated so far. Only used to finally return all distance multivariiances which have been calculated.
<code>g</code>	dependence structure graph
<code>fixed.rejection.level</code>	vector, if not NA the <code>fixed.rejection.level[k]</code> is used for the <code>k</code> -tuples, instead of a level derived from the significance level <code>alpha</code>
<code>alpha</code>	numeric, significance level used for the (distribution-free) tests
<code>p.adjust.method</code>	name of the method used to adjust the <code>p</code> -values for multiple testing, see <a href="#">p.adjust</a> for all possible options.
<code>verbose</code>	boolean, if TRUE details during the detection are printed and whenever a cluster is newly detected the (so far) detected dependence structure is plotted.
<code>kvec</code>	vector, <code>k</code> -tuples are only checked for each <code>k</code> in <code>kvec</code> , i.e., for <code>kvec = 2:4</code> only 2,3 and 4-tuples would be check and then the algorithm stops.
<code>parameter.range</code>	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 <a href="#">resample.multivariate</a> in the case of 'type = resample'

## Details

For further details see [dependence.structure](#).

---

independence.test	<i>test for independence</i>
-------------------	------------------------------

---

## Description

Deprecated. Use [multivariate.test](#) instead. It provides all options and returns test result in a standard R format.

## 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 doubly 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 multivariate 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 <a href="#">cdms</a> (which is only invoked if x is a matrix)

**Details**

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

---

**Description**

Computes m distance multivariate.

**Usage**

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

**Arguments**

x	either a data matrix or a list of doubly 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 multivariate is calculated. The default is all columns and all indexes, respectively.
m	=2 or 3 the m-multivariate will be computed.
Nscale	if TRUE the multivariate 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 multivariates 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 multivariate, otherwise the squared multivariate (less computation)
...	these are passed to <code>cdms</code> (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 characterizes 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 and discussed in [3,4,6].

### Usage

```
multicorrelation(
  x,
  vec = 1:ncol(x),
  type = "total.upper.lower",
  multicorrelation.type = "normalized",
  estimator.type = "bias.corrected",
  squared = TRUE,
  ...
)
```

```
Mcor(
  x,
  vec = 1:ncol(x),
  type = "total.upper.lower",
  multicorrelation.type = "normalized",
  estimator.type = "bias.corrected",
  squared = TRUE,
  ...
)
```

### Arguments

x	either a data matrix or a list of doubly 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 multivariate is calculated. The default is all columns and all indexes, respectively.
type	default: "total.lower.upper", for details and other options see below
multicorrelation.type	one of "normalized", "unnormalized"
estimator.type	one of "biased", "bias.corrected"
squared	if FALSE it returns the actual multivariate, otherwise the squared multivariate (less computation)
...	these are passed to <code>cdms</code> (which is only invoked if x is a matrix)

## Details

There exist many variants of distance multicorrelation as discussed in [6] – and only in specific cases a direct comparison of the values is meaningful.

The implemented options are:

- `total.upper.lower` normalized bias.corrected: default; bounded by 1; fast; population limit characterizes independence by 0
- `pairwise` normalized bias.corrected: bounded by 1; fast; population limit characterizes pairwise independence by 0
- `total.upper` normalized biased: biased versions of the above
- `total.lower` normalized biased
- `pairwise` normalized biased
- `multi` normalized biased: population limit characterizes only in case of lower independence the independence of all variables by 0
- `m.multi.3` normalized biased: population limit characterizes only in case of pairwise independence the 3-independence of all variables by 0
- `pairwise` unnormalized biased population limit characterizes pairwise independence by 0 and relation by similarity transforms by 1
- `multi` unnormalized biased: population limit characterizes only in case of lower independence the independence of all variables by 0 and relation by similarity transforms by 1
- `m.multi.3` unnormalized biased: population limit characterizes only in case of pairwise independence the 3-independence of all variables by 0 and relation by similarity transforms by 1

Further details:

The "bias.corrected" versions require a data matrix, since they compute bias corrected centered distance matrices.

For "multi" the unnormalized and normalized version coincide if an even number of variables is considered. They usually differ if an odd number of variables is considered. If all variables are related by similarity transforms the unnormalized "unnormalized" multicorrelations are 1.

For "pairwise" an alias is "m.multi.2".

For total multicorrelation there is currently only a feasible empirical estimator for a lower or upper bound. These are upper and lower bounds for in the population setting. When using bias corrected estimators these are in general no proper bounds, but their range can be used as values for comparisons.

## Value

Value of the multicorrelation(s).

## References

For the theoretic background see the references [2,3,6] given on the main help page of this package: [multivariate-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("total.lower","total.upper","pairwise","m.multi.3","multi"))
  for (mty in c("normalized"))
    print(paste(format(multicorrelation(
      x,type=ty,multicorrelation.type = mty,estimator.type = "biased")
      ,digits=3,nsmall = 3,width = 7),mty,ty,"correlation - biased estimate"))

for (ty in c("total.upper.lower","pairwise"))
  for (mty in c("normalized"))
    print(paste(format(multicorrelation(
      x,type=ty,multicorrelation.type = mty,estimator.type = "bias.corrected")
      ,digits=3,nsmall = 3,width = 7),mty,ty,"correlation - bias corrected estimate"))

for (ty in c("m.multi.2","m.multi.3","multi"))
  for (mty in c("unnormalized"))
    print(paste(format(multicorrelation(
      x,type=ty,multicorrelation.type = mty,estimator.type = "biased")
      ,digits=3,nsmall = 3,width = 7),mty,ty,"correlation - biased estimate"))

```

---

multivariate

*distance multivariate*


---

**Description**

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

**Usage**

```

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

```

**Arguments**

**x** either a data matrix or a list of doubly 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 multivariate is calculated. The default is all columns and all indexes, respectively.

Nscale	if TRUE the multivariate is scaled up by the sample size (and thus it is exactly as required for the test of independence)
correlation	deprecated, please use the function <a href="#">multicorrelation</a> instead.
squared	if FALSE it returns the actual multivariate, otherwise the squared multivariate (less computation)
...	these are passed to <a href="#">cdms</a> (which is only invoked if x is a matrix)

### Details

If x is a 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 multivariate 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 multivariate, for a multivariate 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 multivariate 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.multivariate](#).
- 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.multivariate](#).
- 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 multivariate 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: [multivariate-package](#).

### Examples

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

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

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

---

```
multivariate.pvalue transform multivariate to p-value
```

---

### Description

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

### Usage

```
multivariate.pvalue(x)
```

### Arguments

x value of a normalized [multivariate](#) 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: [multivariate-package](#).

---

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

---

### Description

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

**Usage**

```

multivariate.test(
  x,
  vec = 1:ncol(x),
  type = "total",
  p.value.type = "pearson_approx",
  verbose = TRUE,
  ...
)

```

**Arguments**

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

**Details**

For the use of `vec` see the examples below and the more detailed explanation of this argument for [multivariate](#).

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". Finally, "pearson\_unif" uses fixed parameters in Pearson's estimate, it is only applicable for univariate uniformly distributed marginals

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: [multivariate-package](#).

## Examples

```
# an independence test
multivariate.test(dep_struct_several_26_100,p.value.type = "distribution_free") # conservative
multivariate.test(dep_struct_several_26_100,p.value.type = "resample") #sharp but slow
multivariate.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(multivariate.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
multivariate.test(x,vec,p.value.type = "distribution_free") # 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
multivariate.test(x,vec,p.value.type = "distribution_free")
```

---

multivariate.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

```
multivariate.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 doubly 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 multivariate and total/ 2-/ 3-  
multivariate*

---

**Description**

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

**Usage**

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



**Arguments**

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

**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 multivariate 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: [multivariate-package](#).

---

pearson.qf

*approximate distribution function of a Gaussian quadratic form*

---

**Description**

Approximation 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: [multivariate-package](#).

---

rejection.level	<i>rejection level for the test statistic</i>
-----------------	---

---

## 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.multivariate
      resampling (total /m-) multivariate
```

---

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

### Arguments

<code>x</code>	matrix, the rows should be iid samples
<code>vec</code>	vector, which indicates which columns of <code>x</code> are treated together as one sample
<code>times</code>	integer, number of samples to generate
<code>type</code>	one of "multi", "total", "m.multi.2", "m.multi.3", "all"
<code>resample.type</code>	one of "permutation", "bootstrap". The samples are generated without replacement (permutations) or with replacement (bootstrap).
<code>...</code>	is passed to <code>cdms</code> , <code>multivariate</code> , <code>total.multivariate</code> , <code>m.multivariate</code> , 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-) multivariate is computed and its p-value.

### Value

A list with elements

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

`original` the (total/m-)multivariate 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: [multivariate-package](#).

**Examples**

```
re.m = resample.multivariate(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	<i>p-value via resampling</i>
-----------------	-------------------------------

---

**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-)multivariate for which the p-value shall be computed
...	passed to <a href="#">resample.multivariate</a> . 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.multivariate\(...\)\\$p.v](#) 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: [multivariate-package](#).

**Examples**

```
x = coins(100)
resample.pvalue(multivariance(x),x=x,times = 300)
resample.pvalue(multivariiances.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 doubly centered distance matrices*

---

**Description**

resamples doubly centered distance matrices

**Usage**

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

**Arguments**

list.cdm	a list of doubly 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 doubly 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	<i>resample the columns of a matrix</i>
-------------	---

---

**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: [multivariate-package](#).

### Examples

```
tetrahedron(10)
```

---

total.multivariate      *total distance multivariate*

---

### Description

computes the total distance multivariate



**Usage**

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

**Arguments**

x	either a data matrix or a list of doubly 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 multivariate is calculated. The default is all columns and all indexes, respectively.
lambda	a scaling parameter >0. Each k-tuple multivariate gets weight $\lambda^{(n-k)}$ .
Nscale	if TRUE the multivariate 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 multivariates 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 multivariate, otherwise the squared multivariate (less computation)
...	these are passed to <a href="#">cdms</a> (which is only invoked if x is a matrix)

**Details**

Total distance multivariate is per definition the scaled sum of certain distance multivariates, and it characterizes dependence.

As a rough guide to interpret the value of total distance multivariate 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.multivariate](#).
- 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 multivariate 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: [multivariate-package](#).

**Examples**

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

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

# Index

- \* **datasets**
  - anscombe.extended, 5
  - dep\_struct\_iterated\_13\_100, 15
  - dep\_struct\_ring\_15\_100, 16
  - dep\_struct\_several\_26\_100, 17
  - dep\_struct\_star\_9\_100, 17
- all.equal, 24, 28, 41
- anscombe.extended, 4, 5
- cdm, 4, 6, 8, 11, 30
- cdms, 4, 7, 11, 21–23, 25, 28, 33, 34, 36, 41
- clean.graph, 8, 13
- CMcor (copula.multicorrelation), 10
- coins, 4, 9
- copula.multicorrelation, 10
- copula.multicorrelation.test, 11
- copula.multivariance, 3, 11
- dep\_struct\_iterated\_13\_100, 4, 15
- dep\_struct\_ring\_15\_100, 4, 16
- dep\_struct\_several\_26\_100, 4, 17
- dep\_struct\_star\_9\_100, 4, 17
- dependence.structure, 3, 4, 8, 12, 15–17, 19, 21
- emp.transf, 4, 18
- fastdist, 18
- fastEuclideanCdm, 19
- find.cluster, 4, 13, 19
- independence.test, 21
- layout\_on\_circles, 22
- m.multivariance, 3, 23, 33, 36
- Mcor (multicorrelation), 25
- multicorrelation, 3, 10, 25, 28
- multivariance, 3, 27, 29, 30, 33, 36
- multivariance-package, 3, 5, 7, 10–12, 14, 18, 22, 24, 26, 28, 29, 31, 34, 37, 38, 40, 41
- multivariance.pvalue, 3, 7, 29, 35
- multivariance.test, 3, 4, 21, 29
- multivariance.timing, 4, 31
- multivariations.all, 3, 32
- p.adjust, 20
- p.adjust.methods, 13
- pearson.pvalue, 4, 29, 33
- pearson.qf, 34
- rejection.level, 3, 7, 29, 35
- resample.multivariance, 4, 21, 24, 28, 36, 37, 38, 41
- resample.pvalue, 4, 29, 37
- resample.rejection.level, 4, 35, 38
- sample.cdms, 38
- sample.cols, 39
- tetrahedron, 4, 40
- total.multivariance, 3, 28, 33, 36, 40