

Package ‘copula’

October 8, 2009

Version 0.8-12

Date 2009-10-07

Title Multivariate dependence with copulas

Author Jun Yan <jyan@stat.uconn.edu> and Ivan Kojadinovic <ivan@stat.auckland.ac.nz>.

Maintainer Jun Yan <jyan@stat.uconn.edu>

Depends methods, mvtnorm, scatterplot3d, sn, pspline

Enhances norlmix

Description Classes (S4) of commonly used copulas including elliptical (normal and t), Archimedean (Clayton, Gumbel, Frank, and Ali-Mikhail-Haq), extreme value (Gumbel, Husler-Reiss, Galambos, and t-EV), and other families (Plackett and Farlie-Gumbel-Morgenstern). Methods for density, distribution, random number generation, bivariate dependence measures, perspective and contour plots. Functions for fitting copula models with variance estimate. Independence tests among random variables and random vectors. Serial independence tests for univariate and multivariate continuous time series. Goodness-of-fit tests for copulas based on multipliers and on the parametric bootstrap.

License GPL (>= 3)

Collate Classes.R Copula.R derCdfPdf.R E.R amhCopula.R amhExpr.R archmCopula.R asymCopula.R claytonCopula.R claytonExpr.R debye.R ellipCopula.R evCopula.R evTests.R exchTests.R fgmCopula.R fitCopula.R fitMvdc.R frankCopula.R frankExpr.R galambosCopula.R galambosExpr-math.R galambosExpr.R gofEVTTests.R gofTests.R graphics.R gumbelCopula.R gumbelExpr.R huslerReissCopula.R huslerReissExpr.R indepCopula.R indepTests.R logseries.R mvdc.R normalCopula.R plackettCopula.R plackettExpr.R schlatherCopula.R skewNormalCopula.R stable.R tCopula.R tawnCopula.R tawnExpr.R tevCopula.R zzz.R

Repository CRAN

Date/Publication 2009-10-08 12:41:09

R topics documented:

archmCopula	2
archmCopula-class	4
AssocMeasures	5
contour-methods	6
Copula	7
copula-class	9
ellipCopula	10
ellipCopula-class	11
evCopula	12
evCopula-class	13
evTest	14
fgmCopula	15
fgmCopula-class	16
fitCopula	17
fitCopula-class	20
fitMvdc	21
generator	23
gofCopula	23
indepCopula	25
indepCopula-class	26
indepTest	27
loss	29
multIndepTest	30
multSerialIndepTest	32
Mvdc	34
mvdc-class	35
persp-methods	36
plackettCopula	37
rdj	37
serialIndepTest	38
show-methods	40
summary-methods	41
uranium	41
Index	42

archmCopula

Construction of Archimedean copula class object

Description

Constructs an Archimedean copula class object with its corresponding parameter and dimension.

Usage

```
archmCopula(family, param, dim = 2, ...)  
claytonCopula(param, dim = 2)  
frankCopula(param, dim = 2)  
gumbelCopula(param, dim = 2)  
amhCopula(param, dim = 2)
```

Arguments

family	a character string specifying the family of an Archimedean copula. Implemented families are "clayton", "frank", and "gumbel".
param	a numeric vector specifying the parameter values.
dim	the dimension of the copula.
...	currently nothing.

Details

"archmCopula" is a wrapper for "claytonCopula", "frankCopula", "gumbelCopula" and "amhCopula".

Only the bivariate Ali-Mikhail-Haq copula family ("amhCopula") is available.

The maximum dimension for which the expression of the pdf is available is 6 for the Clayton, Gumbel and Frank families. The cdf expression is always available.

The maximum dimension for which "dcopula" can be evaluated is 10 for the Clayton and Gumbel families, and 6 for the Frank family. They are also the maximum dimensions for which maximum likelihood estimation can be done.

Value

An Archimedean copula object of class "claytonCopula", "frankCopula", "gumbelCopula", or "amhCopula".

References

R.B. Nelsen (2006), *An introduction to Copulas*, Springer, New York.

See Also

[ellipCopula](#), [evCopula](#).

Examples

```
clayton.cop <- claytonCopula(2, dim = 3)  
## scatterplot3d(rcopula(clayton.cop, 1000))  
  
frank.cop <- frankCopula(3)  
persp(frank.cop, dcopula)  
  
gumbel.cop <- archmCopula("gumbel", 5)
```

```

contour(gumbel.cop, dcopula)

amh.cop <- amhCopula(0.5)

## A 7-dim Frank copula
frank.cop <- frankCopula(3, dim = 7)
x <- rcopula(frank.cop, 5)
## dcopula doesn't work
## Not run:
dcopula(frank.cop, x)
## End(Not run)

## A 7-dim Gumbel copula
gumbel.cop <- gumbelCopula(2, dim = 7)
## exprdist$pdf is not available
gumbel.cop@exprdist
## but can still be evaluated
dcopula(gumbel.cop, x)

```

archmCopula-class *Class "archmCopula"*

Description

Archimedean copula class.

Objects from the Class

Objects can be created by calls of the form `new("archmCopula", ...)` or by function `"archmCopula"`. Implemented families are Clayton, Gumbel, Frank, and Ali-Mikhail-Haq.

Slots

exprdist: Object of class `"expression"`: expressions of the cdf and pdf of the copula. These expressions are used in function `'pcopula'` and `'dcopula'`.

dimension: Object of class `"numeric"`, dimension of the copula.

parameters: Object of class `"numeric"`, parameter values.

param.names: Object of class `"character"`, parameter names.

param.lowbnd: Object of class `"numeric"`, parameter lower bounds.

param.upbnd: Object of class `"numeric"`, parameter upper bounds.

message: Object of class `"character"`, family names of the copula.

Methods

dcopula signature(copula = "claytonCopula"): ...
pcopula signature(copula = "claytonCopula"): ...
rcopula signature(copula = "claytonCopula"): ...
dcopula signature(copula = "frankCopula"): ...
pcopula signature(copula = "frankCopula"): ...
rcopula signature(copula = "frankCopula"): ...
dcopula signature(copula = "gumbelCopula"): ...
pcopula signature(copula = "gumbelCopula"): ...
rcopula signature(copula = "gumbelCopula"): ...
dcopula signature(copula = "amhCopula"): ...
pcopula signature(copula = "amhCopula"): ...
rcopula signature(copula = "amhCopula"): ...

Extends

Class "archmCopula" extends class "copula" directly. Class "claytonCopula", "frankCopula", "gumbelCopula" and "amhCopula" extends class "archmCopula" directly.

Note

"gumbelCopula" is also of class "evCopula".

See Also

[archmCopula](#), [copula-class](#).

Description

These functions compute Kendall's Tau, Spearman's Rho, and the tail dependence index for bivariate copulas. Calibration is the inverse function: it calibrates the copula parameter given the value of Kendall's Tau or Spearman's Rho.

Usage

```

kendallsTau(copula, ...)
spearmanRho(copula, ...)
tailIndex(copula, ...)
calibKendallsTau(copula, tau)
calibSpearmanRho(copula, rho)
  
```

Arguments

copula	a "copula" object.
tau	a numerical value of Kendall's Tau in [-1, 1].
rho	a numerical value of Spearman's Rho in [-1, 1].
...	currently nothing.

Details

The calibration function in fact returns a moment estimate of the parameter for one-parameter copulas.

When there are no closed-form expressions for Kendall's tau or Spearman's rho, the calibration functions use numerical approximation techniques (see the last reference). For closed-form expressions, see Frees and Valdez (1998). For the t copula, the calibration function based on Spearman's rho uses the corresponding expression for the normal copula as an approximation.

References

E.W. Frees and E.A. Valdez (1998). Understanding relationships using copulas. *North American Actuarial Journal*, 2:1–25.

I. Kojadinovic and J. Yan (2008), Comparison of three semiparametric methods for estimating dependence parameters in copula models, submitted.

Examples

```
gumbel.cop <- gumbelCopula(3)
kendallsTau(gumbel.cop)
spearmansRho(gumbel.cop)
tailIndex(gumbel.cop)

## let us compute the sample versions
x <- rcopula(gumbel.cop, 200)
cor(x, method = "kendall")
cor(x, method = "spearman")
## compare with the true parameter value 3
calibKendallsTau(gumbel.cop, cor(x, method="kendall")[1,2])
calibSpearmanRho(gumbel.cop, cor(x, method="spearman")[1,2])
```

contour-methods *Methods for function 'contour' in package 'copula'*

Description

Methods for function `contour` in package **copula**

Details

When `x` is of class "copula", the following arguments can be supplied:

fun the function to be plotted, i.e., "dcopula" or "pcopula".

n (= 51) the number of points to do the plotting.

arguments for "contour" theta = -30, phi = 30, expand = 0.618.

When `x` is of class "mvdc", the following arguments replace the effect of `n = 51`:

xlim the range of the x variable.

ylim the range of the y variable.

nx the number of points for x to expand.

ny the number of points for y to expand.

Methods

x = "copula" Contour plot for a "copula" or "indepCopula" object.

x = "mvdc" Contour plot for a "mvdc" object.

Examples

```
contour(frankCopula(-0.8), dcopula)
contour(claytonCopula(2), pcopula)
x <- mvdc(gumbelCopula(3), c("norm", "norm"),
          list(list(mean = 0, sd = 1), list(mean = 1)))
contour(x, dmvc, xlim=c(-2, 2), ylim=c(-1, 3))
contour(x, pmvc, xlim=c(-2, 2), ylim=c(-1, 3))
```

Copula

Copula distribution functions

Description

Density, distribution function, and random generation for a copula object.

Usage

```
dcopula(copula, u)
pcopula(copula, u)
rcopula(copula, n)
```

Arguments

<code>copula</code>	a "copula" object.
<code>u</code>	a vector of the copula dimension or a matrix with number of rows being the copula dimension, giving the coordinates of the points where the density or distribution function needs to be evaluated.
<code>n</code>	number of observations to be generated.

Details

The density function of an Archimedean copula was obtained by differentiating the distribution function symbolically using `Mathematica` and then processed by `deriv` to give algorithmic expressions. The maximum dimension implemented is 10 for Clayton and Gumbel, and 6 for Frank.

The distribution function of a `t` copula uses `pmvt` from package `mvtnorm`. The density function of a `t` copula uses the `dmst` from package `sn`.

The random number generator for an Archimedean copula uses the conditional approach for the bivariate case and the Marshal-Olkin (1988) approach for dimension greater than 2.

Value

"`dcopula`" gives the density, "`pcopula`" gives the distribution function, and "`rcopula`" generates random variates.

References

E.W. Frees and E.A. Valdez (1998). Understanding relationships using copulas. *North American Actuarial Journal*, 2:1–25.

C. Genest and A.-C. Favre (2007). Everything you always wanted to know about copula modeling but were afraid to ask. *Journal of Hydrological Engineering*, 12:347–368.

H. Joe (1997). *Multivariate Models and Dependence Concepts*. Chapman and Hall, London.

A.W. Marshal and I. Olkin (1988). Families of multivariate distributions. *Journal of the American Statistical Association*, 83:834–841.

R.B. Nelsen (2006). *An introduction to Copulas*. Springer, New York.

See Also

[copula-class](#), [ellipCopula](#), [archmCopula](#), [fgmCopula](#).

Examples

```
norm.cop <- normalCopula(0.5)
norm.cop
x <- rcopula(norm.cop, 100)
plot(x)
dcopula(norm.cop, x)
pcopula(norm.cop, x)
persp(norm.cop, dcopula)
contour(norm.cop, pcopula)
## a 3-dimensional normal copula
u <- rcopula(normalCopula(0.5, dim = 3), 1000)
## scatterplot3d(u)
## a 3-dimensional clayton copula
v <- rcopula(claytonCopula(2, dim = 3), 1000)
## scatterplot3d(v)
```

copula-class	Class "copula"
--------------	----------------

Description

A class representing multivariate distributions with uniform margins.

Objects from the Class

Objects can be created by calls of the form `new("copula", ...)`.

Slots

dimension: Object of class "numeric", dimension of the copula.

parameters: Object of class "numeric", parameter values.

param.names: Object of class "character", parameter names.

param.lowbnd: Object of class "numeric", parameter lower bounds.

param.upbnd: Object of class "numeric", parameter upper bounds.

message: Object of class "character", family names of the copula.

Warning

This implementation is still at the experimental stage and is subject to change during the development.

Note

The "copula" class is extended by the "evCopula", "archmCopula", and "ellipCopula" classes. Instances of implemented copulas can be created from functions "evCopula", "archmCopula" and "ellipcopula".

"plackettCopula" and "fgmCopula" are special types of copulas which do not belong to either one of the three classes above.

See Also

[archmCopula-class](#), [ellipCopula-class](#), [evCopula-class](#), [fgmCopula-class](#).

 ellipCopula

Construction of elliptical copula class object

Description

Constructs an elliptical copula class object with its corresponding parameters and dimension.

Usage

```
ellipCopula(family, param, dim = 2, dispstr = "ex", df = 4, ...)
normalCopula(param, dim = 2, dispstr = "ex")
tCopula(param, dim = 2, dispstr = "ex", df = 4, df.fixed = FALSE)
```

Arguments

family	a character string specifying the family of an elliptical copula. Implemented families are "normal" and "t".
param	a numeric vector specifying the parameter values.
dim	the dimension of the copula.
dispstr	a character string specifying the type of the symmetric positive definite matrix characterizing the elliptical copula. Implemented structures are "ex" for exchangeable, "ar1" for AR(1), "toep" for Toeplitz, and "un" for unstructured.
df	a numerical value specifying the number of degrees of freedom of the multivariate t distribution used to construct the t copulas.
df.fixed	TRUE means that the degrees of freedom will never be considered as a parameter to be estimated; FALSE means that df will be estimated if the object is passed as argument to <code>fitCopula</code> .
...	currently nothing.

Value

An elliptical copula object of class "normalCopula" or "tCopula".

Note

"ellipCopula" is a wrapper for "normalCopula" and "tCopula".

See Also

[archmCopula](#), [fitCopula](#).

Examples

```

norm.cop <- normalCopula(c(0.5, 0.6, 0.7), dim = 3, dispstr = "un")
t.cop <- tCopula(c(0.5, 0.3), dim = 3, dispstr = "toep",
               df = 2, df.fixed = TRUE)
## from the wrapper
norm.cop <- ellipCopula("normal", param = c(0.5, 0.6, 0.7),
                       dim = 3, dispstr = "un")
## 3d scatter plot of 1000 random observations
## scatterplot3d(rcopula(norm.cop, 1000))
## scatterplot3d(rcopula(t.cop, 1000))

```

ellipCopula-class *Class "ellipCopula"*

Description

Copulas generated from elliptical multivariate distributions.

Objects from the Class

Objects can be created by calls of the form `new("ellipCopula", ...)`, or by function `"ellipCopula"`.

Slots

dispstr: Object of class "character", indicating how the dispersion matrix is parameterized; can 'ex', 'ar1', 'toep', or 'un'.

dimension: Object of class "numeric", dimension of the copula.

parameters: Object of class "numeric", parameter value.

param.names: Object of class "character", parameter names.

param.lowbnd: Object of class "numeric", parameter lower bound.

param.upbnd: Object of class "numeric", parameter upper bound.

message: Object of class "character", family names of the copula.

Extends

Class "ellipCopula" extends class "copula" directly. Class "normalCopula" and "tCopula" extends class "ellipCopula" directly.

Methods

dcopula signature(copula = "normalCopula"): ...

pcopula signature(copula = "normalCopula"): ...

rcopula signature(copula = "normalCopula"): ...

dcopula signature(copula = "tCopula"): ...

pcopula signature(copula = "tCopula"): ...

rcopula signature(copula = "tCopula"): ...

See Also

[ellipCopula](#), [copula-class](#).

 evCopula

Construction of extreme value copula class objects

Description

Constructs an extreme value copula class object with its corresponding parameter.

Usage

```
evCopula(family, param, dim = 2, ...)
galambosCopula(param)
huslerReissCopula(param)
```

Arguments

family	a character string specifying the family of an extreme value copula. Implemented families are "galambos" and "gumbel".
param	a numeric vector specifying the parameter values.
dim	the dimension of the copula.
...	currently nothing.

Value

An object of class "galambosCopula", "gumbelCopula", or "huslerReissCopula".

Note

The Gumbel copula is both an Archimedean and an extreme value copula.

See Also

[ellipCopula](#), [archmCopula](#).

Examples

```
gumbel.cop <- evCopula("gumbel", param=2)
contour(gumbel.cop, dcopula)
galambos.cop <- galambosCopula(2)
contour(galambos.cop, dcopula)
```

evCopula-class *Class "evCopula"*

Description

Class representing extreme value copulas.

Objects from the Class

Objects can be created by calls of the form `new("evCopula", ...)` or by function `"evCopula"`.

Slots

dimension: Object of class "numeric", the dimension of the copula.

parameters: Object of class "numeric", parameter values.

param.names: Object of class "character", parameter names.

param.lowbnd: Object of class "numeric", parameter lower bound.

param.upbnd: Object of class "numeric", parameter upper bound.

message: Object of class "character", family names of the copula.

Methods

dcopula signature(copula = "galambosCopula"): ...

pcopula signature(copula = "galambosCopula"): ...

rcopula signature(copula = "galambosCopula"): ...

dcopula signature(copula = "huslerReissCopula"): ...

pcopula signature(copula = "huslerReissCopula"): ...

rcopula signature(copula = "huslerReissCopula"): ...

Extends

Class "evCopula" extends class "copula" directly. Both class "galambosCopula" and class "huslerReissCopula" extend class "evCopula" directly.

Note

The expressions of pdf are obtained by differentiating the cdf expression using function `"deriv"`.
"gumbelCopula" is also of class "archmCopula".

See Also

[evCopula](#), [copula-class](#).

`evTest`*Large-sample test of extreme-value dependence*

Description

Test of extreme-value dependence based on the empirical copula and max-stability. The test statistics are defined in Kojadinovic and Yan (2009). Approximate p-values for the test statistics are obtained by means of a fast *multiplier* technique.

Usage

```
evTest(x, N = 1000)
```

Arguments

<code>x</code>	a data matrix that will be transformed to pseudo-observations.
<code>N</code>	number of multiplier iterations to be used to simulate realizations of the test statistic under the null hypothesis.

Details

More details are available in Kojadinovic and Yan (2009). See also Remillard and Scaillet (2009).

Value

Returns a list whose attributes are:

<code>statistic</code>	value of the test statistic.
<code>pvalue</code>	corresponding approximate p-value.

References

B. Remillard and O. Scaillet (2009). Testing for equality between two copulas. *Journal of Multivariate Analysis*, 100(3), pages 377-386.

I. Kojadinovic and J. Yan (2009). Large-sample tests of extreme-value dependence for multivariate copulas. Submitted.

See Also

[evCopula](#).

Examples

```
## Do the data come from an extreme-value copula?
evTest(rcopula(gumbelCopula(3), 200))
evTest(rcopula(claytonCopula(3), 200))

## A three-dimensional example
evTest(rcopula(gumbelCopula(3, dim=3), 200))
evTest(rcopula(claytonCopula(3, dim=3), 200))
```

`fgmCopula`*Construction of a fgmCopula class object*

Description

Constructs a multivariate multiparameter Farlie-Gumbel-Morgenstern copula class object with its corresponding parameters and dimension.

Usage

```
fgmCopula(param, dim = 2)
```

Arguments

<code>param</code>	a numeric vector specifying the parameter values.
<code>dim</code>	the dimension of the copula.
<code>...</code>	currently nothing.

Value

A Farlie-Gumbel-Morgenstern copula object of class "fgmCopula".

Note

The verification of the validity of the parameter values is of high complexity and may not work for high dimensional copulas.

The random number generation needs to be properly tested, especially for dimensions higher than 2.

References

R.B. Nelsen (2006), *An introduction to Copulas*, Springer, New York.

See Also

[Copula](#), [copula-class](#), [fitCopula](#).

Examples

```
## a bivariate example
fgm.cop <- fgmCopula(1)
x <- rcopula(fgm.cop, 1000)
cor(x, method = "kendall")
kendallsTau(fgm.cop)
cor(x, method = "spearman")
spearmanRho(fgm.cop)
persp(fgm.cop, dcopula)
contour(fgm.cop, dcopula)

## a trivariate example with wrong parameter values
## fgm2.cop <- fgmCopula(c(1,1,1,1), dim = 3)

## a trivariate example with satisfactory parameter values
fgm2.cop <- fgmCopula(c(.2,-.2,-.4,.6), dim = 3)
fgm2.cop
```

fgmCopula-class *Class "fgmCopula"*

Description

Multivariate multiparameter Farlie-Gumbel-Morgenstern copula.

Objects from the Class

Objects can be created by calls of the form `new("fgmCopula", ...)`.

Slots

exprdist: Object of class "expression", expressions for the cdf and pdf of the copula.

These expressions are used in function "pcopula" and "dcopula".

dimension: Object of class "numeric", the dimension of the copula.

parameters: Object of class "numeric", parameter values.

param.names: Object of class "character", parameter names.

param.lowbnd: Object of class "numeric", parameter lower bound.

param.upbnd: Object of class "numeric", parameter upper bound.

message: Object of class "character", family names of the copula.

Methods

dcopula signature(copula = "fgmCopula"): ...

pcopula signature(copula = "fgmCopula"): ...

rcopula signature(copula = "fgmCopula"): ...

Extends

Class "fgmCopula" extends class "copula" directly.

Note

The verification of the validity of the parameter values is of high complexity and may not work for high dimensional copulas.

The random number generation needs to be properly tested, especially for dimensions higher than 2.

References

R.B. Nelsen (2006), *An introduction to Copulas*, Springer, New York.

See Also

[copula-class](#), [fgmCopula-class](#).

fitCopula

Estimation of the dependence parameters in copula models

Description

Fits a copula model to multivariate data belonging to the unit hypercube. The data can be pseudo-observations constructed from empirical or parametric marginal c.d.f.s, or from true observations from the copula.

Usage

```
loglikCopula(param, x, copula, suppressMessages=FALSE)
fitCopula(copula, data, method="mpl", start=NULL, lower=NULL, upper=NULL,
          optim.control = list(NULL), optim.method = "BFGS",
          estimate.variance = TRUE)
```

Arguments

param	a vector of parameter values.
x	a data matrix.
data	a data matrix.
copula	a "copula" object.
suppressMessages	logical, if TRUE, warnings messages from evaluating loglikelihood at invalid parameter values are suppressed.

method can be either "ml" (maximum likelihood), "mpl" (maximum pseudo-likelihood), "itau" (inversion of Kendall's tau), and "irho" (inversion of Spearman's rho). The last three methods assume that the "data" are pseudo-observations (scaled ranks), while the first method assumes that the "data" are observations from the unknown copula. The default method is "mpl".

start a vector of starting value for "param".

lower, upper bounds on the variables for the "L-BFGS-B" method.

optim.control a list of controls to be passed to "optim".

optim.method the method for "optim".

estimate.variance logical, if TRUE, the large-sample variance is estimated.

Value

The return value of "loglikCopula" is the loglikelihood evaluated at the given value of "param".

The return values of "fitCopula" is an object of class "fitCopula" containing slots:

estimate the estimate of the parameters.

var.est large-sample variance estimate of the parameter estimator.

method the estimation method.

loglik loglikelihood at "est".

copula the fitted copula.

Note

In the multiparameter elliptical case and when the estimation is based on Kendall's tau or Spearman's rho, the estimated correlation matrix may not always be positive-definite. If it is not, the correction proposed by Rousseeuw and Molenberghs (1993) is applied and a warning message given.

If method "mpl" in "fitCopula" is used and if "start" is not assigned a value, estimates obtained from method "itau" are used as initial values in the optimization.

For the t copula with `df.fixed=FALSE` (see [ellipCopula](#)), the methods "itau" and "irho" in "fitCopula" cannot be used. The methods "ml" and "mpl" can be employed provided the parameter "start" is assigned a value. See the example below. Also, the large-sample variance cannot be estimated if method "mpl" is chosen.

To implement the *inference functions for margins* method (see e.g. Joe 2005), the "data" need to be pseudo-observations obtained from fitted parametric marginal c.d.f.s and "method" need to be set to "ml". The returned large-sample variance will then underestimate the true variance.

Finally, note that the fitting functions generate error messages because invalid parameter values are tried during the optimization process (see [optim](#)).

References

- C. Genest (1987). Frank's family of bivariate distributions. *Biometrika* 74, 549-555.
- C. Genest and L.-P. Rivest (1993). Statistical inference procedures for bivariate Archimedean copulas. *Journal of the American Statistical Association* 88, 1034-1043.
- P. Rousseeuw and G. Molenberghs (1993). Transformation of nonpositive semidefinite correlation matrices. *Communications in Statistics: Theory and Methods* 22, 965-984.
- C. Genest, K. Ghoudi and L.-P. Rivest (1995). A semiparametric estimation procedure of dependence parameters in multivariate families of distributions. *Biometrika*, 82, 543-552.
- H. Joe (2005). Asymptotic efficiency of the two-stage estimation method for copula-based models. *Journal of Multivariate Analysis* 94, 401-419.
- S. Demarta and A. McNeil (2005). The t copula and related copulas. *International Statistical Review* 73, 111-129.
- C. Genest and A.-C. Favre (2007). Everything you always wanted to know about copula modeling but were afraid to ask. *Journal of Hydrological Engineering* 12, 347-368.
- I. Kojadinovic and J. Yan (2008), Comparison of three semiparametric methods for estimating dependence parameters in copula models, submitted.

See Also

[Copula](#), [mvdc](#), [gofCopula](#).

Examples

```
gumbel.cop <- gumbelCopula(3, dim=2)

n <- 200
x <- rcopula(gumbel.cop, n)          ## true observations
u <- apply(x, 2, rank) / (n + 1)    ## pseudo-observations
## inverting Kendall's tau
fit.tau <- fitCopula(gumbel.cop, u, method="itau")
fit.tau
## inverting Spearman's rho
fit.rho <- fitCopula(gumbel.cop, u, method="irho")
fit.rho
## maximum pseudo-likelihood
fit.mpl <- fitCopula(gumbel.cop, u, method="mpl")
fit.mpl
## maximum likelihood
fit.ml <- fitCopula(gumbel.cop, x, method="ml")
fit.ml

## a multiparameter example
normal.cop <- normalCopula(c(0.6, 0.36, 0.6), dim=3, dispstr="un")
x <- rcopula(normal.cop, n)          ## true observations
u <- apply(x, 2, rank) / (n + 1)    ## pseudo-observations
## inverting Kendall's tau
fit.tau <- fitCopula(normal.cop, u, method="itau")
fit.tau
```

```

## inverting Spearman's rho
fit.rho <- fitCopula(normal.cop, u, method="irho")
fit.rho
## maximum pseudo-likelihood
fit.mpl <- fitCopula(normal.cop, u, method="mpl")
fit.mpl
## maximum likelihood
fit.ml <- fitCopula(normal.cop, x, method="ml")
fit.ml
## with dispstr="toep"
normal.cop.toep <- normalCopula(c(0, 0), dim=3, dispstr="toep")
## inverting Kendall's tau
fit.tau <- fitCopula(normal.cop.toep, u, method="itau")
fit.tau
## inverting Spearman's rho
fit.rho <- fitCopula(normal.cop.toep, u, method="irho")
fit.rho
## maximum pseudo-likelihood
fit.mpl <- fitCopula(normal.cop.toep, u, method="mpl")
fit.mpl
## maximum likelihood
fit.ml <- fitCopula(normal.cop.toep, x, method="ml")
fit.ml
## with dispstr="ar1"
normal.cop.ar1 <- normalCopula(c(0), dim=3, dispstr="ar1")
## inverting Kendall's tau
fit.tau <- fitCopula(normal.cop.ar1, u, method="itau")
fit.tau
## inverting Spearman's rho
fit.rho <- fitCopula(normal.cop.ar1, u, method="irho")
fit.rho
## maximum pseudo-likelihood
fit.mpl <- fitCopula(normal.cop.ar1, u, method="mpl")
fit.mpl
## maximum likelihood
fit.ml <- fitCopula(normal.cop.ar1, x, method="ml")
fit.ml

## a t copula with df.fixed=FALSE
t.cop <- tCopula(c(0.2,0.4,0.6),dim=3,dispstr="un",df=5)
x <- rcopula(t.cop, n) ## true observations
u <- apply(x, 2, rank) / (n + 1) ## pseudo-observations
## maximum likelihood
fit.ml <- fitCopula(t.cop, x, method="ml", start=c(0,0,0,10))
fit.ml
## maximum pseudo-likelihood; the asymptotic variance cannot be estimated
fit.mpl <- fitCopula(t.cop, u, method="mpl", start=c(0,0,0,10),
                    estimate.variance=FALSE)

fit.mpl

```

fitCopula-class *Class "fitCopula"*

Description

Classes and summaries related to copula model fitting.

Objects from the Class

Objects can be created by calls to `fitCopula`, `fitMvdc` or to their summary method.

Slots

estimate: numeric, parameter estimate.

var.est: numeric, variance matrix estimate of the parameter estimator. See note below.

method: character, method of estimation.

loglik: numeric, loglikelihood evaluated at the maximizer.

convergence: numeric, convergence code from "optim".

nsample: numeric, integer representing the sample size.

copula: Object of class "copula".

mvdc: Object of class "mvdc".

References

C. Genest, K. Ghoudi and L.-P. Rivest (1995). A semiparametric estimation procedure of dependence parameters in multivariate families of distributions. *Biometrika*, 82, 543-552.

 fitMvdc

Estimation of multivariate models defined via copulas

Description

Fits a copula-based multivariate distribution to multivariate data.

Usage

```
loglikMvdc(param, x, mvdc, suppressMessages=FALSE)
fitMvdc(data, mvdc, start, optim.control = list(NULL), method = "BFGS")
```

Arguments

param	a vector of parameter values. When specifying parameters for mvdc objects the parameters should be ordered with the marginals first and the copula parameters last. When the mvdc object has marginsIdentical = TRUE, only the parameters of one marginal should be specified.
x	a data matrix.
mvdc	a "mvdc" object.

suppressMessages
 logical, if TRUE, warnings messages from evaluating loglikelihood at invalid parameter values are suppressed.

data
 a data matrix.

start
 a vector of starting value for "param". See "param" above for ordering of this vector.

optim.control
 a list of controls to be passed to "optim".

method
 the method for `optim`.

Value

The return value "loglikMvdc" is the loglikelihood evaluated for the given value of "param".

The return value of "fitMvdc" is an object of class "fitMvdc" containing slots:

estimate the estimate of the parameters.

var.est large-sample variance estimate of the parameter estimator.

loglik loglikelihood at "est".

copula the fitted copula.

Note

User-defined marginal distribution can be used as long as the "{dpq}" functions are defined. See `demo(QARClayton)` prepared by Roger Koenker <rkoenker@uiuc.edu>.

When covariates are available for marginal distributions or for the copula, one can construct the loglikelihood function and feed it to "optim" to estimate all the parameters.

Finally, note that the fitting functions generate error messages because invalid parameter values are tried during the optimization process (see `optim`).

See Also

[Copula](#), [fitCopula](#), [gofCopula](#).

Examples

```
gumbel.cop <- gumbelCopula(3, dim=2)
myMvd <- mvdc(gumbel.cop, c("exp", "exp"), list(list(rate=2), list(rate=4)))
x <- rmvdc(myMvd, 1000)
fit <- fitMvdc(x, myMvd, c(1,1,2))
fit

#### Roger Koenker prepared a demo illustrating MLE for a Clayton AR(1)
#### copula model with identical, uder-defined Student marginals
## demo("QARClayton")
```

generator

Generator functions for Archimedean and extreme value copulas

Description

Methods to evaluate the generator function, the inverse generator function, and derivatives of the generator function.

Usage

```
genFun(copula, u)
genInv(copula, s)
genFunDer1(copula, u)
genFunDer2(copula, u)
Afun(copula, w)
AfunDer(copula, w)
```

Arguments

`copula` an object of class "copula".
`u, s, w` numerical vector at which these functions are to be evaluated.

Details

"genFun" and "genInv" are, respectively, the generator function and its inverse function for an Archimedean copula.

"genFunDer1" and "genFunDer2" are, respectively, the 1st and 2nd derivatives of the generator function for an Archimedean copula.

"Afun" is the generator function of an extreme value copula.

"AfunDer" returns a data.frame containing the 1st and 2nd derivative of "Afun".

gofCopula

Goodness-of-fit tests for copulas

Description

Goodness-of-fit tests for copulas based on the empirical process comparing the empirical copula with a parametric estimate of the copula derived under the null hypothesis. The test statistic is the Cramer-von Mises functional S_n defined in equation (2) of Genest, Remillard and Beaudoin (2009). Approximate p-values for the test statistic can be obtained either using the *parametric bootstrap* (see the two first references) or by means of a fast *multiplier* approach (see the two last references).

Usage

```
gofCopula(copula, x, N = 1000, method = "mpl",
          simulation = "pb", print.every = 100, optim.method = "BFGS")
```

Arguments

<code>copula</code>	object of class "copula" representing the hypothesized copula family.
<code>x</code>	a data matrix that will be transformed to pseudo-observations.
<code>N</code>	number of bootstrap or multiplier iterations to be used to simulate realizations of the test statistic under the null hypothesis.
<code>method</code>	estimation method to be used to estimate the dependence parameter(s); can be either "mpl" (maximum pseudo-likelihood), "itau" (inversion of Kendall's tau) or "irho" (inversion of Spearman's rho).
<code>simulation</code>	simulation method for generating realizations of the test statistic under the null hypothesis; can be either "pb" (parametric bootstrap) or "mult" (multiplier).
<code>print.every</code>	progress is printed every "print.every" iterations. No progress is printed if it is nonpositive.
<code>optim.method</code>	the method for "optim".

Details

If the parametric bootstrap is used, the dependence parameters of the hypothesized copula family can be estimated either by maximizing the pseudo-likelihood or by inverting Kendall's tau or Spearman's rho. If the multiplier is used, any estimation method can be used in the bivariate case, but only maximum pseudo-likelihood estimation can be used in the multivariate (multiparameter) case.

For the normal and t copulas, several dependence structures can be hypothesized: "ex" for exchangeable, "ar1" for AR(1), "toep" for Toeplitz, and "un" for unstructured (see [ellipCopula](#)). For the t copula, "df.fixed" has to be set to TRUE, which implies that the degrees of freedom are not considered as a parameter to be estimated.

Thus far, the multiplier approach is implemented for six copula families: the Clayton, Gumbel, Frank, Plackett, normal and t.

Value

Returns a list whose attributes are:

<code>statistic</code>	value of the test statistic.
<code>pvalue</code>	corresponding approximate p-value.
<code>parameters</code>	estimates of the parameters for the hypothesized copula family.

References

C. Genest and B. Remillard (2008). Validity of the parametric bootstrap for goodness-of-fit testing in semiparametric models. *Annales de l'Institut Henri Poincaré: Probabilités et Statistiques*, 44, 1096-1127.

C. Genest, B. Remillard and D. Beaudoin (2009). Goodness-of-fit tests for copulas: A review and a power study. *Insurance: Mathematics and Economics*, 44, 199-214.

I. Kojadinovic and J. Yan (2009). Fast large-sample goodness-of-fit tests for copulas. Submitted.

I. Kojadinovic and J. Yan (2009). A goodness-of-fit test for multivariate multiparameter copulas based on multiplier central limit theorems. *Statistics and Computing*. In press.

See Also

[fitCopula](#), [ellipCopula](#).

Examples

```
## the following example is available in batch through
## demo(gofCopula)
## Not run:
## A two-dimensional data example
x <- rcopula(claytonCopula(3), 200)

## Does the Gumbel family seem to be a good choice?
gofCopula(gumbelCopula(1), x)
## What about the Clayton family?
gofCopula(claytonCopula(1), x)

## The same with a different estimation method
gofCopula(gumbelCopula(1), x, method="itau")
gofCopula(claytonCopula(1), x, method="itau")

## A three-dimensional example
x <- rcopula(tCopula(c(0.5, 0.6, 0.7), dim = 3, dispstr = "un"), 200)

## Does the Clayton family seem to be a good choice?
gofCopula(gumbelCopula(1, dim = 3), x)
## What about the t copula?
t.copula <- tCopula(rep(0, 3), dim = 3, dispstr = "un", df.fixed=TRUE)
gofCopula(t.copula, x)

## The same with a different estimation method
gofCopula(gumbelCopula(1, dim = 3), x, method="itau")
gofCopula(t.copula, x, method="itau")

## The same using the multiplier approach
gofCopula(gumbelCopula(1, dim = 3), x, simulation="mult")
gofCopula(t.copula, x, simulation="mult")
## End(Not run)
```

Description

Constructs an independence copula class object with its corresponding dimension.

Usage

```
indepCopula(dim = 2)
```

Arguments

`dim` the dimension of the copula.

Value

An independence copula object of class "indepCopula".

See Also

[archmCopula](#), [ellipCopula](#), [evCopula](#).

Examples

```
indep.cop <- indepCopula(3)
x <- rcopula(indep.cop, 10)
dcopula(indep.cop, x)
persp(indepCopula(), pcopula)
```

indepCopula-class *Class "indepCopula"*

Description

Independent copula class.

Objects from the Class

Objects can be created by calls of the form `new("indepCopula", ...)` or by function "indepCopula". Such objects can be useful as special cases of parametric copulas, bypassing copula-specific computations such as distribution, density, and sampler.

Slots

dimension: Object of class "numeric", dimension of the copula.

parameters: Object of class "numeric", parameter values.

param.names: Object of class "character", parameter names.

param.lowbnd: Object of class "numeric", parameter lower bounds.

param.upbnd: Object of class "numeric", parameter upper bounds.

message: Object of class "character", family names of the copula.

Methods

Afun signature(copula = "indepCopula"): ...
dcopula signature(copula = "indepCopula"): ...
pcopula signature(copula = "indepCopula"): ...
rcopula signature(copula = "indepCopula"): ...

See Also

[indepCopula](#), [copula-class](#).

indepTest	<i>Independence test among continuous random variables based on the empirical copula process</i>
-----------	--

Description

Multivariate independence test based on the empirical copula process as proposed by Christian Genest and Bruno Rémillard. The test can be seen as composed of three steps: (i) a simulation step, which consists of simulating the distribution of the test statistics under independence for the sample size under consideration; (ii) the test itself, which consists of computing the approximate p-values of the test statistics with respect to the empirical distributions obtained in step (i); and (iii) the display of a graphic, called a *dependogram*, enabling to understand the type of departure from independence, if any. More details can be found in the articles cited in the reference section.

Usage

```
indepTestSim(n, p, m = p, N = 1000, print.every = 100)
indepTest(x, d, alpha=0.05)
dependogram(test, pvalues = FALSE, print = FALSE)
```

Arguments

n	Sample size when simulating the distribution of the test statistics under independence.
p	Dimension of the data when simulating the distribution of the test statistics under independence.
m	Maximum cardinality of the subsets of variables for which a test statistic is to be computed. It makes sense to consider $m \ll p$ especially when p is large.
N	Number of repetitions when simulating under independence.
print.every	Progress is printed every "print.every" iterations. No progress is printed if it is nonpositive.
x	Data frame or data matrix containing realizations (one per line) of the random vector whose independence is to be tested.

d	Object of class "indepTestDist" as returned by the function "indepTestSim". It can be regarded as the empirical distribution of the test statistics under independence.
alpha	Significance level used in the computation of the critical values for the test statistics.
test	Object of class "indepTest" as return by the function indepTest.
pvalues	Logical indicating whether the dependogram should be drew from test statistics or the corresponding p-values.
print	Logical indicating whether details should be printed.

Details

See the references below for more details, especially the third one.

Value

The function "indepTestSim" returns an object of class "indepTestDist" whose attributes are: `sample.size`, `data.dimension`, `max.card.subsets`, `number.repetitions`, `subsets` (list of the subsets for which test statistics have been computed), `subsets.binary` (subsets in binary 'integer' notation), `dist.statistics.independence` (a N line matrix containing the values of the test statistics for each subset and each repetition) and `dist.global.statistic.independence` (a vector a length N containing the values of the global Cramér-von Mises test statistic for each repetition - see last reference p. 175).

The function "indepTest" returns an object of class "indepTest" whose attributes are: `subsets`, `statistics`, `critical.values`, `pvalues`, `fisher.pvalue` (a p-value resulting from a combination *à la* Fisher of the subset statistic p-values), `tippett.pvalue` (a p-value resulting from a combination *à la* Tippett of the subset statistic p-values), `alpha` (global significance level of the test), `beta` ($1 - \beta$ is the significance level per statistic), `global.statistic` (value of the global Cramér-von Mises statistic derived directly from the independence empirical copula process - see last reference p. 175) and `global.statistic.pvalue` (corresponding p-value).

References

- P. Deheuvels (1979). La fonction de dépendance empirique et ses propriétés: un test non paramétrique d'indépendance, *Acad. Roy. Belg. Bull. Cl. Sci.*, 5th Ser. 65:274–292.
- P. Deheuvels (1981), A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris*. 26:29–50.
- C. Genest and B. Rémillard (2004), Tests of independence and randomness based on the empirical copula process. *Test*, 13:335–369.
- C. Genest, J.-F. Quessy and B. Rémillard (2006). Local efficiency of a Cramer-von Mises test of independence, *Journal of Multivariate Analysis*, 97:274–294.
- C. Genest, J.-F. Quessy and B. Rémillard (2007), Asymptotic local efficiency of Cramér-von Mises tests for multivariate independence. *The Annals of Statistics*, 35:166–191.

See Also

[serialIndepTest](#), [multIndepTest](#), [multSerialIndepTest](#).

Examples

```
## Consider the following example taken from
## Genest and Remillard (2004), p 352:

x <- matrix(rnorm(500),100,5)
x[,1] <- abs(x[,1]) * sign(x[,2] * x[,3])
x[,5] <- x[,4]/2 + sqrt(3) * x[,5]/2

## In order to test for independence "within" x, the first step consists
## in simulating the distribution of the test statistics under
## independence for the same sample size and dimension,
## i.e. n=100 and p=5. As we are going to consider all the subsets of
## {1,...,5} whose cardinality is between 2 and 5, we set p=m=5.
## This may take a while...

d <- indepTestSim(100,5)

## The next step consists of performing the test itself:

test <- indepTest(x,d)

## Let us see the results:

test

## Display the dependogram with the details:

dependogram(test,print=TRUE)

## We could have tested for a weaker form of independence, for instance,
## by only computing statistics for subsets whose cardinality is between 2
## and 3. Consider for instance the following data:
y <- matrix(runif(500),100,5)
## and perform the test:
d <- indepTestSim(100,5,3)
test <- indepTest(y,d)
test
dependogram(test,print=TRUE)

## NB: In order to save d for future use, the save function can be used.
```

loss

LOSS and ALAE Insurance Data

Description

Indemnity payment and allocated loss adjustment expense from an insurance company.

Usage

```
data(loss)
```

Format

A data frame with 1500 observations on the following 4 variables.

loss a numeric vector of loss amount up to the `limit`.

alae a numeric vector of the allocated loss adjustment expense.

limit a numeric vector of limit (-99 means no limit).

censored 1 means censored (limit reached) and 0 otherwise.

References

Frees, E. and Valdez, E. (1998). Understanding relationships using copulas. *North American Actuarial Journal*, 2:1–25.

Examples

```
data(loss)
```

multIndepTest	<i>Independence test among continuous random vectors based on the empirical copula process</i>
---------------	--

Description

Analog of the independence test based on the empirical copula process proposed by Christian Genest and Bruno Rémillard (see [indepTest](#)) for *random vectors*. The main difference comes from the fact that critical values and p-values are obtained through the bootstrap/permutation methodology, since, here, test statistics are not distribution-free.

Usage

```
multIndepTest(x, d, m=length(d), N=1000, alpha=0.05, print.every = 100)
```

Arguments

x	Data frame or data matrix containing realizations (one per line) of the random vectors whose independence is to be tested.
d	Dimensions of the random vectors whose realizations are given in x. It is required that <code>sum(d) = ncol(x)</code> .
m	Maximum cardinality of the subsets of random vectors for which a test statistic is to be computed. It makes sense to consider $m \ll p$ especially when p is large.
N	Number of bootstrap/permutation samples.

alpha	Significance level used in the computation of the critical values for the test statistics.
print.every	Progress is printed every "print.every" iterations. No progress is printed if it is nonpositive.

Details

See the references below for more details, especially the last one.

Value

The function "multIndepTest" returns an object of class "indepTest" whose attributes are: subsets, statistics, critical.values, pvalues, fisher.pvalue (a p-value resulting from a combination *à la* Fisher of the subset statistic p-values), tippett.pvalue (a p-value resulting from a combination *à la* Tippett of the subset statistic p-values), alpha (global significance level of the test), beta (1 - beta is the significance level per statistic), global.statistic (value of the global Cramér-von Mises statistic derived directly from the independence empirical copula process - see In in the last reference) and global.statistic.pvalue (corresponding p-value).

References

- P. Deheuvels (1979). La fonction de dépendance empirique et ses propriétés: un test non paramétrique d'indépendance, *Acad. Roy. Belg. Bull. Cl. Sci.*, 5th Ser. 65:274–292.
- P. Deheuvels (1981), A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris*. 26:29–50.
- C. Genest and B. Rémillard (2004), Tests of independence and randomness based on the empirical copula process. *Test*, 13:335–369.
- C. Genest, J.-F. Quessy and B. Rémillard (2006). Local efficiency of a Cramer-von Mises test of independence, *Journal of Multivariate Analysis*, 97:274–294.
- C. Genest, J.-F. Quessy and B. Rémillard (2007), Asymptotic local efficiency of Cramér-von Mises tests for multivariate independence. *The Annals of Statistics*, 35:166–191.
- I. Kojadinovic and M. Holmes (2009), Tests of independence among continuous random vectors based on Cramér-von Mises functionals of the empirical copula process. *Journal of Multivariate Analysis*, 100:1137–1154.

See Also

[indepTest](#), [serialIndepTest](#), [multSerialIndepTest](#), [dependogram](#).

Examples

```
## Consider the following example taken from
## Kojadinovic and Holmes (2008):

n <- 100

## Generate data
```

```

y <- matrix(rnorm(6*n),n,6)
y[,1] <- y[,2]/2 + sqrt(3)/2*y[,1]
y[,3] <- y[,4]/2 + sqrt(3)/2*y[,3]
y[,5] <- y[,6]/2 + sqrt(3)/2*y[,5]

nc <- normalCopula(0.3,dim=3)
x <- cbind(y,rcopula(nc,n),rcopula(nc,n))

x[,1] <- abs(x[,1]) * sign(x[,3] * x[,5])
x[,2] <- abs(x[,2]) * sign(x[,3] * x[,5])
x[,7] <- x[,7] + x[,10]
x[,8] <- x[,8] + x[,11]
x[,9] <- x[,9] + x[,12]

## Dimensions of the random vectors
d <- c(2,2,2,3,3)

## Run the test
test <- multIndepTest(x,d)
test

## Display the dependogram
dependogram(test,print=TRUE)

```

```
multSerialIndepTest
```

Serial independence test for multivariate continuous time series based on the empirical copula process

Description

Analog of the serial independence test based on the empirical copula process proposed by Christian Genest and Bruno Rémillard (see [serialIndepTest](#)) for *multivariate* time series. The main difference comes from the fact that critical values and p-values are obtained through the bootstrap/permutation methodology, since, here, test statistics are not distribution-free.

Usage

```
multSerialIndepTest(x, lag.max, m=lag.max+1, N=1000, alpha=0.05,
  print.every = 100)
```

Arguments

x	Data frame or data matrix containing realizations the multivariate continuous time series whose serial independence is to be tested.
lag.max	Maximum lag.
m	Maximum cardinality of the subsets of 'lags' for which a test statistic is to be computed. It makes sense to consider $m \ll \text{lag.max}+1$ especially when lag.max is large.

N	Number of bootstrap/permutation samples.
alpha	Significance level used in the computation of the critical values for the test statistics.
print.every	Progress is printed every "print.every" iterations. No progress is printed if it is nonpositive.

Details

See the references below for more details, especially the last one.

Value

The function "multSerialIndepTest" returns an object of class "indepTest" whose attributes are: subsets, statistics, critical.values, pvalues, fisher.pvalue (a p-value resulting from a combination *à la* Fisher of the subset statistic p-values), tippett.pvalue (a p-value resulting from a combination *à la* Tippett of the subset statistic p-values), alpha (global significance level of the test), beta (1 - beta is the significance level per statistic), global.statistic (value of the global Cramér-von Mises statistic derived directly from the independence empirical copula process - see In in the last reference) and global.statistic.pvalue (corresponding p-value).

References

- P. Deheuvels (1979). La fonction de dépendance empirique et ses propriétés: un test non paramétrique d'indépendance, *Acad. Roy. Belg. Bull. Cl. Sci.*, 5th Ser. 65:274–292.
- P. Deheuvels (1981), A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris.* 26:29–50.
- C. Genest and B. Rémillard (2004), Tests of independence and randomness based on the empirical copula process. *Test*, 13:335–369.
- K. Ghoudi, R. Kulperger, and B. Rémillard (2001), A nonparametric test of serial independence for times series and residuals. *Journal of Multivariate Analysis*, 79:191–218.
- I. Kojadinovic and J. Yan (2008), Tests of multivariate serial independence based on a Möbius decomposition of the independence empirical copula process, *Annals of the Institute of Statistical Mathematics*, in press.

See Also

[serialIndepTest](#), [indepTest](#), [multIndepTest](#), [dependogram](#)

Examples

```
## A multivariate time series
d <- 2
n <- 100
param <- 0.25
ar <- matrix(0, 2*n, d)
ar[1,] <- rnorm(d)
for (i in 2:(2*n))
```

```

    ar[i,] <- matrix(param,d,d) %*% ar[i-1,] + rnorm(d)
x <- ar[(n+1):(2*n),]

## Run the test
test <- multSerialIndepTest(x,3)
test

## Display the dependogram
dependogram(test,print=TRUE)

```

Mvdc

Multivariate distributions constructed from copulas

Description

Density, distribution function, and random generator for a multivariate distribution via copula.

Usage

```

mvdc(copula, margins, paramMargins, marginsIdentical = FALSE)
dmvdc(mvdc, x)
pmvdc(mvdc, x)
rmvdc(mvdc, n)

```

Arguments

<code>copula</code>	an object of "copula".
<code>margins</code>	a character vector specifying all the marginal distributions. See details below.
<code>paramMargins</code>	a list whose each component is a list of named components, giving the parameter values of the marginal distributions. See details below.
<code>marginsIdentical</code>	logical variable restricting the marginal distributions to be identical.
<code>mvdc</code>	a "mvdc" object.
<code>x</code>	a vector of the copula dimension or a matrix with number of rows being the copula dimension, giving the coordinates of the points where the density or distribution function needs to be evaluated.
<code>n</code>	number of observations to be generated.

Details

The characters in argument `margins` are used to construct density, distribution, and quantile function names. For example, `norm` can be used to specify marginal distribution, because `dnorm`, `pnorm`, and `qnorm` are all available.

A user-defined distribution, for example, `fancy`, can be used as margin provided that `dfancy`, `pfancy`, and `qfancy` are available.

Each component list in argument `paramMargins` is a list with named components which are used to specify the parameters of the marginal distributions. For example, `paramMargins = list(list(mean = 0, sd = 2), list(rate = 2))` can be used to specify that the first margin is normal with mean 0 and standard deviation 2, and the second margin is exponential with rate 2.

Value

"mvdc" constructs an object of class "mvdc". "dmvdc" gives the density, "pmvdc" gives the distribution function, and "rmvdc" generates random variates.

See Also

[ellipCopula](#), [archmCopula](#), [mvdc-class](#), [copula-class](#).

Examples

```
## construct a bivariate distribution whose marginals
## are normal and exponential respectively, coupled
## together via a normal copula
x <- mvdc(normalCopula(0.75), c("norm", "exp"),
          list(list(mean = 0, sd =2), list(rate = 2)))
x.samp <- rmvdc(x, 100)
dmvdc(x, x.samp)
pmvdc(x, x.samp)
persp(x, dmvdc, xlim = c(-4, 4), ylim=c(0, 1))
```

mvdc-class

Class "mvdc"

Description

Class representing multivariate distributions constructed using Sklar's theorem.

Objects from the Class

Objects can be created by calls of the form `new("mvdc", ...)` or by function `mvdc`.

Slots

copula: Object of class "copula", specifying the copula.

margins: Object of class "character", specifying the marginal distributions.

paramMargins: Object of class "list", whose each component is a list of named components, giving the parameter values of the marginal distributions. See [mvdc](#).

marginsIdentical: Object of class "logical", that, if TRUE, restricts the marginal distributions to be identical, default is FALSE.

Methods

contour signature(x = "mvdc"): ...

persp signature(x = "mvdc"): ...

See Also

[mvdc](#).

persp-methods

Methods for function 'persp' in Package 'copula'

Description

Methods for function `persp` in package **copula**

Details

When `x` is of class `"copula"`, the following arguments can be supplied:

fun the function to be plotted, i.e., `"dcopula"` or `"pcopula"`.

n (= 51) the number of points to do the plotting.

arguments for "persp" `theta = -30`, `phi = 30`, `expand = 0.618`.

When `x` is of class `"mvdc"`, the following arguments replace the effect of `n = 51`:

xlim the range of the `x` variable.

ylim the range of the `y` variable.

Methods

x = "copula" perspective plot for a `"copula"` or `"indepCopula"` object.

x = "mvdc" perspective plot for a `"mvdc"` object.

Examples

```
persp(francCopula(-0.8), dcopula)
persp(claytonCopula(2), pcopula)
x <- mvdc(gumbelCopula(3), c("norm", "norm"),
          list(list(mean = 0, sd = 1), list(mean = 1)))
persp(x, dmvc, xlim=c(-2, 2), ylim=c(-1, 3))
persp(x, pmvc, xlim=c(-2, 2), ylim=c(-1, 3))
```

plackettCopula *Construction of a Plackett copula class object*

Description

Constructs a Plackett copula class object with its corresponding parameter.

Usage

```
plackettCopula (param)
```

Arguments

param a numeric vector specifying the parameter values.

Value

A Plackett copula object of class "plackettCopula".

References

R.L. Plackett (1965). A Class of Bivariate Distributions. *Journal of the American Statistical Association*, 60:516–522.

See Also

[ellipCopula](#), [archmCopula](#).

Examples

```
plackett.cop <- plackettCopula (param=2)
tailIndex(plackett.cop)
```

rdj *Daily Returns of Three Stocks in Dow-Jones*

Description

Five years of daily log-returns (from 1996 to 2000) of the Intel (INTC), Microsoft (MSFT) and General Electric (GE) stocks. These data were analysed in Chapter 5 of McNeil, Frey and Embrechts (2005).

Usage

```
data (rdj)
```

Format

A data frame of 1262 daily log-returns from 1996 to 2000.

DATE a character vector specifying the date

INTC daily log-return of the Intel stock

MSFT daily log-return of the Microsoft stock

GE daily log-return of the General Electric

References

A.J. McNeil, R. Frey and P. Embrechts (2005), *Quantitative risk management*, Princeton University Press, Princeton Series in Finance, New Jersey.

Examples

```
data(rdj)
```

serialIndepTest	<i>Serial independence test for continuous time series based on the empirical copula process</i>
-----------------	--

Description

Serial independence test based on the empirical copula process as proposed in Ghoudi et al. (2001) and Genest and Rémillard (2004). The test, which is the serial analog of `indepTest`, can be seen as composed of three steps: (i) a simulation step, which consists in simulating the distribution of the test statistics under serial independence for the sample size under consideration; (ii) the test itself, which consists in computing the approximate p-values of the test statistics with respect to the empirical distributions obtained in step (i); and (iii) the display of a graphic, called a *dependogram*, enabling to understand the type of departure from serial independence, if any. More details can be found in the articles cited in the reference section.

Usage

```
serialIndepTestSim(n, lag.max, m=lag.max+1, N=1000, print.every = 100)
serialIndepTest(x, d, alpha=0.05)
```

Arguments

n	Length of the time series when simulating the distribution of the test statistics under serial independence.
lag.max	Maximum lag.
m	Maximum cardinality of the subsets of 'lags' for which a test statistic is to be computed. It makes sense to consider $m \ll \text{lag.max}+1$ especially when <code>lag.max</code> is large.
N	Number of repetitions when simulating under serial independence.

<code>print.every</code>	Progress is printed every " <code>print.every</code> " iterations. No progress is printed if it is nonpositive.
<code>x</code>	Numeric vector containing the time series whose serial independence is to be tested.
<code>d</code>	Object of class <code>serialIndepTestDist</code> as returned by the function <code>serialIndepTestSim</code> . It can be regarded as the empirical distribution of the test statistics under serial independence.
<code>alpha</code>	Significance level used in the computation of the critical values for the test statistics.

Details

See the references below for more details, especially the third and fourth ones.

Value

The function "`serialIndepTestSim`" returns an object of class "`serialIndepTestDist`" whose attributes are: `sample.size`, `lag.max`, `max.card.subsets`, `number.repetitions`, `subsets` (list of the subsets for which test statistics have been computed), `subsets.binary` (subsets in binary 'integer' notation), `dist.statistics.independence` (a N line matrix containing the values of the test statistics for each subset and each repetition) and `dist.global.statistic.independence` (a vector a length N containing the values of the serial version of the global Cramér-von Mises test statistic for each repetition — see last reference p.175).

The function "`serialIndepTest`" returns an object of class "`indepTest`" whose attributes are: `subsets`, `statistics`, `critical.values`, `pvalues`, `fisher.pvalue` (a p-value resulting from a combination *à la* Fisher of the subset statistic p-values), `tippett.pvalue` (a p-value resulting from a combination *à la* Tippett of the subset statistic p-values), `alpha` (global significance level of the test), `beta` ($1 - \beta$ is the significance level per statistic), `global.statistic` (value of the global Cramér-von Mises statistic derived directly from the serial independence empirical copula process — see last reference p 175) and `global.statistic.pvalue` (corresponding p-value).

References

- P. Deheuvels (1979). La fonction de dépendance empirique et ses propriétés: un test non paramétrique d'indépendance, *Acad. Roy. Belg. Bull. Cl. Sci.*, 5th Ser. 65:274–292.
- P. Deheuvels (1981), A non parametric test for independence, *Publ. Inst. Statist. Univ. Paris*. 26:29–50.
- C. Genest and B. Rémillard (2004), Tests of independence and randomness based on the empirical copula process. *Test*, 13:335–369.
- C. Genest, J.-F. Quessy and B. Rémillard (2006). Local efficiency of a Cramer-von Mises test of independence, *Journal of Multivariate Analysis*, 97:274–294.
- C. Genest, J.-F. Quessy and B. Rémillard (2007), Asymptotic local efficiency of Cramér-von Mises tests for multivariate independence. *The Annals of Statistics*, 35:166–191.

See Also

[indepTest](#), [multIndepTest](#), [multSerialIndepTest](#), [dependogram](#)

Examples

```
## AR 1 process

ar <- numeric(200)
ar[1] <- rnorm(1)
for (i in 2:200)
  ar[i] <- 0.5 * ar[i-1] + rnorm(1)
x <- ar[101:200]

## In order to test for serial independence, the first step consists
## in simulating the distribution of the test statistics under
## serial independence for the same sample size, i.e. n=100.
## As we are going to consider lags up to 3, i.e., subsets of
## {1,...,4} whose cardinality is between 2 and 4 containing {1},
## we set lag.max=3. This may take a while...

d <- serialIndepTestSim(100,3)

## The next step consists in performing the test itself:

test <- serialIndepTest(x,d)

## Let us see the results:

test

## Display the dependogram:

dependogram(test,print=TRUE)

## NB: In order to save d for future use, the save function can be used.
```

show-methods

Methods for function 'show' in package 'copula'

Description

Methods for function show in package **copula**.

Methods

object = "copula" see [Copula](#).

object = "fitCopula" see [fitCopula](#).

object = "fitMvdc" see [fitCopula](#).

summary-methods *Methods for function 'summary' in package 'copula'*

Description

Methods for function `summary` in package **copula**.

Methods

object = "fitCopula" see [fitCopula](#).

object = "fitMvdc" see [fitCopula](#).

uranium *Uranium exploration dataset of Cook & Johnson (1986)*

Description

These data consist of 655 chemical analyses from water samples collected from the Montrose quadrangle of Western Colorado. Concentrations were measured for the following elements: uranium (U), Lithium (Li), cobalt (Co), potassium (K), cesium (Cs), scandium (Sc), and titanium (Ti).

Usage

```
data(uranium)
```

Format

A data frame with 655 observations on the following 7 variables:

U a numeric vector of uranium.

Li a numeric vector of Lithium.

Co a numeric vector of cobalt.

K a numeric vector of potassium.

Cs a numeric vector of cesium.

Sc a numeric vector of scandium.

Ti a numeric vector of titanium.

References

R. D. Cook & M. E. Johnson (1986). Generalized BurrPareto-logistic distributions with applications to a uranium exploration data set. *Technometrics*, 28:123–131.

Examples

```
data(uranium)
```

Index

*Topic **classes**

- archmCopula-class, 3
- copula-class, 8
- ellipCopula-class, 10
- evCopula-class, 12
- fgmCopula-class, 15
- fitCopula-class, 20
- indepCopula-class, 25
- mvdc-class, 34

*Topic **datasets**

- loss, 28
- rdj, 36
- uranium, 40

*Topic **distribution**

- archmCopula, 2
- Copula, 6
- ellipCopula, 9
- evCopula, 11
- fgmCopula, 14
- indepCopula, 24
- Mvdc, 33
- plackettCopula, 36

*Topic **hplot**

- contour-methods, 5
- persp-methods, 35

*Topic **htest**

- evTest, 13
- gofCopula, 22
- indepTest, 26
- multIndepTest, 29
- multSerialIndepTest, 31
- serialIndepTest, 37

*Topic **methods**

- contour-methods, 5
- generator, 22
- persp-methods, 35
- show-methods, 39
- summary-methods, 40

*Topic **models**

- fitCopula, 16

- fitMvdc, 20

- gofCopula, 22

*Topic **multivariate**

- archmCopula, 2

- AssocMeasures, 4

- Copula, 6

- ellipCopula, 9

- evCopula, 11

- evTest, 13

- fgmCopula, 14

- fitCopula, 16

- fitMvdc, 20

- generator, 22

- gofCopula, 22

- indepCopula, 24

- Mvdc, 33

- plackettCopula, 36

*Topic **print**

- show-methods, 39

- Afun (*generator*), 22

- Afun, asymCopula-method

 - (*generator*), 22

- Afun, galambosCopula-method

 - (*generator*), 22

- Afun, gumbelCopula-method

 - (*generator*), 22

- Afun, huslerReissCopula-method

 - (*generator*), 22

- Afun, indepCopula-method

 - (*generator*), 22

- Afun, tawnCopula-method

 - (*generator*), 22

- Afun, tevCopula-method

 - (*generator*), 22

- Afun-methods (*generator*), 22

- AfunDer (*generator*), 22

- AfunDer, galambosCopula-method

 - (*generator*), 22

- dcopula, claytonCopula-method
(*Copula*), 6
- dcopula, fgmCopula-method
(*Copula*), 6
- dcopula, frankCopula-method
(*Copula*), 6
- dcopula, galambosCopula-method
(*Copula*), 6
- dcopula, gumbelCopula-method
(*Copula*), 6
- dcopula, huslerReissCopula-method
(*Copula*), 6
- dcopula, indepCopula-method
(*Copula*), 6
- dcopula, normalCopula-method
(*Copula*), 6
- dcopula, plackettCopula-method
(*Copula*), 6
- dcopula, tawnCopula-method
(*Copula*), 6
- dcopula, tCopula-method (*Copula*), 6
- dcopula, tevCopula-method
(*Copula*), 6
- dependogram, 30, 32, 39
- dependogram (*indepTest*), 26
- dmvdc (*Mvdc*), 33

- ellipCopula, 2, 7, 9, 11, 17, 23–25, 34, 36
- ellipCopula-class, 8
- ellipCopula-class, 10
- evCopula, 2, 11, 12, 13, 25
- evCopula-class, 8
- evCopula-class, 12
- evTest, 13

- fgmCopula, 7, 14
- fgmCopula-class, 8, 16
- fgmCopula-class, 15
- fitCopula, 9, 14, 16, 21, 24, 39, 40
- fitCopula-class, 19
- fitMvdc, 20
- fitMvdc-class (*fitCopula-class*),
20
- frankCopula (*archmCopula*), 2
- frankCopula-class
(*archmCopula-class*), 3

- galambosCopula (*evCopula*), 11

- galambosCopula-class
(*evCopula-class*), 12
- generator, 22
- genFun (*generator*), 22
- genFun, amhCopula-method
(*generator*), 22
- genFun, claytonCopula-method
(*generator*), 22
- genFun, frankCopula-method
(*generator*), 22
- genFun, gumbelCopula-method
(*generator*), 22
- genFun-methods (*generator*), 22
- genFunDer1 (*generator*), 22
- genFunDer1, amhCopula-method
(*generator*), 22
- genFunDer1, claytonCopula-method
(*generator*), 22
- genFunDer1, frankCopula-method
(*generator*), 22
- genFunDer1, gumbelCopula-method
(*generator*), 22
- genFunDer1-methods (*generator*), 22
- genFunDer2 (*generator*), 22
- genFunDer2, amhCopula-method
(*generator*), 22
- genFunDer2, claytonCopula-method
(*generator*), 22
- genFunDer2, frankCopula-method
(*generator*), 22
- genFunDer2, gumbelCopula-method
(*generator*), 22
- genFunDer2-methods (*generator*), 22
- genInv (*generator*), 22
- genInv, amhCopula-method
(*generator*), 22
- genInv, claytonCopula-method
(*generator*), 22
- genInv, frankCopula-method
(*generator*), 22
- genInv, gumbelCopula-method
(*generator*), 22
- genInv-methods (*generator*), 22
- gofCopula, 18, 21, 22
- gumbelCopula (*archmCopula*), 2
- gumbelCopula-class
(*archmCopula-class*), 3

- huslerReissCopula (*evCopula*), 11

- huslerReissCopula-class
 - (*evCopula-class*), 12
- indepCopula, 24, 26
- indepCopula-class, 25
- indepTest, 26, 29, 30, 32, 37, 39
- indepTestSim(*indepTest*), 26
- kendallsTau(*AssocMeasures*), 4
- kendallsTau, amhCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, ANY-method
 - (*AssocMeasures*), 4
- kendallsTau, archmCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, claytonCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, copula-method
 - (*AssocMeasures*), 4
- kendallsTau, evCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, fgmCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, frankCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, galambosCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, gumbelCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, huslerReissCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, normalCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, plackettCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, tawnCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, tCopula-method
 - (*AssocMeasures*), 4
- kendallsTau, tevCopula-method
 - (*AssocMeasures*), 4
- kendallsTau-methods
 - (*AssocMeasures*), 4
- loglikCopula(*fitCopula*), 16
- loglikMvdc(*fitMvdc*), 20
- loss, 28
- multIndepTest, 28, 29, 32, 39
- multSerialIndepTest, 28, 30, 31, 39
- Mvdc, 33
- mvdc, 18, 34, 35
- mvdc(*Mvdc*), 33
- mvdc-class, 34
- mvdc-class, 34
- normalCopula(*ellipCopula*), 9
- normalCopula-class
 - (*ellipCopula-class*), 10
- optim, 17, 21, 23
- pcopula(*Copula*), 6
- pcopula, amhCopula-method
 - (*Copula*), 6
- pcopula, asymCopula-method
 - (*Copula*), 6
- pcopula, claytonCopula-method
 - (*Copula*), 6
- pcopula, fgmCopula-method
 - (*Copula*), 6
- pcopula, frankCopula-method
 - (*Copula*), 6
- pcopula, galambosCopula-method
 - (*Copula*), 6
- pcopula, gumbelCopula-method
 - (*Copula*), 6
- pcopula, huslerReissCopula-method
 - (*Copula*), 6
- pcopula, indepCopula-method
 - (*Copula*), 6
- pcopula, normalCopula-method
 - (*Copula*), 6
- pcopula, plackettCopula-method
 - (*Copula*), 6
- pcopula, tawnCopula-method
 - (*Copula*), 6
- pcopula, tCopula-method(*Copula*), 6
- pcopula, tevCopula-method
 - (*Copula*), 6
- persp, copula-method
 - (*persp-methods*), 35
- persp, indepCopula-method
 - (*persp-methods*), 35
- persp, mvdc-method
 - (*persp-methods*), 35
- persp-methods, 35
- plackettCopula, 36

- plackettCopula-class
(*copula-class*), 8
- pmvdc (*Mvdc*), 33
- rcopula (*Copula*), 6
- rcopula, amhCopula-method
(*Copula*), 6
- rcopula, asymCopula-method
(*Copula*), 6
- rcopula, claytonCopula-method
(*Copula*), 6
- rcopula, evCopula-method (*Copula*),
6
- rcopula, fgmCopula-method
(*Copula*), 6
- rcopula, frankCopula-method
(*Copula*), 6
- rcopula, galambosCopula-method
(*Copula*), 6
- rcopula, gumbelCopula-method
(*Copula*), 6
- rcopula, huslerReissCopula-method
(*Copula*), 6
- rcopula, indepCopula-method
(*Copula*), 6
- rcopula, normalCopula-method
(*Copula*), 6
- rcopula, plackettCopula-method
(*Copula*), 6
- rcopula, tCopula-method (*Copula*), 6
- rdj, 36
- rmvdc (*Mvdc*), 33
- serialIndepTest, 28, 30–32, 37
- serialIndepTestSim
(*serialIndepTest*), 37
- show, copula-method
(*show-methods*), 39
- show, fitCopula-method
(*show-methods*), 39
- show, fitMvdc-method
(*show-methods*), 39
- show, normalCopula-method
(*show-methods*), 39
- show, tCopula-method
(*show-methods*), 39
- show-methods, 39
- spearmanRho (*AssocMeasures*), 4
- spearmanRho, ANY-method
(*AssocMeasures*), 4
- spearmanRho, claytonCopula-method
(*AssocMeasures*), 4
- spearmanRho, copula-method
(*AssocMeasures*), 4
- spearmanRho, evCopula-method
(*AssocMeasures*), 4
- spearmanRho, fgmCopula-method
(*AssocMeasures*), 4
- spearmanRho, frankCopula-method
(*AssocMeasures*), 4
- spearmanRho, galambosCopula-method
(*AssocMeasures*), 4
- spearmanRho, gumbelCopula-method
(*AssocMeasures*), 4
- spearmanRho, huslerReissCopula-method
(*AssocMeasures*), 4
- spearmanRho, normalCopula-method
(*AssocMeasures*), 4
- spearmanRho, plackettCopula-method
(*AssocMeasures*), 4
- spearmanRho, tawnCopula-method
(*AssocMeasures*), 4
- spearmanRho, tCopula-method
(*AssocMeasures*), 4
- spearmanRho, tevCopula-method
(*AssocMeasures*), 4
- spearmanRho-methods
(*AssocMeasures*), 4
- summary, fitCopula-method
(*summary-methods*), 40
- summary, fitMvdc-method
(*summary-methods*), 40
- summary-methods, 40
- summaryFitCopula-class
(*fitCopula-class*), 20
- summaryFitMvdc-class
(*fitCopula-class*), 20
- tailIndex (*AssocMeasures*), 4
- tailIndex, ANY-method
(*AssocMeasures*), 4
- tailIndex, claytonCopula-method
(*AssocMeasures*), 4
- tailIndex, copula-method
(*AssocMeasures*), 4
- tailIndex, evCopula-method
(*AssocMeasures*), 4

tailIndex, frankCopula-method
 (*AssocMeasures*), 4

tailIndex, gumbelCopula-method
 (*AssocMeasures*), 4

tailIndex, normalCopula-method
 (*AssocMeasures*), 4

tailIndex, tCopula-method
 (*AssocMeasures*), 4

tailIndex-methods
 (*AssocMeasures*), 4

tCopula (*ellipCopula*), 9

tCopula-class
 (*ellipCopula-class*), 10

uranium, 40